An object-oriented tool for modeling Phase-Type distributions and a computational benchmarking of fitting algorithms

Trabajo de Tesis
presentado al
Departamento de Ingeniería Industrial

por

Juan Fernando Pérez Bernal

Asesor: Germán Riaño, Ph.D

Para optar al título de
Maestría en Ingeniería Industrial

Ingeniería Industrial
Universidad de Los Andes
Mayo 2006

An object-oriented tool for modeling Phase-Type distributions and a computational benchmarking of fitting algorithms

Aprobado por:

____

Germán Riaño, Ph.D, Asesor

____

Maria Elsa Correal, Ph.D

____

René Meziat, Ph.D

Fecha de Aprobación ________________

To my parents. To Adriana. To Julian.

Acknowledgements

I want to thank Professor Riaño for his work and advice in the development of this work.

Contents

List of Tables

List of Figures

Introduction

Phase-Type distributions are a general class of probability distributions that generalize the well known exponential distribution through the composition of exponential phases [7]. They were first introduced by Marcel Neuts [8], and have the important property of a rational Laplace transform, which makes them a subset of the distributions presented by David R. Cox [1] [9]. In this case, the extension is made as a generalization of the method of phases proposed by Erlang, and has the relevant feature of numerical tractability, as noted by Neuts [8].

The Phase-Type distributions can be defined as the time until absorption in an irreducible Markov chain with one absorbing state and all others transient. A particular realization of a Phase-Type random variable implies the random selection of an initial state, according to the initial distribution, and then jump to any other state, according to transition probability matrix (discrete case) or the generator matrix (continuous case). The process terminates when the absorbing state is reached and the value of the realization is the sum of the total time spent in all the states. Therefore, the parameters of a Phase-Type distribution are the initial probability vector and the transition matrix of a Markov Chain, which completely determine the whole process.

A relevant property of the Phase-Type distributions is its denseness, which implies that any behavior of a random variable with support in $[0, \infty)$ can be approximated through a distribution of this family [10]. As these distributions can be used as input of markovian models of real systems, this extension has several consequences in applied probability, extending the modeling capabilities to general distributions. However, an important problem to solve is the fitting of

[1]Even though the distributions proposed by Cox admit complex probabilities, the Coxian distributions treated in this documents are all related to real probabilities

the distribution parameters, which are large in number and have a non-unique representation [11]. Different approaches have been proposed to find the set of parameters of a Phase-Type distribution, including maximum likelihood methods and moment matching techniques.

In this thesis, an object-oriented tool (JPhase) is developed to model Phase-Type distributions in an computational framework, allowing the manipulation of these distributions as computational objects. The developed structure induces a formal representation of a Phase-type distribution and a set of properties that it should have. These properties are related to the computation of the probability density or mass function, the cumulative distribution function and the k-th moment, among others. Another important issue is the implementation of closure properties, which are the result of operations on the set of Phase-Type distributions.

This tool also includes two complementary packages: the first one (JPhaseGenerator), establishes the structure for any Phase-Type random variates generator, and implements the algorithms developed by Neuts and Pagano [12] for the discrete and the continuous cases. The second one (JPhaseFitter) has a set of classes to fit the parameters of a Phase-Type distribution from a data set, through the implementation of some recently developed algorithms. These classes are included in a computational structure that allows the characterization of the desired input and output of any fitting algorithm, in terms of computational objects.

With this structure, any person with a basic knowledge in object-oriented programming can use Phase-Type distributions in the analysis of a system, through the fitting of a real data set to a Phase-Type distribution and computation of some performance measures with the help of the procedures and utilities implemented. This can be seen as an alternative to simulation approaches, because it's possible to take some data, build a probabilistic model and obtain performance measures though matrix operations, avoiding some problems of simulation models, such as large replications and correlated results. Nevertheless, a graphic user interface was developed in order to allows the interaction with the tool through the familiar windows, buttons and menu bars. This interface allows an easier interaction with the user, and can be used to make an interesting analysis of a real system, including data fit, closure properties computation, and graphical presentation of the probability density function and the cumulative probability function.

As it was stated above, the parameter fitting is not an easy task and has received the attention of different researchers in the last years [3]. The wide variety of efforts makes interesting the evaluation of the algorithms developed, which is the final contribution of this work. The evaluation realized is related to the effective moments and shape fitting, and also to the computational efficiency of the algorithms. This evaluation reveals the comparative behavior of the algorithms, and it can be useful to people who find interesting the use of Phase-Type distributions in the performance analysis of real systems, but who are not willing to read the whole body of research developed around the fitting algorithms.

This document is organized as follows: in the first section the basic properties of Phase-Type distributions are presented, as well as an overview of the parameter fitting methods and the random variates generation algorithms. In section 2, 3 and 4, the developed computational structure is presented: the principal, the generator and the fitter packages. In the fifth section, the benchmark of fitting algorithms is presented, including the methodology and the results. Finally, some conclusions are stated in the last section.

Chapter I

Phase-Type Distributions

In this section, the definition and basic properties of Phase-Type distributions are stated, according to the treatment presented in [8] y [13]. Therefore, the proofs of the definitions in this section are not included and the interested reader can find them in the given references.

1.1 Continuous Phase-Type Distributions

A Continuous Phase-Type distribution can be defined as the time until absorption in a Continuous Markov Chain, with one absorbing state and all others transient. The generator matrix of that process can be written as:

$$\mathbf{Q} = \begin{bmatrix} 0 & \mathbf{0} \\ \mathbf{a} & \mathbf{A} \end{bmatrix},$$

where the first entry in the state space represents the absorbing state. As the sum of the elements on each row must be equal to zero, $\mathbf{a}$ is determined by:

$$\mathbf{a} = -\mathbf{A1},$$

where $\mathbf{1}$ is a vector of ones. In order to completely determine the process, the initial probability distribution is defined and can be partitioned in the same way of the generator matrix:

$$\begin{bmatrix} \alpha_0 & \boldsymbol{\alpha} \end{bmatrix},$$

where α_0 is the probability of starting the process in the state 0, and the sum of all the components in the vector must be equal to 1. Therefore, α_0 is determined by the following relationship:

$$\alpha_{\mathbf{0}} = 1 - \boldsymbol{\alpha}\mathbf{1}.$$

In this way, a Continuous Phase-Type Distribution is completely determined by the parameters (α, A), and its probability distribution function is defined as:

$$F(x) = 1 - \boldsymbol{\alpha} e^{\mathbf{A}x}\mathbf{1}, \quad x \geq 0,$$

which has a clear connection to the well known exponential distribution. Furthermore, if there is just one transient phase with associate rate λ and it is selected with probability one, then the distribution is exactly the exponential case. From the previous expression, the probability density function can be computed as:

$$f(x) = \boldsymbol{\alpha} e^{\mathbf{A}x}\mathbf{a}, \quad x > 0.$$

And similarly, the Laplace-Stieltjes transform of $f(\cdot)$, is given by:

$$f(s) = \alpha_0 + \boldsymbol{\alpha}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{a}, \quad Re(s) \geq 0,$$

from which, the non-centered moments can be calculated as:

$$E[X^k] = k!\boldsymbol{\alpha}(-\mathbf{A}^{-1})^k\mathbf{1}, \quad k \geq 1.$$

1.2 Discrete Phase-Type Distributions

A Discrete Phase-Type distribution can be seen as an analogous case to the continuous distribution. In this case, the distribution can be defined as the number of steps until absorption in a Discrete Markov Chain, with one absorbing state and all other transient. The transition probability matrix of that process may be defined as:

$$\mathbf{P} = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{a} & \mathbf{A} \end{bmatrix},$$

where the first row in the matrix represents the absorbing state. As the sum of the elements in every row of the matrix must equal to one (in order to be a probability mass function), $\mathbf{t}$ is determined by:

$$\mathbf{a} = \mathbf{1} - \mathbf{A}\mathbf{1}.$$

Similarly, the initial probability distribution is defined as:

$$\begin{bmatrix} \alpha_0 & \boldsymbol{\alpha} \end{bmatrix},$$

where $\alpha_0 = 1 - \boldsymbol{\alpha}\mathbf{1}$ is the probability of starting the process in the absorbing state, i.e. the number of steps in that case would be equal to zero. As before, the discrete distribution is completely determined by the parameters (α, T) and its probability mass function is defined as:

$$P(X = k) = \begin{cases} \alpha_0 & , k = 0 \\ \boldsymbol{\alpha}\mathbf{A}^k\mathbf{a} & , k \geq 1 \end{cases}$$

This last definition makes natural the definition of the cumulative probability function of the discrete Phase-Type variable:

$$P(X \leq k) = 1 - \boldsymbol{\alpha}\mathbf{A}^k\mathbf{1}, \quad k \geq 0.$$

Also, the moment generating function, or Z-transform, can be calculated from $P(\cdot)$:

$$G(z) = \alpha_0 + z\boldsymbol{\alpha}(\mathbf{I} - z\mathbf{A})^{-1}\mathbf{a}, \quad |z| \leq 1.$$

from which, the factorial moments of the distribution can be calculated as:

$$E[X(X-1)(X-2)\dots(X-k+1)] = k!\boldsymbol{\alpha}(\mathbf{I} - \mathbf{A})^{-k}\mathbf{A}^{k-1}\mathbf{1}, \quad k \geq 1.$$

1.3 Closure Properties

An important issue of Phase-Type distributions is that they are closed under some operations, which can be useful in the analysis of some systems. The following closure properties are valid for both discrete and continuous distributions.

1. **Convolution of a finite number of Phase-Type distributions**
   If $X \sim PH(\alpha, T)$ and $Y \sim PH(\beta, S)$, with n and m phases respectively, then the convolution $X + Y \sim PH(\gamma, C)$ has $m + n$ phases, with

$$\boldsymbol{\gamma} = [\boldsymbol{\alpha}, \alpha_0\boldsymbol{\beta}] \qquad \text{and} \qquad \mathbf{C} = \begin{bmatrix} \mathbf{T} & \mathbf{t}\boldsymbol{\beta} \\ \mathbf{0} & \mathbf{S} \end{bmatrix}.$$

2. **Convex mixture of a finite number of Phase-Type distributions**
   If $X \sim PH(\alpha, T)$ and $Y \sim PH(\beta, S)$, with n and m phases respectively, and distribution functions $F(\cdot)$ and $G(\cdot)$. Then the convex mixture $\theta F(\cdot) + (1-\theta)G(\cdot)$, with $0 \leq \theta \leq 1$, has representation $PH(\gamma, C)$ with $m + n$ phases, where

$$\boldsymbol{\gamma} = [\theta\boldsymbol{\alpha}, (1-\theta)\boldsymbol{\beta}] \qquad \text{and} \qquad \mathbf{C} = \begin{bmatrix} \mathbf{T} & \mathbf{0} \\ \mathbf{0} & \mathbf{S} \end{bmatrix}.$$

3. **Convolution of a discrete Phase-Type number of Phase-Type distributions**
   If X_i are i.i.d. continuous Phase-Type distribution with representation $PH(\alpha, T)$ and N a discrete Phase-Type distribution with representation $PH(\beta, S)$, then the mixture $G(\cdot) = \sum_{k=0}^{N} X_i$ is $PH(\gamma, C)$, with

$$\boldsymbol{\gamma} = \boldsymbol{\alpha} \otimes \boldsymbol{\beta} \qquad \text{and} \qquad \mathbf{C} = \mathbf{T} \otimes \mathbf{I} + \mathbf{t}\boldsymbol{\alpha} \otimes \mathbf{S}.$$

The function $\otimes$ denotes the Kronecker product and $\oplus$ the Kronecker sum [1] .

- Convolution of a geometric number of Phase-Type distributions
  As the geometric distribution is a particular case of Discrete Phase-Type distributions, this property also holds for the geometric case. If X_i are i.i.d. continuous Phase-Type distribution with representation $PH(\alpha, T)$ and N is geometric distributions with parameter p, then the mixture $G(\cdot) = \sum_{k=0}^{N} X_i$ is $PH(\gamma, C)$, with

$$\boldsymbol{\gamma} = \boldsymbol{\alpha} \qquad \text{and} \qquad \mathbf{C} = \mathbf{T} + (1-p)\mathbf{t}\boldsymbol{\alpha}.$$

4. **The minimum of a set of Phase-Type distributions**
   If $X \sim PH(\alpha, T)$ and $Y \sim PH(\beta, S)$, with n and m phases respectively, then $min(X, Y) \sim PH(\gamma, C)$ with mn phases and

$$\boldsymbol{\gamma} = \boldsymbol{\alpha} \otimes \boldsymbol{\beta}.$$

   In this case, the matrix C has a different definition if the process is discrete or continuous. In the discrete case, the resulting probability transition matrix is given by

$$\mathbf{C} = \mathbf{T} \otimes \mathbf{S}.$$

   For the continuous case, the generator matrix is given by

$$\mathbf{C} = \mathbf{T} \oplus \mathbf{S}.$$

5. **The maximum of a set of Phase-Type distributions**
   If $X \sim PH(\alpha, T)$ and $Y \sim PH(\beta, S)$, with n and m phases respectively, then $max(X, Y) \sim PH(\gamma, C)$ with $mn + n + m$ phases and

$$\boldsymbol{\gamma} = [\boldsymbol{\alpha} \otimes \boldsymbol{\beta}, \beta_0\boldsymbol{\alpha}, \alpha_0\boldsymbol{\beta}] \qquad \text{and} \qquad \mathbf{C} = \begin{bmatrix} \mathbf{T} \oplus \mathbf{S} & \mathbf{I} \otimes \mathbf{s} & \mathbf{t} \otimes \mathbf{I} \\ \mathbf{0} & \mathbf{T} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{S} \end{bmatrix}.$$

[1] The Kronecker product of matrices **A** and **B** is defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \dots & a_{1n}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \dots & a_{2n}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & a_{m2}\mathbf{B} & \dots & a_{mn}\mathbf{B} \end{bmatrix}$$

And the Kronecker sum of matrices **A** and **B** is defined as $\mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{B}$

1.4 Further Closure Properties for Continuous Distributions

There are some other important closure properties that only apply for the case of Continuous Phase-Type distributions, which are listed below.

