# Model-based clustering for aCGH data using variational EM

by

Guillaume Alain

B.Sc., The University of Ottawa, 2004
M.Sc., The University of Ottawa, 2006

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

August 2009

# Abstract

DNA copy number alterations (CNAs) are genetic changes that can produce adverse effects in numerous human diseases, including cancer. Copy number variations (of which CNAs are a subset) are a common phenomenon and not much is known about the nature of many of the mutations. By clustering patients according to CNA patterns, we can identify recurrent CNAs and understand molecular heterogeneity. This differs from normal distance-based clustering that doesn't exploit the sequential structure of the data.

Our approach is based on the *hmmmix* model introduced by [Sha08]. We show how it can be trained with variational methods to achieve better results and make it more flexible. We show how this allows for soft patient clusterings and how it partly addresses the difficult issue of determining the number of clusters to use. We compare the performance of our method with that of [Sha08] using their original benchmark test as well as with synthetic data generated from the *hmmmix* model itself. We show how our method can be parallelized and adapted to huge datasets.

# Table of Contents

# List of Tables

# List of Figures

# Acknowledgments

This work is based on Sohrab Shah's work with my M.Sc. supervisor Kevin Murphy. I thank them both for giving me the chance to work on this project. I am indebted to Kevin Murphy for taking me as student and to Sohrab Shah for providing me with an opportunity to work on a relevant biological problem without having background knowledge in the field.

# Introduction

The purpose of this thesis is to expand on Sohrab Shah's work on detecting shared chromosomal aberrations in cohorts of patients from aCGH data. His approach, presented in [Sha08] and [SJJ+09], which he names the *hmmmix* model, treats the gene-probes sequences as hidden Markov chains and clusters the patients to discover shared patterns of observations.

In this thesis, we investigate the idea that Shah's model would perform better and would be more flexible if it were trained using variational methods (see chapter 10 of [Bis06]). Exact inference with the model is intractable and it is currently trained using iterative conditional modes (ICM). We also discuss techniques to estimate the best number of clusters to be used to model the data.

We start in this chapter by explaining the context of the problem and giving a synthetic example based on the *hmmmix* model. The model itself is described briefly in section 1.1.3. Our particular contributions and the plan for the thesis follow right after.

## 1.1 Context and motivation

### 1.1.1 Copy number variations

DNA copy number variations (CNVs) are a natural phenomenon in which certain gene segments are replicated a higher or lower number of times compared to a reference genome. These changes can be responsible for various diseases such as cancer, CHARGE syndrome [JAVDD+06], Parkinson [SFJ+03] and Alzeihmer [RLHR+06], but not all CNVs are associated with disease. Those that are dangerous are referred to as copy number alterations (CNAs).

The introduction of array-based comparative genomic hybridization (array CGH) is responsible for the recent explosion in data, but one of the problems is that not much information is available about the nature of the identified CNVs. They are so prevalent that the proportion of any given chromosome susceptible to CNVs varies from 6% to 19% [RIF+06].

Different features of a population can be studied by looking at the CNV patterns of the individuals. An interesting experiment is presented in [RIF+06] where they cluster populations based on the CNV genotypes. They show how their results match those of similar experiments that put African, European and Asian populations in different clusters.

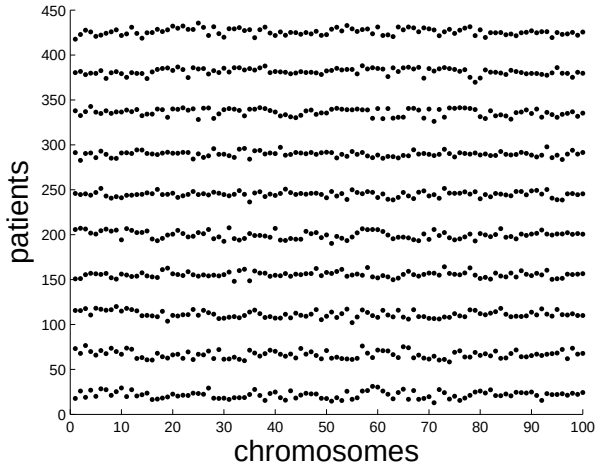
The same thing that be done for CNAs that result from human disease. Patients sharing a disease may exhibit the same characteristic CNAs and that knowledge may be used to identify people in the early stages of that disease. In that case, the pathogenic CNVs need to be sorted out from the abundant healthy CNVs that constitute background noise. By analyzing aCGH data for multiple patients simultaneously, we can have a more robust method to identify the recurrent CNA patterns. We can then cluster all patients based on these patterns. This can be used to shed some light into the mechanisms of certain types of cancers and it can be used to make better informed medical decisions.