1. **Waiting time in a $M/PH/1$ queue**
   If $X \sim PH(\alpha, T)$ is the service time distribution in a $M/G/1$ queue, then the distribution of the waiting time $W(\cdot)$ is $PH(\gamma, C)$, with

   $$\boldsymbol{\gamma} = (1-\rho)\boldsymbol{\pi} \qquad \text{and} \qquad \mathbf{C} = \mathbf{T} + \rho\mathbf{t}\boldsymbol{\pi},$$

   where $\rho = \lambda m$ is the traffic coefficient, λ is the arrival rate and m is the expected value of the service time. $\boldsymbol{\pi}$ is the stationary probability vector of $\mathbf{T}+\mathbf{t}\boldsymbol{\alpha}$, i.e. $\boldsymbol{\pi} = (\boldsymbol{\alpha}\mathbf{T}^{-1}\mathbf{e})\boldsymbol{\alpha}\mathbf{T}^{-1}$.

2. **Residual time distribution**
   If $X \sim PH(\alpha, T)$, then the residual time distribution

   $$G(x) = \mathrm{P}(X - \tau \leq x | X > \tau)$$

   has representation $PH(\gamma, T)$, with

   $$\boldsymbol{\gamma} = \frac{1}{1-F(\tau)}\boldsymbol{\alpha}e^{\mathbf{A}\tau}.$$

3. **Equilibrium Residual time distribution**
   If $X \sim PH(\alpha, T)$, then the equilibrium residual time distribution

   $$G(x) = \frac{1}{E[X]}\int_0^x (1-F(u))du,$$

   has representation $PH(\pi, T)$, where $\boldsymbol{\pi}$ has the same meaning as state above.

4. **Termination time of a Phase-Type process with Phase-Type failures [14]**
   Consider a process where the service time is determined by Phase-Type distribution with m phases and representation $PH(\alpha, T)$, and it is subject to failures that occur according to a Poisson process with rate λ. If the duration of the failure is $PH(\beta, S)$ with n phases, then the completion time has distribution $G(\cdot)$ with representation $PH(\gamma, C)$. Two different cases must be differentiated: if the service must be restarted after the failure, or if the task can begin from the point where it was left before the failure. In the first case, the resulting distribution has $m+n$ phases and

   $$\boldsymbol{\gamma} = [\boldsymbol{\alpha}, \mathbf{0}] \qquad \text{and} \qquad \mathbf{C} = \begin{bmatrix} \mathbf{T} - \mu\mathbf{I} & \mu\mathbf{e}\boldsymbol{\beta} \\ \mathbf{s}\boldsymbol{\alpha} & \mathbf{S} \end{bmatrix}.$$

   In the second case, the representation has $m + mn$ phases and

$$\boldsymbol{\gamma} = [\boldsymbol{\alpha}, \mathbf{0}] \qquad \text{and} \qquad \mathbf{C} = \begin{bmatrix} \mathbf{T} - \mu\mathbf{I} & \mu\mathbf{I} \otimes \boldsymbol{\beta} \\ \mathbf{I} \otimes \mathbf{s} & \mathbf{I} \otimes \mathbf{S} \end{bmatrix}.$$

1.5 Denseness Property

The Continuous and Discrete Phase-Type distributions have the important property of being dense in $[0, \infty)$ and the non-negative integers, respectively [2] . This implies that any distribution with support on those sets can be approximated by a Phase-Type distribution with the appropriate number of phases and parameters $\boldsymbol{\alpha}$ and $\mathbf{T}$. In this sense, any non-negative behavior could be represented through these distributions, but this is not completely true because the number of phases needed could be infinite, which is computationally nonviable.

1.6 Phase-Type Random Variates Generation

In many large applications, simulation is the appropriate tool to model the system because of the complex relations between different stochastic variables. This makes that a random number generator become an important tool to model a wide range of non-deterministic systems. Neuts and Pagano [12] developed two similar algorithms to generate random variates from discrete and continuous Phase-Type distributions. These algorithms are supported on the alias method to generate variates from discrete distributions in order to simulate the process of selecting an initial state and then jump to the next one according to random vectors.

1.7 Fitting Algorithms

In the last twenty years, the problem of fitting the parameters of a Phase-Type distribution has received great attention from the applied probability community. The relevance of the problem relies in the wide range of applications that comes from the relaxation of the exponential assumption. This allows the inclusion of different behaviors in the markovian framework, such as long tails, self-similarity [16] and fractality [17].

These different approaches can be classified in two major groups: maximum likelihood methods and moment matching techniques, as noted in [18]. Nevertheless, almost all the algorithms designed for this task have an important characteristic in common: they reduce they set of distributions to be fitted, from the whole Phase-Type set to a special subset. In the next section, some of the existing algorithms will be listed with a brief description. In section 4, those algorithms included in the computational package will be revisited and further explained.

[2]The proof of this property can be found in [15]

1.7.1 Moment Matching Algorithms

The moment matching techniques are the most studied algorithms to fit the parameters of Phase-Type distributions. The first effort is probably due to Sauer and Chandy [19], who fit any first two moments with a two phase hyper-exponential (for the case of squared coefficient of variation[3] greater than 1), or a generalized Erlang (for C_x^2 between 0 and 1). Other approach is proposed in [20], where the first two moments are matched through a Cox distribution, which in the case of $C_x^2 \geq 1$ has two phases, and in the case $0 < C_x^2 < 1$ becomes an Erlang distribution.

An algorithm to tackle the problem of matching three moments was first proposed by Johnson and Taaffe [21], who used a subset of the Phase-Type distributions known as Hyper-Erlang distributions, which is a convex mixture of Erlang distributions. There they give formulas to calculate the parameters of a mixture of 2 Erlangs, in order to fit a set of first three moments. A problem with the obtained distributions is the very large number of phases required. This paper was followed by [22], where some nonlinear programming techniques are used to fit the parameters of a Hyper-Erlang and a Coxian distribution. In a companying paper [10], Johnson and Taaffe analyze the shapes of the densities obtained by their proposed algorithms. As a result of these and other papers of the authors [23] and [24], they developed a software tool for fitting the parameters of Phase-Type distributions known as MEFIT. A close effort in this directions was made by Schmickler [25], who present an algorithm to fit the parameters of a mixture of Erlang distributions using a nonlinear algorithm to solve a set of moment matching nonlinear equations. The implementation of this technique is done in a computational tool known as MEDA. This two different approaches were analyzed in [18] and will not be discussed elsewhere in this document.

In recent years, some approaches have been developed, beginning by the one presented by Telek and Heindl [4] in 2002, who worked with acyclic Phase-Type distributions of second order (two phases). Then in 2003 Osogami and Harchol published a series of papers [26] [27] [5] where they characterizes the bounds imposed over the first three moments representable by Erlang-Coxian distributions, a subset of Phase-Type distributions introduced by them. Afterwards Bobbio, Horvath and Telek [6] presented in 2005 an algorithm to match first three moments with acyclic Phase-Type distributions with the minimal number of phases needed to do it. These algorithms will be further illustrated in section 4 as well as they implementation in the framework.

[3]The squared coefficient of variation is defined as $C_x^2 = \frac{V(x)}{E(x)^2}$

1.7.2 Maximum Likelihood Algorithms

In contrast to moment matching approaches, maximum likelihood algorithms for fitting Phase-Type distributions have been recently developed. The first approach in this way was developed by Bobbio and Cumani [28], and some results about its computational behavior can be found in [29]. The algorithm is built over the subset of Acyclic Phase-Type distributions, which have an upper triangular generator matrix and a canonical form [30]. For the maximization of the log-likelihood function, the algorithm chooses an initial point and linearizes its neighborhood, then solves a linear programming problem and the solution is the initial point for the next iteration. Because this algorithm has been extensively studied in [18] and [29], it will not be discussed further in this document.

Another approach has come from the use of the EM algorithm proposed by Dempster et. al. [31], to fit the parameters of a Phase-Type distribution. The first one is due to Asmussen et. al. [1] and is the only one that faces the problem of fitting the whole set of Phase-Type distributions. Other approximations have been done by Khayari et. al. [2], who use the EM algorithm to fit the parameters of a hyper-exponential distribution, and by Thümmler et. al.[3], who work with set subset of hyper-Erlang distributions, that are also dense in $[0, \infty)$.

1.7.3 Other approaches

Other important approach is the one developed by Feldmann and Whitt [16], which is centered in the approximation of theoretical density functions with long tails through hyper-exponential distributions. To apply it for data set, it is necessary to make an intermediate step, in which the data must be fitted to a long-tailed distribution function, as the well-known Weibull and Pareto distributions. This imposes a restriction in the applicability of the method that was noted by [2]. Finally, the approach developed by Riska et. al.[32] can be seen as an extension of the one proposed by Khayari [2], in the sense that it uses the EM algorithm to fit long-tailed data sets to hyper-exponential distributions. Nevertheless, they tackle the problem by splitting the data set in several subsets that have a Squared Coefficient of Variation in a predetermined range. In this way, the algorithm does not obtain a global optimal distribution but a set of distributions that are finally mixed in a single one. This approach allows that the tail and the body of the distributions have different treatments and the distribution obtained captures the whole behavior of the data set. They finally improve the algorithm by adding an initial Erlang distribution in order to capture non-monotone decreasing probability density functions, but the parameter fit for the Erlang distribution is made by matching the the first two moments of the data subset that has the non-monotone decreasing behavior.

Even though the algorithms described are all for the case of continuous Phase-Type distributions, some important efforts have been done to solve the analogous problem for the discrete case. One of the approaches was made for discrete distribution of second order[4], as an analog of the presented above. The other one was presented in [33], where a canonical form for acyclic discrete distributions is constructed as well as an algorithm designed for fitting the distribution parameters of this class from a data set.

Chapter II

jPhase: the Object-Oriented Framework

One of the contributions of this work is the design and implementation of an object-oriented framework that allows the computational manipulation Phase-Type distributions. To date, there is no academic or commercial software that can offer the capabilities of representation nor manipulation of Phase-Type distribution in a unified fashion. For example, with the developed tool, it is possible to create an object that represents a continuous Phase-Type distribution, calculate the value of its probability density function (pdf) or its cumulative distribution function (cdf), as well as any power moment. It is also possible to compute the minimum or the maximum between two distributions, as well as other closure properties, e.g. the distribution of the waiting time in a $M/PH/1$ queue.

Now, some of the most important issues about the computational structure will be discussed in order to give a good understanding of the framework. It must be said that the computational architecture is divided in three gross packages: `jPhase`, `jPhaseGenerator` and `jPhaseFit`. The first is related to the computational representation of Phase-Type distributions and will be explained in this section. The second builds the structure to implement Phase-Type random variates generators and will be discussed in section 3. The last one offers a computational representation of Phase-Type fitting algorithms and will be explained in detail in section 4. The first package can be seen as the heart of the whole framework and the others are supported on it.

2.1 General Structure

The `jPhase` package is supported on a set of Interfaces, Abstract Classes and Concrete Classes. The Interfaces determine the characteristics of an object and have no implementation of any method. As can be seen in the simple Class Diagram of Figure 1, there are three Interfaces in the JPhase package: `PhaseVar`, `ContPhaseVar`, and `DiscPhaseVar`. These Interfaces determines the behavior of a PhaseType distribution in both the continuous and discrete cases.

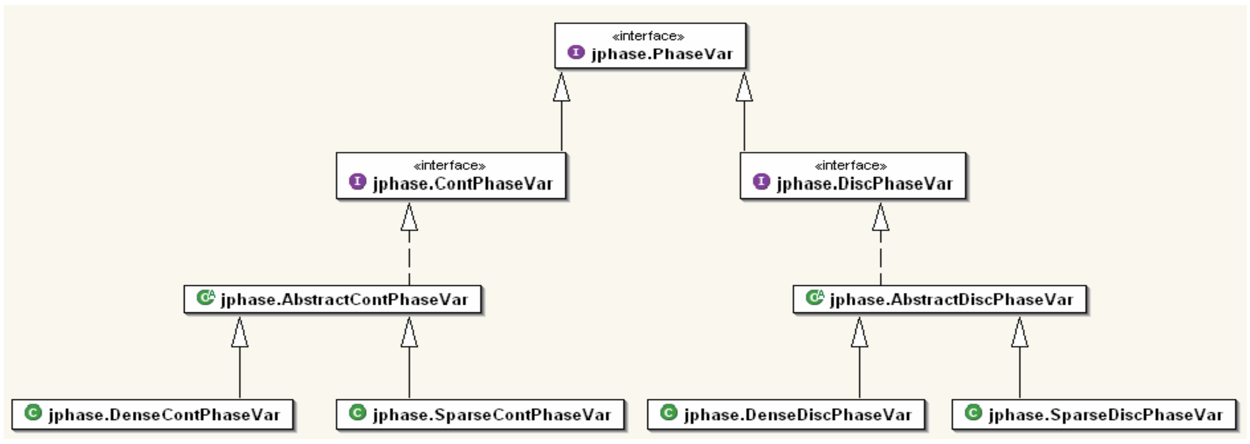

Figure 1: Simple jPhase Package Class Diagram

The Abstract Classes `AbstractContPhaseVar` and `AbstractDiscPhaseVar` implements the corresponding Interface (discrete or continuous), in order to develop some of the methods determined by the Interfaces. Finally, the Concrete Classes extends the corresponding Abstract class, and thus they make use of the already implemented methods. These methods are useful for any user that wants to develop an own Concrete Class, because he or she doesn't need to get worried about the whole set of distribution properties, but only needs to implement a little set of simple methods. In the next sections, the properties of these Interfaces, Abstract and Concrete classes will be explained.

2.2 Interfaces

As it was said above, the jPhase package consists of three interfaces, that determine the behavior of any Phase-Type distribution as shown next.

- `PhaseVar`
  This interface defines the set of properties that are common to both discrete and continuous Phase-type distributions. Since this is the core Interface in the framework, it has the major quantity of methods and all other Interfaces ans Classes have fewer. The methods that the Interface force to implement for any distribution can be divided in three groups: access, moments and distribution methods.

- `DiscPhaseVar` and `ContPhaseVar`
  This interfaces determine some of the closure properties valid for discrete and continuous Phase-Type variables, as those discussed in section 1. The methods defined by each one of this interfaces can be partitioned in two groups: distribution and closure methods. The closure properties can only be defined at this level because each one of the discrete and

Table 1: Some methods for the `PhaseVar` interface

Type	Method	Result
Access methods	`getMatrix()`	Generator matrix $\mathbf{A}$
	`setMatrix(`A`)`	Set the transition matrix equal to the parameter
	`getVector()`	Initial probability distribution vector $\boldsymbol{\alpha}$
	`setVector(`α`)`	Set the initial probability vector equal to the parameter
	`getNumPhases()`	Number of transient phases in the distribution
	`getVec0()`	Value of α_0
	`getMat0()`	Exit rate vector $\mathbf{a} = -\mathbf{A1}$
	`copy()`	Deep copy of the distribution
Moments methods	`expectedValue()`	Expected value of the distribution
	`variance()`	Variance of the distribution
	`stdDeviation()`	Returns the standard deviation.
	`CV()`	Squared coefficient of Variance.
	`moment(`k`)`	k-th non-central moment of the distribution.
Distribution methods	`cdf(`x`)`	Cumulative distribution function at x
	`prob(`a`, `b`)`	Probability that the variable takes a value between a and b
	`survival(`x`)`	Survival function at x
	`lossFunction1(`x`)`	Value of the order-one loss function evaluated at x
	`lossFunction2(`x`)`	Value of the order-two loss function evaluated at x
	`quantil(`x`)`	Quantil x of the distribution
	`median()`	Median of the distribution

continuous sets are closed under these properties, but not the whole set of Phase-Type distributions. Some of the methods defined by the interfaces are shown in Table 2, where all but the distribution-related methods apply for both cases. Next, in Table 3 some other methods are shown, but they are only defined for the continuous class, as discussed in section 1.4.

Table 2: Some methods for the `DiscPhaseVar` and `ContPhaseVar` interface

Type	Method	Result
Distribution methods	`pmf(`x`)` or `pdf(`x`)`	Value of the probability mass function at x (discrete case) or the probability density function (continuous case)
Closure methods	`sum(`Y`)`	Convolution between the original distribution and Y
	`sumGeom(`p`)`	Sum of a geometric number (with parameter p) of i.i.d. Phase-Type distributions as the original one
	`sumPH(`Y`)`	Convolution of a discrete Phase-Type (Y) number of i.i.d. Phase-Type distributions
	`mix(`p`,` Y`)`	Convex mixture between the original distribution (with weight p) and Y
	`min(`Y`)`	Minimum between the original distribution and Y
	`max(`Y`)`	Maximum between the original distribution and Y
Other methods	`newVar(`n`)`	New n phase variable with the same representation as the original
	`toString()`	Returns a string representation of the Phase-Type distribution (including its associated vector and the matrix)

Table 3: Some further closure methods for the `ContPhaseVar` interface

Method	Result
`times(`k`)`	Distribution of the variable scaled by k
`residualTime(`x`)`	Distribution of the residual time at x
`eqResidualTime()`	Distribution of the equilibrium residual time
`waitingQ(`ρ`)`	Distribution of the waiting time in a $M/PH/1$ queue with traffic coefficient equal to ρ

2.3 Abstract Classes

As shown in Figure 1, the `ContPhaseVar` interface is implemented by the abstract class `Abstract-ContPhaseVar`, which implements almost all the methods defined by `PhaseVar` and `Cont-PhaseVar`. In particular, none of the methods implemented by this class depends on the formal representation of the matrices and vectors involved. This means that all the operations are executed using solvers and preconditioners that apply for both sparse and dense representations of matrices and vectors. Moreover the probably most difficult routines are solved by this abstract class, such as the computation of the probability density function, that implies the use of uniformization methods for solve a set of differential equations[13]. The same arguments apply for the abstract class `AbstractDiscPhaseVar`, that implements the interface `DiscPhaseVar`.

This way, the only methods that the user must implement when developing a Concrete Class that extends `AbstractContPhaseVar` or `AbstractDiscPhaseVar` are:

- getMatrix and setMatrix
- getVector and setVector
- newVar
- copy

As can be seen, this methods depend on the particular representation of the distribution, e.g. if the matrix is represented by a particular sparse pattern, then the only one class of matrices that can be set must have the same pattern. Also the `newVar` and `copy` methods must return a variable that belongs to the same class of the original one. The Concrete classes explained in the next section are themselves examples of classes that extend the abstract ones.

2.4 Concrete Classes

The developed concrete classes are those that are a final user will usually utilize. They have been designed as general Phase-Type representations for the continuous and discrete cases, and with dense and sparse storage. The `DenseContPhaseVar` and `DenseDiscPhaseVar` are classes that represent continuous and discrete Phase-Type distributions, using the `DenseMatrix` and `DenseVector` classes defined by MTJ. This classes are useful for many applications, where the number of phases is not large and the memory is not a problem. They also have constructors for many simple distributions such as exponential or Erlang in the continuous case, and geometric or negative binomial in the discrete case.

Nevertheless, the use of matrices with dense representation can be a problem because of the large number of phases. The `SparseContPhaseVar` and `SparseDiscPhaseVar` classes are built over the `FlexCompRowMatrix` and `SparseVector` MTJ classes, which give a good alternative when the number of phases is large but the number of entries is little relative to the total number of n^2 entries. It is important to note that the `FlexCompRowMatrix` allows a flexible sparse pattern stored by rows, that makes of this class a general sparse representation. Other specific representation could be developed by using a particular sparse pattern, e.g. upperdiagonal matrices.

2.5 Examples

In order to give a closer understanding of jPhase, some examples will be given to clarify the construction and manipulation of the computational objects. As shown in Figure 2, the distributions can be created from arrays of doubles, that represent the initial probability vector and the generator matrix of the transient states (as specified in section 1). Once the distributions are created, they can be manipulated through the use of closure properties, as shown in Figure 2, where the convolution between the variables $v1$ and $v2$ is calculated.

File `example1.java`

```java
double[][] A = new double[][] { {-2,2} , {2,-5} } ;
double[] alpha = new double[] {0.2,0.4};
DenseContPhaseVar v1 = new DenseContPhaseVar(alpha , A);

double[][] B = new double[][] { {-4,2,1} , {1,-3,1} , {2, 1,-5} } ;
double[] beta = new double[] {0.1, 0.2, 0.2};
DenseContPhaseVar v2 = new DenseContPhaseVar(beta , B);

ContPhaseVar v3 = v1.sum(v2);
System.out.println("v3: "+v3.toString());
```

Figure 2: jPhase: Example 1

The resulting variable from the precious code has the usual representation, which includes the initial probability vector $\boldsymbol{\alpha}$ and the transition matrix $\mathbf{T}$, as explained in section 1. The result from the former example is shown next, where the calculated variable is printed.

File `resExample1.txt`

```
v3:
-------------------------------------------------
Phase-Type Distribution
Number of Phases: 5
Vector:
        0.2000   0.4000   0.0400   0.0800   0.0800
Matrix:
        -2.0000   2.0000   0.0000   0.0000   0.0000
         2.0000  -5.0000   0.3000   0.6000   0.6000
         0.0000   0.0000  -4.0000   2.0000   1.0000
         0.0000   0.0000   1.0000  -3.0000   1.0000
         0.0000   0.0000   2.0000   1.0000  -5.0000
-------------------------------------------------
```

Figure 3: jPhase: Result for Example 1

Since jPhase is built over the Matrix Toolkit for Java (MTJ) library [34], it is also possible to construct Phase-Type distributions from matrices and vectors defined in that library. As can be seen in the following example, the matrix and the vector of the Phase-Type distribution are first built as `DenseMatrix` and `DenseVector` (MTJ objects), and then the continuous Phase-Type distribution is constructed.

File `example2.java`

```
DenseMatrix A = new DenseMatrix(
                        new double[][] { {-4,2,1} , {1,-3,1} , {2, 1,-5} } );
DenseVector alpha = new DenseVector(new double[] {0.1 , 0.2 , 0.2});

DenseContPhaseVar v1 = new DenseContPhaseVar(alpha , A);

double rho = 0.5;
PhaseVar v2 = v1.waitingQ(rho);
System.out.println("v2:\n"+v2.toString());
```

Figure 4: jPhase: Example 2

In the previous example, the distribution of the waiting time in queue is computed taking the variable $v1$ as the service time distribution and assuming that the traffic coefficient of the $M/PH/1$ queue is equal to 0.5. The resulting distribution is then printed and the output is shown next.

File `resExample2.txt`

```
v2:
-------------------------------------------------
Phase-Type Distribution
Number of Phases: 3
Vector:
        0.1500   0.2250   0.1250
Matrix:
        -3.8500   2.2250   1.1250
         1.1500  -2.7750   1.1250
         2.3000   1.4500  -4.7500
-------------------------------------------------
```

Figure 5: jPhase: Result for Example 2

Another way to do the former calculations is through the use of the Graphic User Interface (GUI). This can be used to build Phase-Type variables from direct input, or from a data set that can be fit the parameters of the distribution. It also allows to compute closure properties and has the capabilities to show graphically the probability density function or the cumulative probability distribution of a specified Phase-Type distribution. A sample screenshot of the developed GUI is shown in Figure 6.

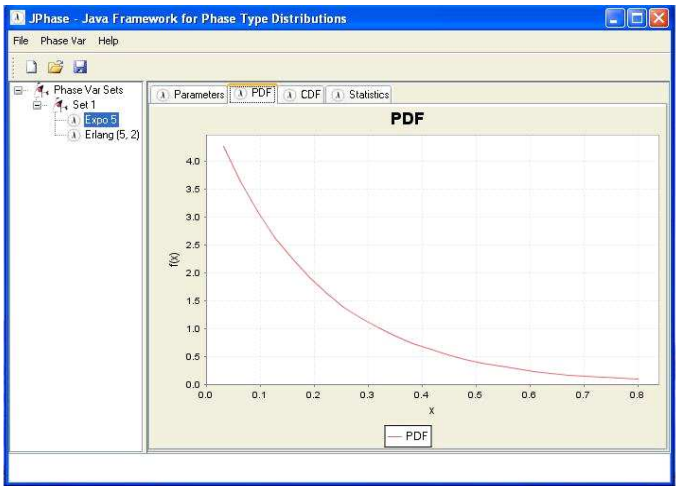

Figure 6: jPhase: Graphic User Interface

As can be seen, the developed framework is an easy way to deal with Phase-Type distributions and can be used as a supporting tool in several practical researches, where the main point is to build a probabilistic model that describes the system, and the Phase-Type distributions are an important tool to do it. Thus, the researcher can focus on the modeling issue based on the computational representation developed in this work.

Chapter III

jPhaseGenerator: the variates generator module

This package was developed in order to define the behavior of any Phase-Type random variates generator. This behavior is specified by the Abstract Class `PhaseGenerator`, which is the core the package. As can be seen in Figure 7, this abstract class is extended by the concrete classes `NeutsContPHGenerator` and `NeutsDiscPHGenerator`, that implement the algorithms proposed by Neuts and Pagano [12].

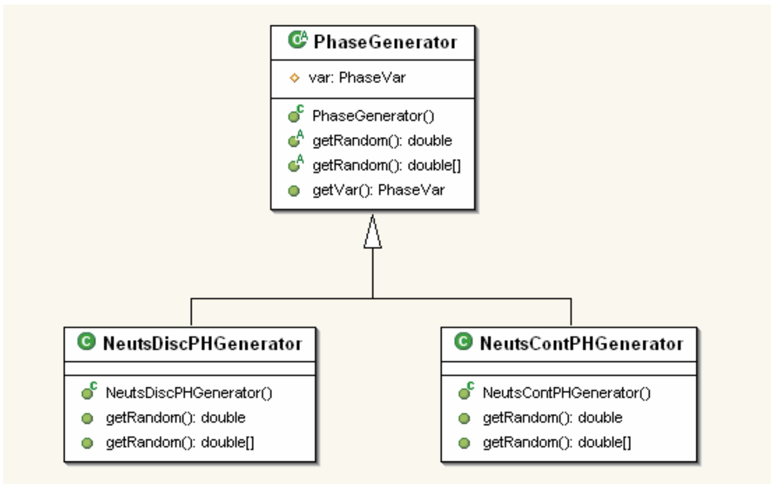

Figure 7: Simple jPhaseGenerator Package Class Diagram

3.1 `PhaseGenerator` Interface

This abstract class defines the basic methods that a Phase-Type random variate generator should have. The class includes an attribute, that belongs to the `PhaseVar` class, and is the distribution from which, the random variates will be generated. This distribution can only be specified in the constructor method, because the variable must be persistent in time for a particular PhaseGenerator object. This means that if the user wants to generate variates from another distribution, he or she must create a new PhaseGenerator.

In the constructor method, the variable is assigned and the `initialization()` method is called. It is expected that the user employs this method in order to effectively initialize the algorithm, and then a random variate can be generated after the construction of the PhaseGenerator. Another method defined by the Abstract class is `getVar()`, which always returns the Phase-Type variable that remain under the PhaseGenerator and is already implemented.

The last two methods that a PhaseGenerator must implement are `getRandom()` and `getRandom(`k`)`. The first one must return a variate that follows the distribution specified at the construction, and the second must return k independent variates with the same characteristic.

3.2 Concrete Classes

Up to day, two concrete classes extend the previously explained `PhaseGenerator` abstract class. These are `NeutsContPHGenerator` and `NeutsDiscPHGenerator`, which implement the method proposed by Neuts and Pagano [12]. The first one implements the continuous case and the second the discrete one. The continuous algorithm has a first step, in which the continuous chain is transformed is a discrete one, using the well-known embedded chain. Thereafter, the main algorithm (for discrete distributions) can be used for both cases.

The algorithm simulates the whole process in the chain: it first choose an initial state from the distribution given by the initial probability vector; then it selects a next state to visit using the discrete distribution associated with the present state, given by the associated row in the transition matrix; the selection of the next state is repeated until the chosen state is the absorbing one. In the discrete case, the value of the random variate is the number of steps (selections) made until absorption. For the continuous case, the number of visits to each state is stored and an Erlang variate is generated for each state with non-zero number of visits. The parameters of the Erlang distributions are the associated rate of the state and the number of visits carried out. For example, if the state i was visited n_i times and has an associated rate of λ_i, an Erlang(λ_i,n_i) random variate must be generated. The sum of these variates over all the states is the value of the Phase-Type random variate.

Two important issues of this algorithm must be emphasized. The first one is the several use of discrete distributions to generate the variates, which can be done efficiently through the alias method [35]. The second issue is that for the continuous case, in addition to the discrete variates, only Erlang variates must be generated. In the case of many visits to the same state, these variates can also be efficiently generated by multiplying a gamma variate with parameters $(n_i, 1)$ times λ_i, that will be an Erlang variate with the required parameters [12].

The algorithms implemented in these classes are supported by the utilities class `GeneratorUtils`, that have several procedures useful for the generators. Particularly, it has a general implementation of the alias method used to generate variates from discrete distributions [35]. It also has an implementation of the polynomial-time algorithm proposed by Gonzalez et. al. [36] to perform a Kolmogorov-Smirnov test, that can be useful to test the goodness-of-fit of the generated numbers in relation to the theoretic Phase-Type distribution.

Chapter IV

jPhaseFit: the Fitting Module

This package contains the structure that defines the behavior of the classes that implement algorithms to fit the parameters of a Phase-Type distribution. As shown in Figure 8, the interface `PhaseFitter` is in the top of the package and defines the basic method that any PhaseFitter should have: `fit()`. This method has no parameters and must return a Phase-Type variable as the result of the fitting process.

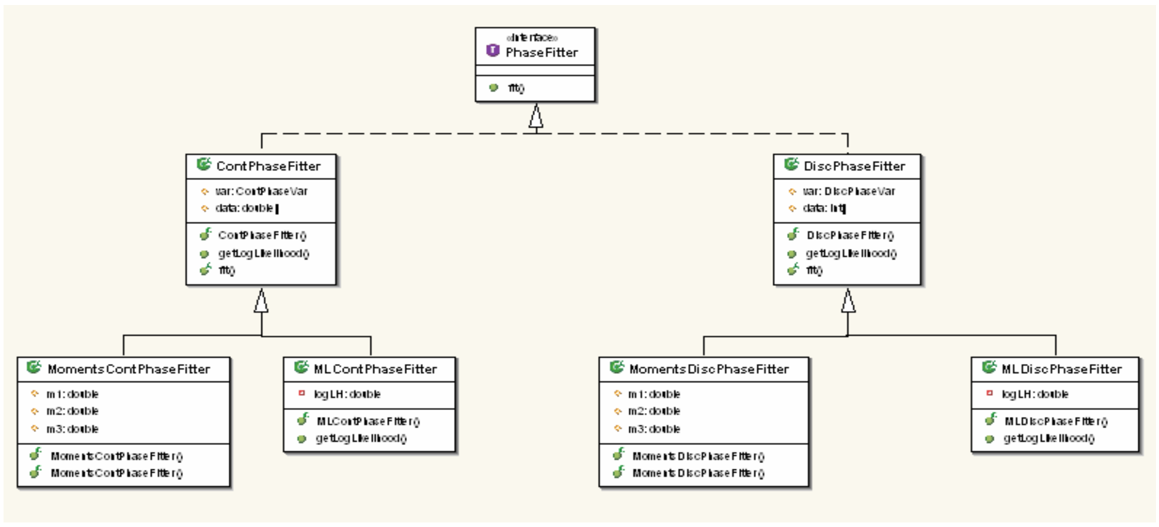

Figure 8: Simple jPhaseFit Package Class Diagram

4.1 Abstract Classes

In the next level, there are two abstract classes that implement the `PhaseFitter` interface: `ContPhaseFitter` and `DiscPhaseFitter`, for the continuous and discrete case, respectively. These classes have two additional issues: a constructor method from a data set in array format; and a method to compute the log-likelihood of the fitted distribution in relation to the data set (`getLoglikelihood()`). This is done because the log-likelihood is a usual way to compare the performance of fitting algorithms. In addition, this classes specify the continuous or discrete nature of the variable to be fitted in two different ways: the first one is the inclusion of the `var` attribute, where the fitted variable must be stored (a `ContPhaseVar` object for the continuous case or a `DiscPhaseVar` for the discrete case); the other way is the use of a data array as attribute, that in the continuous case is a double array, and in the discrete case is an integer array.

In the next level of abstract classes, a further division is done between classes that implement Maximum Likelihood (ML) algorithms and those related to Moment Matching techniques. This is done for both continuous and discrete cases. For the ML classes (`MLContPhaseFitter` and `MLDiscPhaseFitter`), there is a new attribute called `logLH`, that stores the log-likelihood value in order to take advance of the usual computation of the log-likelihood in the fitting process. For the Moment-Matching related classes (`MomentsContPhaseFitter` and `MomentsDiscPhaseFitter`), a new set of attributes is defined: `m1`, `m2`, and `m3`. These are the moments to me matched and are specified with a new constructor that receives only the three moments to be matched. An alternative way is the use of the redefined constructor that receives the data trace and calculates its moments. It must be said that there is not alternative to change the data, moments of log-likelihood attributes from outside the class, implying a safe fitting process.

4.2 Concrete Classes: Maximum Likelihood Algorithms

The set of classes that implement maximum likelihood algorithms are almost all for Continuous Phase-Type distributions, because the most of the efforts have been done in that direction. For each one of the following algorithms, there is an associated class the executes the procedures to fit the parameters of a distribution.

4.2.1 General Phase-Type Distribution EM Algorithm [1]

The EM algorithm proposed by Asmussen, Nerman and Olsson [1] is the only one algorithm that deals with the fitting of the whole set of continuous Phase-Type distribution, without reducing the distributions to a restricted subset. The EM algorithm was first introduced by Dempster et. al. [31] to deal with the problem of incomplete data (a good source to review it may be [37]). The idea behind this algorithm is that a complete sample from Phase-Type realizations should include the selected initial state, the whole path of states followed until absorption, and the time spent in each of these states. With this complete sample, it's easy to estimate the parameters of the distribution.

Nevertheless the sample obtained from Phase-Type realizations are only the time until absorption. In this way, the problem can be seen as the estimation of the parameters from an incomplete sample, which makes natural the use of the EM algorithm. The algorithm begins from an initial guess of the parameters and the iterations include the computation of the likelihood (E-step) and the its maximization to obtain a new set of parameters (M-step). In this case, the heavy work must be done in the E-step, where a set of $n(n+2)$ linear differential equations must be solved for a distribution of n phases. It must be noted that this algorithm does not select the number of phases, and it must be entered as an initial parameter. Even though this algorithm has been already evaluated in [18], it is included in the benchmark evaluation because it is the only one that fits the parameters of the whole Phase-Type class.

The concrete class that implements the algorithm is `EMPhaseFit`, that extends the abstract class `MLContPhaseFitter`. In this implementation, the method `fit()` doesn't need the specification of any parameter but it tries with distributions from 1 to 10 to find the one that shows the greatest log-likelihood. To do this, it calls the method `fit(`n`)`, that executes the proper algorithm to fit the parameters of a general Phase-Type distribution with n phases. In every iteration, this method calls the `eStep()` and `mStep` methods that executes the procedures for each of those steps in the EM algorithm. Particularly, the E-step uses an order-four Runge-Kutta procedure to solve the set of differential equations, which solution is expressed in the inner class `solution`.

The user could also make use of another constructor for this class in order to specify some features for the algorithm. With the method `EMPhaseFit(precision, iterations, evalPoints)`, three important features can be set: the *precision* for stopping the algorithm when the parameters show little change; the maximum number of *iterations* that the algorithm can execute; and the *evalPoints* parameter determines the factor to multiply the data trace size in order to obtain the number of evaluation points for the Runge-Kutta method.

4.2.2 HyperExponential Distribution EM Algorithm [2]

The hyper-exponential distribution is a very special case of Phase-Type distributions, since the initial probability vector defines the probability of choosing the exponential phase to visit, and the generator matrix have diagonal representation with the rates of the i-th phase in the position (i,i). Thus the number of parameter to fit a n-phase hyper-exponential distribution are $2n$. The algorithm proposed by Khayari et. al. [2] is also an EM algorithm like the explained above. It begins with an initial guess of the parameters, that can be random or related to the properties of the trace (e.g. the expected value). The authors propose an easy way to select the initial parameters. Then a function to evaluate the quality of the parameters is calculated in the E-step through the probability density function of the data trace given the parameters.

In the M-step, the new set of parameters is computed using estimators for the rates and the probabilities but not for the number of phases, that is taken as a given parameter.

The implementation of the algorithm was done in the `EMHyperExpoFit` class, where a method `fit(`n`)` is implemented in order to fit a distribution with n phases. As the method `fit()` must also be implemented in order to follow the parameters of the `PhaseFitter` interface, it executes several trials of configurations from one to ten phases, and selects the distribution with greatest likelihood. With the use of another constructor, the user can also specify the maximum number of iterations that the algorithm can execute and the precision level required to determine when the change in the estimated parameters is too little and the algorithm should stop.

4.2.3 HyperErlang Distribution EM Algorithm [3]

In 2005, Thümmler et. al. presented a method that fits the parameters of a hyper-Erlang distribution [3], which is a very interesting subset of the Phase-Type distributions since they are also dense in $[0, \infty)$. In some results provided by Thümmler et. al.[3], the EM algorithm developed for this special class has a better behavior in terms of likelihood than the one designed for the complete Phase family [1]. The algorithm needs receives as a parameter the number of Erlang branches in the distribution as well as the total number of exponential phases in the distribution. With this information, the algorithm determines all the possible configurations of the Erlang branches and executes a version of the EM algorithm for each case. Finally, the configuration with the greatest likelihood is selected as the result of the algorithm.

As can be seen, this algorithm needs more information than the previous ones, and so the routine `fit()` makes a different work than just try distributions with one to ten phases. In the `EMHyperErlangFit` class, the method `fit()` guides the search of the configuration by means of the coefficient of variation of the data trace. When the coefficient is lower than one, then it doesn't allow more than one branch since it has been shown that the Phase-Type variable with the least coefficient of variation is the n-Erlang($\frac{1}{n}$) [38]. When the coefficient of variation is grater than one, it enforces the creation of multiple branches as well as phases in each of them. The method `fit(`n, m`)` executes the effective procedure proposed in the paper for a distributions with n phases and m branches. When all the possible configurations has been determined, this method calls `fit(`n, m, r`)` where the number of phases at the i-th branch is r_i. For this method, the parameters related to precision and number of iterations are important and the user can fix them with use of a special constructor.

4.3 Concrete Classes: Moment Matching Algorithms

The distribution moments usually play an important role in the performance analysis of real systems [5]. This has been an important motivation for the improvement of moment matching techniques, and the attention given by different research communities (Operations Research, Computer Science and Telecommunication Networks, among others). Some of the most recent advances have been implemented in the **jPhaseFit** module, as will be explained in this section.

4.3.1 Acyclic Continuous order-2 Distributions [4]

In 2002 Telek and Heindl [4] proposed an algorithm to fit the parameters of an acyclic Phase-Type distributions of second order (two phases). Acyclic distributions have been extensively studied since they have some important properties, as a canonic form developed by Cumani [30] and a upper triangular transition or generator matrix. In that paper, they establish bounds on the set of first three moments representable by acyclic distributions of second order, for the discrete and continuous cases. Over the characterization of these bounds, they build the algorithm that matches three moments with the three parameters of this distribution: the rates of each phase and the absorption probability after the first phase (the initial probability is all in the first phase as in the Coxian distribution).

This algorithm is implemented by the class `MomentsACPH2Fit`, that extends the abstract class `MomentsContPhaseFitter`. As it is constructed with the three moments to be matched (given explicitly or computed from a data trace), the algorithm begins with the computation of the bounds, in order to determine if the moment set is representable. If not, the moments are corrected to the nearest point in the representable region with a warning message about the correction for the user. When the moment set is representable, the parameters of the distribution are calculated according to the equations shown by the authors. Finally, the distribution is constructed with the parameters and returned to the user.

In the same paper, the authors present an analogous algorithm for the discrete case. It works in a similar fashion and is completely implemented by the class `MomentsADPH2Fit`, that extends the abstract class `MomentsDiscPhaseFitter`.

4.3.2 ErlangCoxian Distributions[5]

The next step in moment-matching techniques was given by Osogami and Harchol in a series of papers [26] [27] [5]. This extension consists on the characterization of the bounds imposed over the first three moments representable by a Phase-Type distribution withn n phases. They also introduce Erlang-Coxian distributions, a name due to the fact that they can be represented as the convolution of an Erlang and a Coxian distribution of second order. They present an algorithm to fit the parameters of a Erlang-Coxian distribution with or without mass at zero, an important issue in constructing matrix-geometric models from phase type distributions. An important issue is that the algorithm itself determines the number of phases needed to represent the set of moments, making easier the use of the algorithm since the user doesn't need to try with different configurations. The resulting distributions are not large in the number of phases but are not strictly minimal.

The implementation of the algorithms is given by two classes: `MomentsECCompleteFit` and `MomentsECPositiveFit`. The first one is built over the "complete solution" proposed by the authors, where the moment set is representable by the convolution of Erlang and Coxian distribution but the resulting distribution can have a positive mass on zero. To avoid this, the second class implements the "positive solution", where all the resulting distributions have no mass at zero, but the Erlang-Coxian distribution must be extended through a convolution or a convex mixture with a exponential distribution in order to obtain the strictly positiveness. Whenever the complete solution returns a positive distribution, this will be used by the `MomentsECPositiveFit`, an issue that forces this this class to depend on the `MomentsECCompleteFit` class.

4.3.3 Acyclic Continuous Distributions [6]

The last effort done in this area was made in 2005 by Bobbio, Horvath and Telek [6], who present an algorithm to match a set of first three moments with acyclic Phase-Type distributions(APH). They show the possible sets that can be represented by an acyclic distribution of order n. Then they show how to match the first three moments in a minimal way, i.e. using the minimal number of phases needed to do it. It is done by determining the region representable by an APH of n phases but not with an APH with $n-1$. This region is then partitioned in five areas that represent different distribution configurations, such as the Erlang-Exp structure that represents and $n-1$ Erlang distribution with an additional exponential phase after it.

The algorithm proposed by the authors for the positive case is implemented in the `MomentsACPHFit` class. There the algorithm begins with the first three non-central moments and computes the first two normalized moments. With this information, the required number of phases is computed and the moment set is evaluated in order to find in which region it falls. When it is determined, the parameters are fitted according to the equations presented by the authors.

Chapter V

Fitting Algorithms Benchmarking

The algorithms presented in the preceding section show different features that may make them more suitable for certain applications. In this section, these algorithm are subject to a comparative analysis in order to get offer more information about their performance. The following sections are devoted to the illustration of the methodology used for the analysis, the data traces used to test the algorithms and the obtained results. With this results, it is expected that the jPhase user has more information about which algorithm select in the development of a specific application.

5.1 Methodology

The methodology here exposed has been widely used in different works as a way to evaluate the performance of a particular algorithm [29] [3]. It consists of the selection of a wide range of probability distributions that present different behaviors, generate random numbers with such distributions and used those traces as input for the algorithms. In addition, some real traces are brought from well-known sources in order to evaluate the algorithms against data characteristics closer to those that an analyst can face in a real application.

Once any algorithm has been executed, a Phase-Type distribution is obtained and the Measures of Performance (MOPs) are computed. These measures were first presented in [29], as the result of the discussion held by many researchers at the workshop "Fitting Phase-Type Distributions" in 1991. The measures are:

- Absolute error on First Moment:
$$e_1 = \frac{|e_1 - \hat{e}_1|}{e_1}.$$
- Absolute error on Second Moment:
$$e_2 = \frac{|e_2 - \hat{e}_2|}{e_2}.$$

- Absolute error on Third Moment:

$$e_3 = \frac{|e_3 - \hat{e}_3|}{e_3}.$$

- Minus Cross Entropy (log-likelihood):

$$\int_0^\infty log\hat{f}(t)dF(t).$$

- Absolute difference between cumulative probability functions:

$$\int_0^\infty |F(t) - \hat{F}(t)|dt.$$

Here e_i is the i-th non-centered moment of the original distribution (or data trace) and $\hat{e}_i$ is the i-th non-centered moment of the approximated Phase-Type distribution. The Minus Cross Entropy is reduced to the log-likelihood (logLH) when the original distribution is a data trace. Finally, the last MOP is different from that presented in [29], since they use the probability density ($f(t)$) instead of the cumulative distribution function ($F(t)$). This is because the cumulative distribution is limited to take values in the interval $[0, 1]$ and thus, it is a good way to compare the performance of a particular algorithm against different traces.

5.2 Selected Traces

The traces selected are also based in those used in [29] as benchmark distributions. These distributions provide a wide range of behaviors, including distributions usually found in applications, and some challenging characteristics as long tails, multi-modality, low variability and sharp jumps in the density function [18]. Nevertheless, two more distributions were included from those proposed in [17] in order to analyze the behavior of the algorithms when the data exhibits heavy tails. These distributions conform the set of "Generated Traces", since for each one of them a sample of 1000 random numbers was generated. The specific characteristics of the distributions are summarized in Table 4.

In addition to the generated distributions, two real traces were included in the analysis. The first one will be known as the NASA trace, and is a sample of 65000 data points taken from the well known NASA-HTTP trace that contains HTTP requests to the NASA Kennedy Space Center WWW server in Florida [39]. As was stated by [40], this data trace shows heavy tail behavior, which is a relevant feature to analyze the algorithm preformance. The second trace constains data about service times in a Call center. This data set was collected by Professor Mandelbaum and can be accessed from [41]. The relevance of the trace comes from the bimodality shown by the process, that is related to the different costumer behaviors.

Family	Density	Label	Parameters	Observations
Weibull	$f(t) = \frac{\beta}{\eta}\left(\frac{t}{\eta}\right)^{\beta-1} e^{-\left(\frac{t}{\eta}\right)^{\beta}}, \quad t > 0$	W1	$\eta = 1 \quad \beta = 1.5$	Decreasing hazard rate
		W2	$\eta = 1 \quad \beta = 0.5$	Long tail
Lognormal	$f(t) = \frac{1}{t\sqrt{2\pi\sigma^2}} e^{-\frac{(lnt-\mu)}{2\sigma^2}}, \quad t > 0$	L1	$\mu = 1 \quad \sigma = 1.8$	Long Tail
		L2	$\mu = 1 \quad \sigma = 1.2$	
		L3	$\mu = 1 \quad \sigma = 0.2$	Low Variability
Uniform	$f(t) = \frac{1}{b-a}, \quad a \le t \le b$	U1	$a = 0 \quad b = 1$	Low variability
		U2	$a = 1 \quad b = 2$	Sharp jumps in pdf
Shifted Exponential	$f(t) = \frac{1}{2}e^{-t} + \frac{1}{2}e^{-(t-1)}I(t \ge 1), t > 0$	SE		Sharp jumps in pdf
Matrix Exponential	$f(t) = \left(1 + \frac{1}{(2\pi)^2}\right)(1 - cos(2\pi t))e^{-t}$	ME		Multi-modality
Pareto I	$f(t) = \begin{cases} \alpha\beta^{-1}e^{-\frac{\alpha}{\beta}t}, t \le \beta \\ \alpha\alpha^{\beta}e^{-\alpha}t^{-(\alpha+1)}, t > \beta \end{cases}$	PI	$\alpha = 1.5 \quad \beta = 4$	Heavy Tail
Pareto II	$f(t) = \frac{\beta^{\alpha}e^{-\frac{\beta}{t}}}{\Gamma(\alpha)}t^{-(\alpha+1)}, t > 0$	PII	$\alpha = 1.2 \quad \beta = 2$	Heavy Tail

Table 4: Generated Traces

5.3 Results

Given the numerous results related to the comparative analysis, the tables with the whole set of numerical results has been left for the Appendix. In this section, the analysis is centered in the Figures presented and other relevant results included in the tables.

5.3.1 Results on Matching Moment Algorithms

In Figures 9 and 10, the results for the Moment Matching algorithms in term of log-likelihood are presented. As stated above, these algorithms match perfectly the first three moments of any distribution. Therefore the errors in first, second and third moments are all equal to zero.

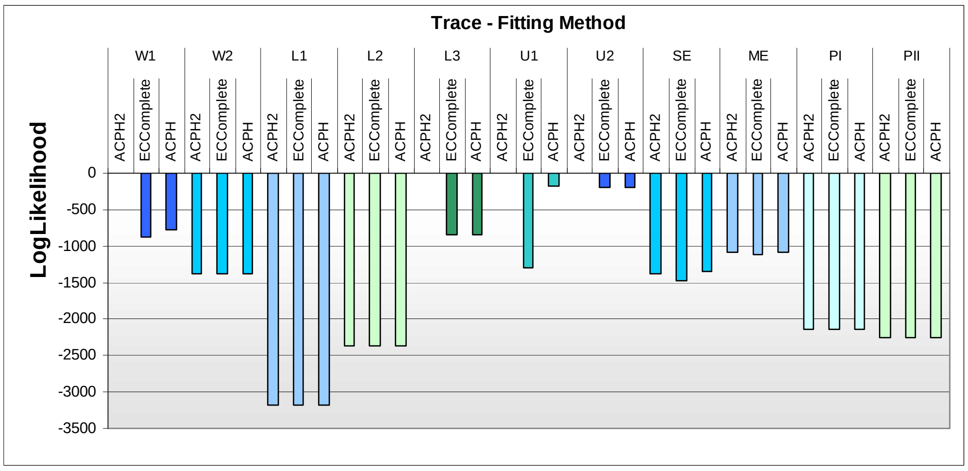

Figure 9: Loglikelihood of Generated Traces for Moment Methods

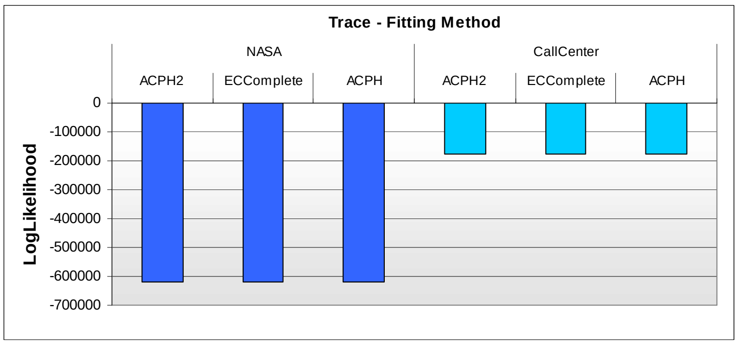

Figure 10: Loglikelihood of Real Traces for Moment Methods

In these graphics, it can be seen that the three algorithms have a very close performance in almost all the traces. Nevertheless, the *ACPH2* method fails when the trace shows low variability, as in `W1`, `L3`, `U1` and `U2`. This is because the least coefficient of variation that can be reached with an order-2 Continuous Phase-Type Distribution is 0.5, as stated in [38].

In all other traces, the differences between *ECComplete* and *ACPH* methods arise only in W1 and U1, where the second one shows greater log-likelihood. As can be seen in Figure 11, this is related to the inclusion of one extra phase in the latter method.

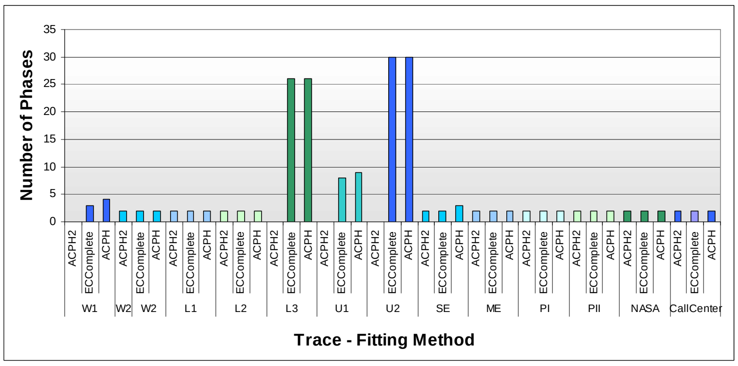

Figure 11: Number of phases for Moment Methods

In relation to the area difference between the cumulative distribution functions (c.d.f.)of the original and the matched distributions, the results are very similar to those found with the log-likelihood measure. The *ACPH* method shows a closer approximation to the original distribution than the *ECComplete* method in the W1 and U1 traces. In other distributions, there are nor great differences between both methods, but some distributions shown to be harder for fitting. That is the case of L1, SE and PII traces, which had errors above 10 percent. The low variability distributions, such as L3, U1 and U2, needed more phases to be represented but the c.d.f. area differences are all around 5%.

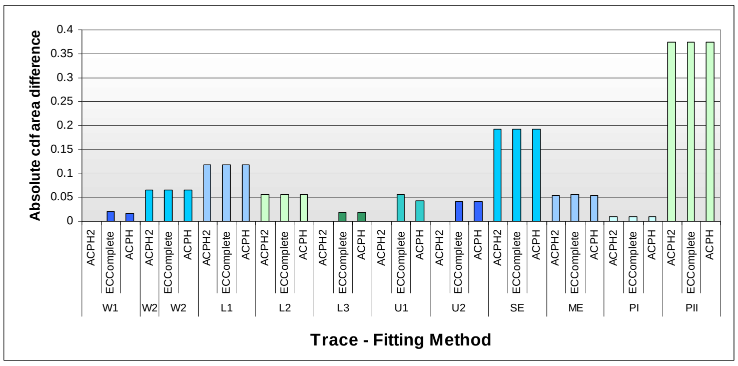

Figure 12: Area difference in CDF for Moment Methods

From the traces examined, the *ACPH2* method seems to be a good alternative when the data does not show low variability. Although it has restrictions on the third representable moment, it assures distributions with only two phases, which can be of great importance to avoid dimensionality problems. Nevertheless, for low variable traces the *ECComplete* and *ACPH* methods offer a solution where the other one fails. Both alternatives will find a distribution with the asked set of moments, but incurring at the cost of a bigger state space. Long and Heavy Tails can be difficult to be fitted but that's not necessarily true, since some of these distributions were adequately approximated.

5.3.2 Results on Maximum Likelihood Algorithms

As all of the algorithms examined in this document need the specification of the number of phases, the results depend on the selection of this parameter. For the analysis here exposed, the distributions are assumed to be composed by 4 phases. Nevertheless, the analysis was done with 2, 4 and 8 phases, results that can be found in the Appendix.

In Figures 13 and 14, the results of the *EMHyperExpo*, *EMPhase* and *EMHyperErlang* related to the log-likelihood are shown. In general, the obtained log-likelihood with the three algorithms is very close, but the two latter methods show better values than the former. That is easily seen in `W1`, `L2`, `U1`, `SE` and `ME` traces. A strange behavior is presented in traces `L3` and `U2`, where the *EMPhase* method shows a worse performance. This outcome can be due to the complexity of the algorithm and maybe a better result could be reached with more than 200 iterations. It also possible that the algorithm fall in a local maximum of the log-likelihood function.

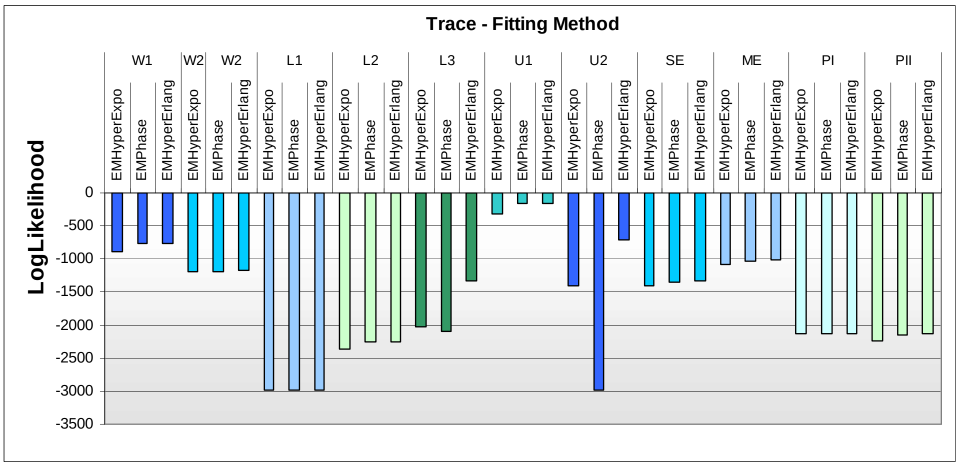

Figure 13: Loglikelihood of Generated Traces for Maximum Likelihood Methods over 4-phases distributions

The better results obtained by the *EMPhase* and *EMHyperErlang* in some difficult traces,

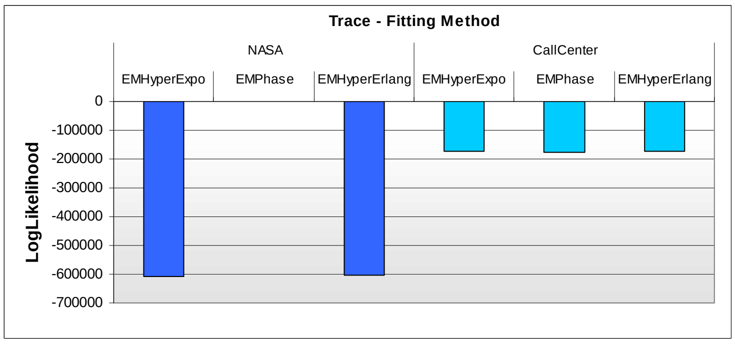

Figure 14: Loglikelihood of Real Traces for Maximum Likelihood Methods over 4-phases distributions

such as `U1`, `SE` or `ME`, is a clear signal of the greater flexibility offered by the more general distributions (Phase-Type and Hyper-Erlang) over the smaller set of Hyper-exponential distributions. This is result is confirmed in Figures 15 and 16, where the errors over second and third moments are shown. In general, the *EMHyperErlang* method reaches the best results in almost all the traces.

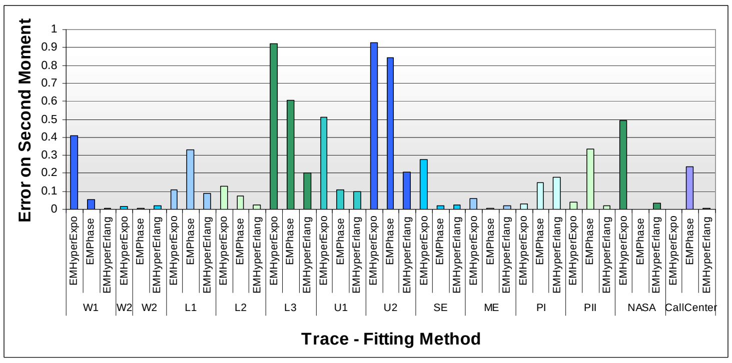

Figure 15: Absolute Error on Second Moment for Maximum Likelihood Methods over 4-phases distributions

An interesting result is given by the *EMHyperExpo* method, that shows a very good approximation of the second and third moments in the heavy tailed distributions `PI` and `PII`. This feature may be related to the flexibility of the Hyper-exponential distributions to fit long and heavy tails, as noted by [16]. In general, the approximation to these moments attained by the three algorithms is very close for the Phase-behaved distributions, as `W2` or `L2`. In other cases, the errors can be large and the results does not show a pattern about an algorithm than perform better that the others in all cases.

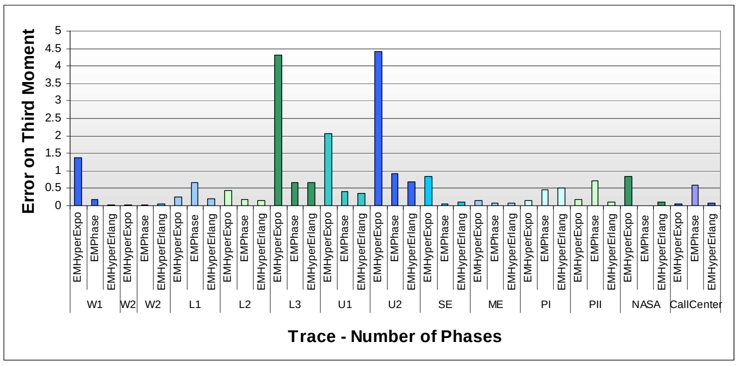

Figure 16: Absolute Error on Third Moment for Maximum Likelihood Methods over 4-phases distributions

5.3.3 Further Results on Maximum Likelihood Algorithm

In this section, a particular emphasis on the Maximum Likelihood Algorithms is made. In order to give more insight on their behavior, each one of the algorithms is analyzed in relation to the log-likelihood obtained by the matched distribution as a function of the assumed number of phases.

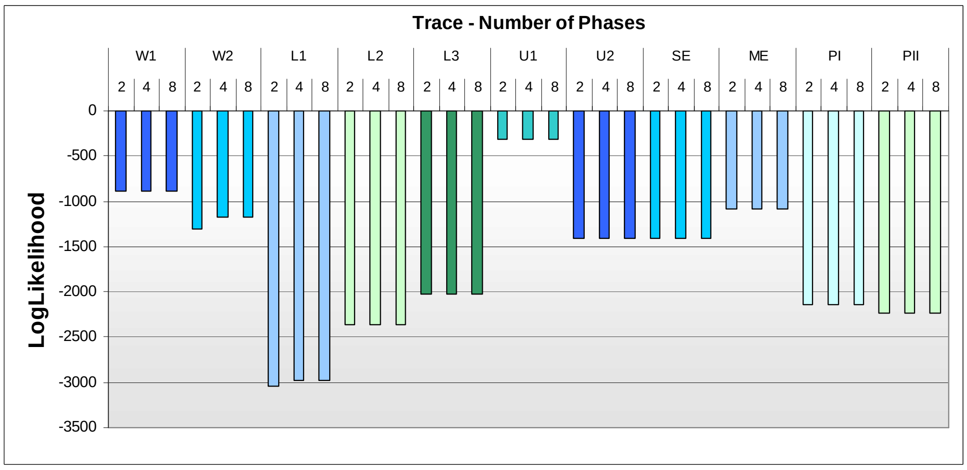

Figure 17: Loglikelihood of Generated Traces for the EMHyperExpo Method

The results for *EMHyperExpo* method is shown in Figures 17 and 18. It can be easily seen that the performance of the algorithm is enhanced with the addition of phases for traces W2, L1, PI, PII. In all this cases, the probability density function if monotonically decreasing, which is an important feature (and restriction) of Hyper-exponential distributions. In this way, it can be said that the method can perform better with the addition of phases if the underlying trace exhibits a decreasing density, but not in other cases.

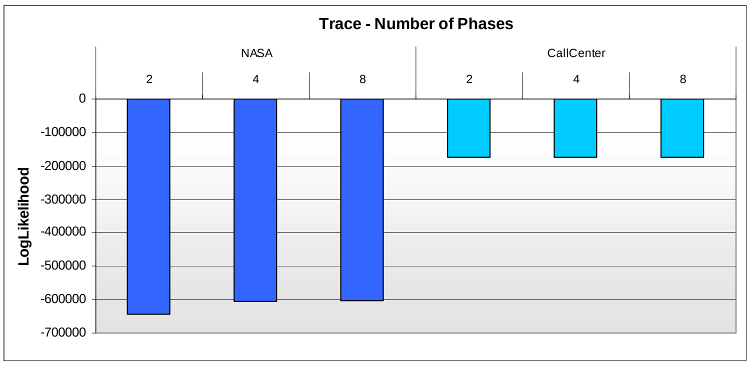

Figure 18: Loglikelihood of Real Traces for the EMHyperExpo Method

In relation to the real traces, the algorithm perform better for the NASA server data, where the addition of phases consistently increase the reached log-likelihood. Although the distribution with four phases shows a much better log-likelihood that the 2-phases one, the improvement is not so large with the addition of four more phases.

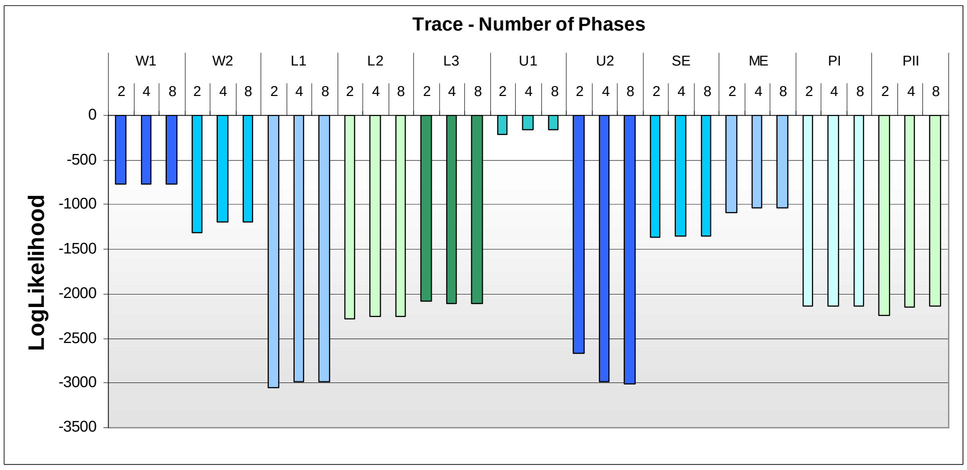

Figure 19: Loglikelihood of Generated Traces for the EMPhase Method

For the *EMPhase* method, the results are shown in Figures 19 and 20. It can be seen, probably better in Tables in the Appendix, that for traces `W2`, `L1`, `L2`, `L3`, `U1`, `U2`, `SE` and `ME`, the algorithm enhance its performance from 2 to 4 phases, but the likelihood does not make an improvement when the distributions is assumed to have 8 phases. The only one traces where this result is obtained (a consistent increase in the log-likelihood with the addition of phases) are `W1` and `SE`.

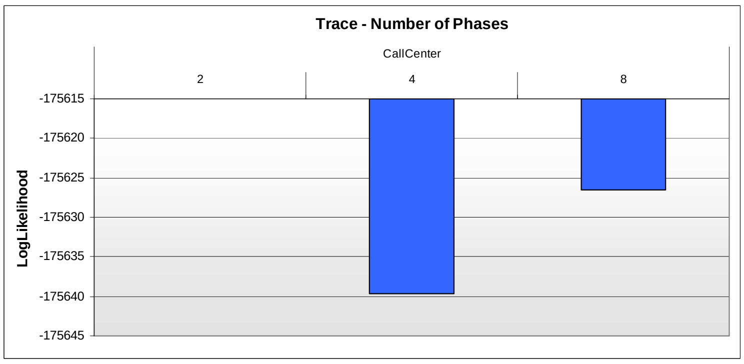

Figure 20: Loglikelihood of Real Traces for the EMPhase Method

Again, some strange results are obtained from the use of `L3` and `U2` traces, where the log-likelihood actually decreases with the addition of phases. This can be due to the restricted number of iterations (200) established for all the algorithms. These two distributions present a major challenge for Phase-Type distributions, since the first one has a big peak near to zero, and the second present two sharp jumps in its density.

In relation to the real traces, a regrettable result is that the size of the NASA trace (65000 data points) made impossible to run the algorithm in the personal computer used for the tests, since its memory was not enough. For the Call Center trace, the results are shown in Figure 20, where it is evident that the addition of phases generate a better log-likelihood. It can be a natural result from the better adaptation of the distribution with more phases to the bi-modality of the trace.

In the case of the *EMHyperErlang* method, the results are shown in Figures 21 and 22. The most important result is that the method improve the reached log-likelihood with addition of phase in almost all the traces. The only one where this is not true is the `L1` data set, where the fit is almost the same in all the Measures of Performance for the 4 and 8 phases cases. For the `W2` trace, the log-likelihood of the matched distribution could not be obtained because of numerical problems.

For the real traces, the results are very similar, since the algorithm behaves better with the addition of phases in both cases. This may be the result of the flexibility of this class of distributions, which is an important issue for getting closer to the original distribution when it present difficult characteristics.

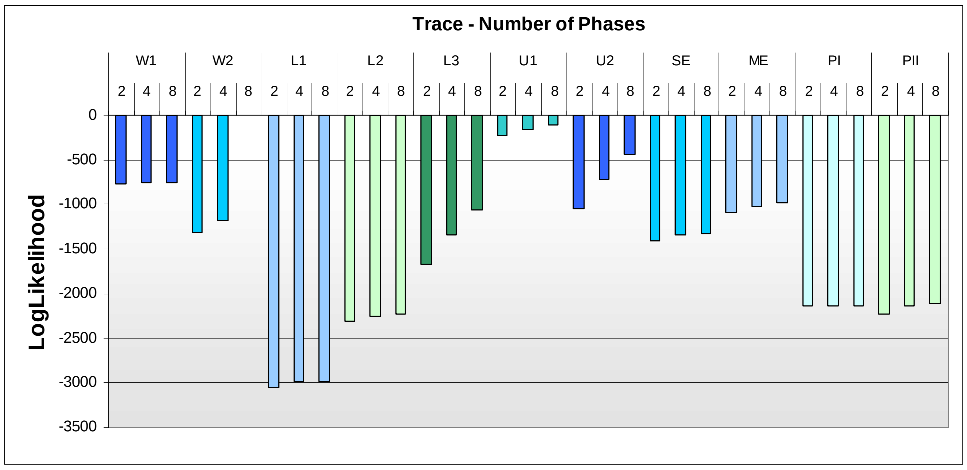

Figure 21: Loglikelihood of Generated Traces for the EMHyperErlang Method

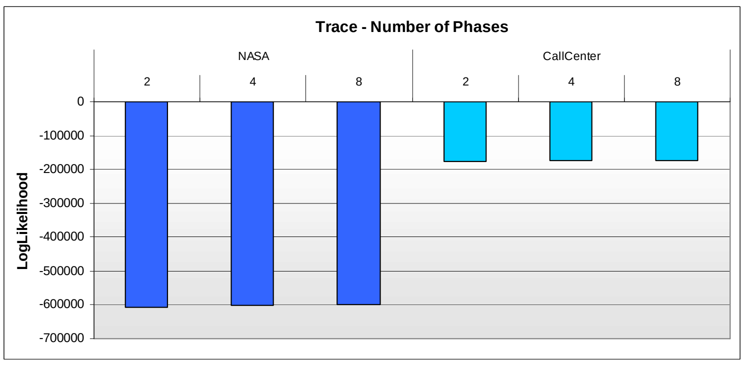

Figure 22: Loglikelihood of Real Traces for the EMHyperErlang Method

Chapter VI

Conclusions

Phase-Type distribution has shown to be a powerful tool in computational probability since they can be used as input of Markov chains, which allows the use of efficient algorithms to compute measures of performance of real systems. In this work a computational framework has been designed and developed in order to allow the computational representation and manipulation of these distributions. The computational objects allows the user to concentrate in the modeling issues and not in the computation of distributions, moments or closure properties. In this way, the developed tool makes more accessible the of Phase-Type distribution for researches interested in stochastic modeling and performance evaluation of real systems.

The extensibility of the framework helps the user to develop new classes that can have a different representation (special sparse structures), but still exploiting the implemented methods in abstract classes. Even more, in the development of such extended classes the user can just implement some simple methods for the specific representation, or can develop procedures for the some or all the methods related to the distribution. In this way, the structure is not restricted to the developed methods, e.g. the researcher could use a different solver to compute the density function of a particular class of distributions.

The framework also includes a module for Phase-Type variates generation, which can be used to model large systems using simulation models with Phase-Type distributions. The tool has itself some procedures to do that, but the user could also develop a new algorithm and implement it with the help of the utilities methods and the unified framework.

Finally, the fitting module offers a set of recently developed algorithms to fit the parameters of a Phase-Type distribution from a data trace. It is possible to use general setting for the algorithms, without specifying any parameter. But the user can also determine specific characteristics, as the number of phases in the distribution, or the precision for convergence criterion. There is also a framework that can help to design the implementation of new algorithms, since the user have all the distribution classes as well as other algorithms to support his or her development.

In relation to the benchmarking, it must be highlighted that the *EMHyperExpo* method is the fastest between the Maximum Likelihood Algorithms and *EMPhase* is the most complex one. The Moment Matching techniques are even faster since they are all closed functions, which imply a smaller number of computations than iterative algorithms. In order to choose a particular algorithm to fit a data set, it is important to account the available computing resources as well as time for building the distribution from data traces.

Between the Moment Matching methods, the *ACPH2* method seems to be a good alternative when the data does not show low variability. Nevertheless, for low variable traces the *ECComplete* and *ACPH* methods offer a solution where the first one fails. Both alternatives will find a distribution with the asked set of moments, but incurring at the cost of a bigger state space. For low variable distributions, the c.d.f. area difference in these methods is smaller than in Maximum Likelihood approaches, which gives them an advantage for dealing with this kind of data.

For Maximum Likelihood algorithms, the traces `L3` and `U2` were the major challenge and generated some strange results for the *EMPhase* method. For all other traces, the behavior of the methods was always better than Moment Matching techniques in relation to c.d.f. area difference. In particular, the *EMHyperExpo* method improves its performance with the addition of phases for those traces with decreasing density. The *EMPhase* method enhances its behavior for most of the traces when the change was between two and four phases, but not with four more phases. The *EMHyperErlang* method showed the most consistent improvements, since it had a better behavior with addition of phases for almost all the traces.

Long and Heavy Tails can be difficult to be fitted but that's not necessarily true, since some of these distributions were adequately approximated. In this case, particularly for `PII` trace, the *EMHyperErlang* exhibits the best results in c.d.f. area difference and log-likelihood. It also shows small errors on the second moment, and its only fail is on the third moment. The relevance of this result will be related to the dependence of the process to the moments of the distribution. In case of having a strongly dependency on the moments, it should be better to use the Moment Matching techniques, since the other methods does not offer a guarantee over this matching.

Appendix A

Results of Fitting Algorithms

A.1 Results for Moment Matching Algorithms

In this section, the results for the Moment Matching Algorithms are presented. The Measures of Performance for each algorithm over each trace can be found in in Tables 5, 6, 7, 8, 9 and 10 .

Table 5: Results of Moment Matching Algorithms for Weibull Distributions

Trace	Measure	Method		
		ACPH2	ECComplete	ACPH
W1	logLH	-	-877.38	-772.34
	Area Difference	-	2.02%	1.68%
	Phases	-	3	4
W2	logLH	-1369.90	-1369.90	-1369.90
	Area Difference	6.55%	6.55%	6.55%
	Phases	2	2	2

Table 6: Results of Moment Matching Algorithms for Lognormal Distributions

Trace	Measure	Method		
		ACPH2	ECComplete	ACPH
L1	logLH	-3183.09	-3183.09	-3183.09
	Area Difference	11.78%	11.78%	11.78%
	Phases	2	2	2
L2	logLH	-2361.99	-2361.99	-2361.99
	Area Difference	5.61%	5.61%	5.61%
	Phases	2	2	2
L3	logLH	-	-837.37	-837.20
	Area Difference	-	1.85%	1.79%
	Phases	-	26	26

Table 7: Results of Moment Matching Algorithms for Uniform Distributions

Trace	Measure	Method		
		ACPH2	ECComplete	ACPH
U1	logLH	-	-1303.72	-180.06
	Area Difference	-	5.60%	4.31%
	Phases	-	8	9
U2	logLH	-	-188.99	-190.71
	Area Difference	-	4.04%	4.09%
	Phases	-	30	30

Table 8: Results of Moment Matching Algorithms for Expo* Distributions

Trace	Measure	Method		
		ACPH2	ECComplete	ACPH
SE	logLH	-1371.83	-1479.89	-1348.35
	Area Difference	19.19%	19.32%	19.29%
	Phases	2	2	3
ME	logLH	-1082.79	-1116.76	-1082.79
	Area Difference	5.44%	5.70%	5.44%
	Phases	2	2	2

Table 9: Results of Moment Matching Algorithms for Pareto Distributions

Trace	Measure	Method		
		ACPH2	ECComplete	ACPH
PI	logLH	-2140.68	-2140.68	-2140.68
	Area Difference	0.99%	0.99%	0.99%
	Phases	2	2	2
PII	logLH	-2256.35	-2256.35	-2256.35
	Area Difference	37.34%	37.34%	37.34%
	Phases	2	2	2

Table 10: Results of Moment Matching Algorithms for Real Data Traces

Trace	Measure	Method		
		ACPH2	ECComplete	ACPH
NASA	logLH	-618645	-618645	-618645
	Phases	2	2	2
CallCenter	logLH	-175029	-175029	-175029
	Phases	2	2	2

A.2 Results for Maximum Likelihood Algorithms

Similarly to the Moment Matching Algorithms, in this sections the results for the Maximum Likelihood algorithms are preseneted. Since all those algorithms need the specification of the number of phases as an input parameter, the results are divided in three pieces: assumming distributions with 2, 4 and 8 phases.

A.2.1 Results for 2-phases Distributions

In this section, the results for the Maximum Likelihood Algorithm assuming 2-phase distributions are presented. The Measures of Performance for each algorithm over each trace can be found in Tables 11, 12, 13, 14, 15 and 16.

Table 11: Results of Maximum Likelihood Algorithms with 2 phases for Weibull Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
W1	e1	0	0	0
	e2	0.4096	0.0643	0.0572
	e3	1.3821	0.2123	0.1911
	logLH	-892.8130	-767.2745	-769.1995
	Area Difference	7.42%	1.20%	1.16%
W2	e1	0	0	0
	e2	0.3413	0.3419	0.3447
	e3	0.6376	0.6383	0.6416
	logLH	-1311.6388	-1311.6382	-1310.6419
	Area Difference	3.83%	3.83%	3.88%

Table 12: Results of Maximum Likelihood Algorithms with 2 phases for Lognormal Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
L1	e1	0	0	0
	e2	0.5656	0.5667	0.5666
	e3	0.8796	0.8803	0.8802
	logLH	-3048.6947	-3048.7016	-3047.6954
	Area Difference	3.91%	3.88%	3.93%
L2	e1	0	0	0
	e2	0.0134	0.2092	0.2601
	e3	0.2646	0.5169	0.6323
	logLH	-2362.0681	-2275.7934	-2311.2925
	Area Difference	5.61%	2.34%	5.22%
L3	e1	0	0	0
	e2	0.9208	0.5545	0.4406
	e3	4.3213	0.5422	1.6606
	logLH	-2029.6261	-2087.3338	-1663.9566
	Area Difference	20.47%	34.04%	17.12%

Table 13: Results of Maximum Likelihood Algorithms with 2 phases for Uniform Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
U1	e1	0	0	0
	e2	0.5128	0.1825	0.1346
	e3	2.0561	0.6801	0.5280
	logLH	-316.9736	-206.4998	-229.1203
	Area Difference	9.36%	5.09%	5.03%
U2	e1	0	0	0
	e2	0.9286	0.8294	0.4464
	e3	4.4006	0.8864	1.7003
	logLH	-1408.8661	-2665.2694	-1041.8041
	Area Difference	21.04%	46.23%	17.30%

Table 14: Results of Maximum Likelihood Algorithms with 2 phases for Expo* Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
SE	e1	0	0	0
	e2	0.2759	0.0508	0.2747
	e3	0.8316	0.1343	0.8279
	logLH	-1409.2037	-1361.7107	-1408.2036
	Area Difference	19.06%	18.91%	19.04%
ME	e1	0	0	0
	e2	0.0581	0.0474	0.0570
	e3	0.1588	0.0224	0.1565
	logLH	-1085.9813	-1085.7114	-1084.9810
	Area Difference	5.67%	5.56%	5.65%

Table 15: Results of Maximum Likelihood Algorithms with 2 phases for Pareto Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
PI	e1	0	0	0
	e2	0.1263	0.1269	0.1275
	e3	0.3981	0.3981	0.4003
	logLH	-2137.7433	-2137.7459	-2136.7440
	Area Difference	0.49%	0.49%	0.46%
PII	e1	0	0	0
	e2	0.2273	0.2654	0.2284
	e3	0.5909	0.6320	0.5923
	logLH	-2233.9000	-2237.0686	-2232.9004
	Area Difference	34.05%	31.91%	34.06%

Table 16: Results of Maximum Likelihood Algorithms with 2 phases for Real Data Traces

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
NASA	e1	0	-	2.49354E-14
	e2	0.8622	-	0.4946
	e3	0.9907	-	0.8509
	logLH	-644108	-	-606352
CallCenter	e1	2.00923E-15	-	4.63668E-15
	e2	0.2368	-	0.0063
	e3	0.5924	-	0.0743
	logLH	-175579	-	-175014

A.2.2 Results for 4-phases Distributions

In this section, the results for the Maximum Likelihood Algorithm assuming 4-phase distributions are presented. The Measures of Performance for each algorithm over each trace can be found in Tables 17, 18, 19, 20, 21 and 22.

Table 17: Results of Maximum Likelihood Algorithms with 4 phases for Weibull Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
W1	e1	0	0	0
	e2	0.4096	0.0524	0.0036
	e3	1.3821	0.1744	0.0253
	logLH	-892.8130	-765.0364	-761.8578
	Area Difference	7.42%	1.00%	2.63%
W2	e1	0	0	0
	e2	0.0161	0.0030	0.0211
	e3	0.0254	0.0345	0.0385
	logLH	-1181.4837	-1190.3995	-1180.4900
	Area Difference	1.19%	1.09%	1.23%

Table 18: Results of Maximum Likelihood Algorithms with 4 phases for Lognormal Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
L1	e1	0	0	0
	e2	0.1066	0.3297	0.0911
	e3	0.2551	0.6627	0.2119
	logLH	-2981.9536	-2989.1799	-2980.9138
	Area Difference	0.92%	0.96%	0.91%
L2	e1	0	0	0
	e2	0.1295	0.0722	0.0270
	e3	0.4306	0.1822	0.1463
	logLH	-2357.5946	-2254.3216	-2248.1968
	Area Difference	5.66%	1.89%	9.58%
L3	e1	0	0	0
	e2	0.9208	0.6040	0.2005
	e3	4.3213	0.6609	0.6629
	logLH	-2029.6261	-2102.4599	-1338.0827
	Area Difference	20.47%	36.47%	13.36%

Table 19: Results of Maximum Likelihood Algorithms with 4 phases for Uniform Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
U1	e1	0	0	0
	e2	0.5128	0.1079	0.0985
	e3	2.0561	0.3999	0.3570
	logLH	-316.9736	-164.4281	-153.5563
	Area Difference	9.36%	3.77%	4.32%
U2	e1	0	0	0
	e2	0.9286	0.8406	0.2053
	e3	4.4006	0.9059	0.6877
	logLH	-1408.8661	-2977.5894	-713.1454
	Area Difference	21.04%	47.18%	13.08%

Table 20: Results of Maximum Likelihood Algorithms with 4 phases for Expo* Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
SE	e1	0	0	0
	e2	0.2759	0.0215	0.0261
	e3	0.8316	0.0453	0.0940
	logLH	-1409.2037	-1354.7100	-1339.6979
	Area Difference	19.06%	19.20%	24.79%
ME	e1	0	0	0
	e2	0.0581	0.0072	0.0221
	e3	0.1588	0.0869	0.0700
	logLH	-1085.9813	-1030.2481	-1015.3506
	Area Difference	5.67%	4.99%	7.69%

Table 21: Results of Maximum Likelihood Algorithms with 4 phases for Pareto Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
PI	e1	0	0	0
	e2	0.0303	0.1480	0.1764
	e3	0.1642	0.4490	0.5136
	logLH	-2135.4742	-2137.1586	-2135.2907
	Area Difference	0.68%	0.28%	9.56%
PII	e1	0	0	0
	e2	0.0378	0.3363	0.0204
	e3	0.1702	0.7191	0.0936
	logLH	-2229.8913	-2151.7707	-2131.2402
	Area Difference	33.80%	33.32%	38.86%

Table 22: Results of Maximum Likelihood Algorithms with 4 phases for Real Data Traces

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
NASA	e1	0	-	0
	e2	0.4946	-	0.0325
	e3	0.8509	-	0.1002
	logLH	-606353	-	-602614
CallCenter	e1	0	0	0
	e2	0.0009	0.2383	0.0070
	e3	0.0566	0.5919	0.0878
	logLH	-174957	-175640	-174956

A.2.3 Results for 8-phases Distributions

In this section, the results for the Maximum Likelihood Algorithm assuming 8-phase distributions are presented. The Measures of Performance for each algorithm over each trace can be found in Tables 23, 24, 25, 26, 27 and 28.

Table 23: Results of Maximum Likelihood Algorithms with 8 phases for Weibull Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
W1	e1	0	0	0
	e2	0.4096	0.0365	0.0018
	e3	1.3821	0.1269	0.0031
	logLH	-892.8130	-762.4475	-756.8369
	Area Difference	7.42%	0.82%	10.47%
W2	e1	0	0	0
	e2	0.0161	0.0061	0.0146
	e3	0.0254	0.0467	0.0218
	logLH	-1181.4837	-1193.5612	
	Area Difference	1.19%	1.09%	

Table 24: Results of Maximum Likelihood Algorithms with 8 phases for Lognormal Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
L1	e1	0	0	0
	e2	0.0435	0.3078	0.0897
	e3	0.0658	0.6339	0.2081
	logLH	-2981.9508	-2988.6785	-2980.9101
	Area Difference	0.94%	0.86%	0.92%
L2	e1	0	0	0
	e2	0.1295	0.0717	0.0319
	e3	0.4305	0.1803	0.1588
	logLH	-2357.5946	-2254.3313	-2230.7668
	Area Difference	5.66%	1.89%	26.49%
L3	e1	0	0	0
	e2	0.9208	0.6067	0.0804
	e3	4.3213	0.6672	0.2472
	logLH	-2029.6261	-2105.2111	-1063.6292
	Area Difference	20.47%	36.64%	8.90%

Table 25: Results of Maximum Likelihood Algorithms with 8 phases for Uniform Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
U1	e1	0	0	0
	e2	0.5128	0.1039	0.0326
	e3	2.0561	0.3852	0.1144
	logLH	-316.9736	-161.3988	-109.8480
	Area Difference	9.36%	3.66%	9.85%
U2	e1	0	0	0
	e2	0.9286	0.8411	0.0848
	e3	4.4006	0.9068	0.2658
	logLH	-1408.8661	-3002.8856	-433.1221
	Area Difference	21.04%	47.23%	8.13%

Table 26: Results of Maximum Likelihood Algorithms with 8 phases for Expo* Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
SE	e1	0	0	0
	e2	0.2759	0.0072	0.0183
	e3	0.8316	0.0057	0.0789
	logLH	-1409.2037	-1348.8209	-1330.1116
	Area Difference	19.06%	19.33%	17.80%
ME	e1	0	0	0
	e2	0.0581	0.0030	0.0024
	e3	0.1588	0.1029	0.0079
	logLH	-1085.9813	-1030.7300	-985.0109
	Area Difference	5.67%	4.98%	27.12%

Table 27: Results of Maximum Likelihood Algorithms with 8 phases for Pareto Distributions

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
PI	e1	0	0	0
	e2	0.0303	0.1358	0.0293
	e3	0.1642	0.4161	0.1674
	logLH	-2135.4742	-2136.4891	-2130.7689
	Area Difference	0.68%	0.27%	1.60%
PII	e1	0	0	0
	e2	0.0378	0.0432	0.0448
	e3	0.1702	0.0802	0.2126
	logLH	-2229.8913	-2139.7649	-2102.5566
	Area Difference	33.80%	32.89%	13.92%

Table 28: Results of Maximum Likelihood Algorithms with 8 phases for Real Data Traces

Trace	Measure	Method		
		EMHyperExpo	EMPhase	EMHyperErlang
NASA	e1	0	-	0
	e2	0.0326	-	0.0292
	e3	0.1006	-	0.0934
	logLH	-602615	-	-598599
CallCenter	e1	0	-	0
	e2	0.0136	-	0.0880
	e3	0.0497	-	0.3395
	logLH	-174955	-	-174325

References

[1] S. Asmussen, O. Nerman, and M. Olsson, “Fitting phase type distributions via the em algorithm,” *Scandinavian Journal of Statistics*, vol. 23, pp. 419,441, 1996.

[2] R. Khayari, R. Sadre, and B. Haverkort, “Fitting world-wide web request traces with the em-algorithm,” *Performance Evaluation*, vol. 52, pp. 175–191, 2003.

[3] A. Thümmler, P. Buchholz, and M. Telek, “A novel approach for fitting probability distributions to real trace data with the em algorithm,” in *Proceedings of the International Conference on Dependable Systems and Networks*, 2005.

[4] M. Telek and A. Heindl, “Matching moments for acyclic discrete and continuous phase-type distribution of second order.” *I.J. of Simulation*, vol. 3, no. 3–4, pp. 47–57, 2002.

[5] T. Osogami and M. Harchol, “Closed form solutions for mapping general distributions to quasi-minimal ph distributions,” *Performance Evaluation*, 2006, to appear.

[6] A. Bobbio, A. Horvath, and M. Telek, “Matching three moments with minimal acyclic phase type distributions,” *Stochastic Models*, vol. 21, pp. 303–326, 2005.

[7] J. Muppala, R. Fricks, and T. K., “Techniques for system dependability evaluation,” in *Computational Probability*, W. Grassmann, Ed. Kluwer Academic Publishers, 2000.

[8] M. F. Neuts, *Matrix-Geometrix Solutions in Stochastic Models.* The John Hopkings University Press., 1981.

[9] D. Cox, “A use of complex probabilities in the theory of stochastic processes,” *Proceedings of the Cambridge Philosophical Society*, vol. 51, pp. 313–319, 1955.

[10] M. A. Johnson and M. R. Taaffe, “Matching moments to phase type distributions: Nonlinear programming approaches,” *Comm. Statist. Stochastic Models*, vol. 2, no. 6, pp. 259–281, 1990.

[11] C. A. O’Cinneide, “On nonuniqueness of representations of phase-type distributions,” *Comm. Statist. Stochastic Models*, vol. 5, no. 2, pp. 247–259, 1989.

[12] M. Neuts and M. Pagano, “Generating random variates from a distribution of Phase-Type,” in *Proceedings of the Winter Simulation Conference*, 1981.

[13] G. Latouche and V. Ramaswami, *Introduction to matrix analytic methods in stochastic modeling.* Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM), 1999.

[14] M. F. Neuts, “Two further closure properties of PH-distributions,” *Asia-Pacific J. Oper. Res.*, vol. 9, no. 1, pp. 77–85, 1992.

[15] Y. Fang and I. Chlamtac, “Teletraffic analysis and mobility modeling for pcs networks,” *IEEE Transactions on Communications*, vol. 47, no. 7, pp. 1062–1072, 1999.

[16] A. Feldmann and W. Whitt, “Fitting mixtures of exponentials to long-tail distributions to analyze network performance models,” *Performance Evaluation. An international journal*, vol. 31, pp. 245–279, 1998.

[17] A. Horvath and M. Telek, “Approximating heavy-tailed behaviour with phase-type distributions,” in *Advances in Algorithmic Methods for Stochastic Models*, G. Latouche and P. Taylor, Eds. Notable Publications, Inc, 2000, pp. 191–213.

[18] A. Lang and J. Arthur, “Parameter approximation for phase-type distributions,” in *Matrix Analytic methods in Stochastis Models*, S. Chakravarty, Ed. Marcel Dekker, Inc., 1996.

[19] C. H. Sauer and K. M. Chandy, “Approximate analysis of central server models.” *IBM Journal of Research and Development*, vol. 19, no. 3, pp. 301–313, 1975.

[20] R. Augustin and K. Büscher, “Characteristics of the cox-distribution,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 12, no. 1, pp. 22–32, 1982.

[21] M. A. Johnson and M. R. Taaffe, “Matching moments to phase distributions: mixtures of Erlang distributions of common order,” *Comm. Statist. Stochastic Models*, vol. 5, no. 4, pp. 711–743, 1989.

[22] ——, “Matching moments to phase distributions: density function shapes,” *Comm. Statist. Stochastic Models*, vol. 6, no. 2, pp. 283–306, 1990.

[23] ——, “An investigation of phase-distribution moment-matching algorithms for use in queueing models,” *Queueing Systems*, vol. 8, pp. 129–147, 1991.

[24] M. A. Johnson, “Selecting parameters of phase distributions: combining nonlinear programming, heuristics and erlang distributions,” *ORSA Journal on Computing*, vol. 5, pp. 69–83, 1993.

[25] L. Schmickler, “Meda: Mixed erlang distributions as phase-type representations of empirical distribution functions,” *Stochastic Models*, vol. 8, pp. 131–156, 1992.

[26] T. Osogami and M. Harchol, “Necessary and sufficient conditions for representing general distributions by coxians,” in *Proceedings of the TOOLS 2003*, 2003.

[27] ——, “A closed form solution for mapping general distributions to minimal ph distributions,” in *Proceedings of the TOOLS 2003*, 2003.

[28] A. Bobbio and A. Cumani, “ML estimation of the parameters of a PH distributions in triangular canonical form,” in *Computer Performance Evaluation*, G. Balbo and G. Serazzi, Eds. Elsevier Science Publishers, 1992, pp. 33–46.

[29] A. Bobbio and M. Telek, “A benchmark for ph estimation algorithms: results for acyclic-ph,” *Stochastic Models*, vol. 10, pp. 661–667, 1994.

[30] A. Cumani, “On the canonical representation of homogeneous markov processes modeling failure-time distributions,” *Microelectronics and Reliability*, vol. 22, no. 3, pp. 583–602, 1982.

[31] A. Dempster, N. Laird, and R. D.B., "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B*, vol. 39, pp. 1–38, 1977.

[32] A. Riska, V. Diev, and E. Smirni, "An em-based technique for approximating long-tailed data sets with ph distributions," *Performance Evaluation*, vol. 54, pp. 147–164, 2004.

[33] A. Bobbio, A. Horvath, M. Scarpa, and M. Telek, "Acyclic discrete phase type distributions: properties and a parameter estimation algorithm," *Performance Evaluation*, vol. 54, pp. 1–32, 2003.

[34] B. Heimsund, "MTJ: Matrix toolkit for java," http://rs.cipr.uib.no/mtj.

[35] A. Law and D. Kelton, *Simulation, Modeling and Analysis*. McGraw-Hill Higher Education, 2000.

[36] T. Gonzalez, S. Sahni, M. Neuts, and W. Franta, "An efficient algorithm for the kolmogorov-smirnov and lilliefors tests," *ACM transactions on Mathematical Software*, vol. 3, no. 1, pp. 60–64, 1977.

[37] C. Gourieroux and A. Monfort., *Statistics and Econometric Models*. Cambridge University Press, 1995, vol. 1, ch. 13 - Numerical Procedures, pp. 443–491.

[38] D. Aldous and L. Shepp, "The least variable phase-type distribution is erlang," *Stochastic Models*, vol. 3, pp. 467–473, 1987.

[39] ACM SIGCOMM, "Internet traffic archive," http://ita.ee.lbl.gov/index.html.

[40] M. Arlitt and C. Williamson, "Internet web servers workload characterization and performance implications," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 631–645, 1997.

[41] A. Mandelbaum, "Call center data," http://iew3.technion.ac.il/serveng/callcenterdata.