### 1.1.2 Synthetic example

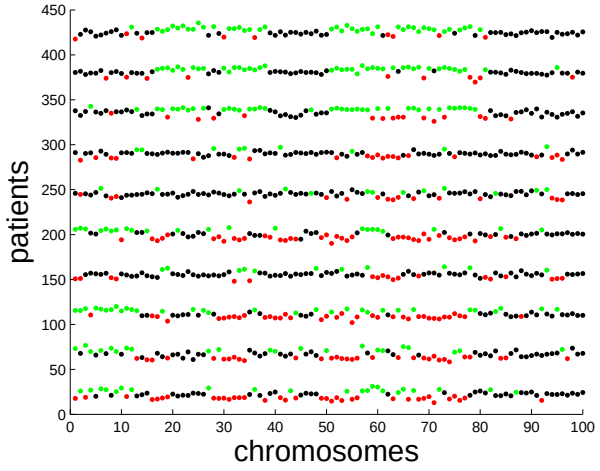
To illustrate the problem better before discussing possible solutions, we give here an example with synthetic data generated by the *hmmmix* model from [Sha08]. We have $P = 10$ patients for which we have sequences of observations $Y_{1:T}^{1:P}$ corresponding to gene expression measurements, where $T = 100$ is the length of the sequences. We show in figure 1.1(a) the raw data sequences. When dealing with more than one chromosome, we can concatenate them in sequence as long as we're careful to differentiate between them during learning (see section 4.6).

For all values $Y_t^p$, we postulate that there is a corresponding hidden state $Z_t^p \in \{\textit{loss, neutral, gain}\}$ whose discrete value determines the emission model from which $Y_t^p$ is drawn. These three states correspond respectively to CNA deletions, no alterations, and CNAs with higher copy numbers. We show in figure 1.1(b) the hidden states $Z_t^p$ by color-coding the observations from figure 1.1(a). The values of $Z_{1:T}^{1:P}$ are never observed directly and the reason that we have access to them is because we are dealing with artificial data for which we generated the hidden values before generating the observed data $Y_{1:T}^{1:P}$.

We want to infer those hidden values $Z_{1:T}^{1:P}$ in real experimental data, but we also want to discover shared patterns among patients. The idea is that there are certain underlying CNAs common to the members of certain cohorts of patients and that these CNAs are identifiable from the observations $Y_{1:T}^{1:P}$ given the right model. In this particular case, the patients have been generated according to $G = 3$ disease profiles (also referred to as cohorts, groups or clusters). We define one hidden Markov chain $M_{1:T}^g$ per disease profile $g = 1, \ldots, G$. The CNAs that characterize the different profiles are modeled by the discrete hidden Markov




(a) experimental measurements




(b) experimental measurements tagged by color using ground truth

Figure 1.1: These two figures show an example for 10 patients of the kind of data that we are interested in modeling. This data has been sampled from the *hmmmix* model itself so it may not be representative of real aCGH data, but it serves to illustrate the problem that we want to solve. Along the horizontal axis we have the measurements $Y^p_{1:T}$ that normally span across multiple chromosomes. For the sake of simplicity, we used sequences of length $T = 100$ with just one chromosome. We have on top (a) the raw data $Y^{1:P}_{1:T}$ as usually measured experimentally, while at the bottom (b) we color-coded the sequences based on the true values of $Z^{1:P}_{1:T}$, the CNVs identified in patients. Red segments corresponds to deletions and green segments to higher copy numbers. We are interested in learning this information. These hidden values are known to us here because we generated the data. The data has been generated by the *hmmmix* model using $G = 3$ clusters. The true cluster assignments place patients $\{1, 2, 3, 5\}$, $\{4, 6, 7\}$ and $\{8, 9, 10\}$ together, counting from the bottom up.

chains taking one of three values {*loss, background, gain*}.

We want to cluster the patients based on the values inferred for $Z^p_{1:T}$ and $M^g_{1:T}$, while refining our estimates of those values based on the clustering in progress. This way, we hope to recover the recurrent CNAs responsible for the diseases, while at the same time determining which patients are afflicted by which type of disease. We show at the top of figure 1.2 the CNA profile information learned by using the method of [Sha08]. The particular values of $Z^{1:P}_{1:T}$ for the patients are shown at the bottom of figure 1.2. The patients have been reordered after being clustered into the 3 profiles identified. The cluster numbers are in the right margin.

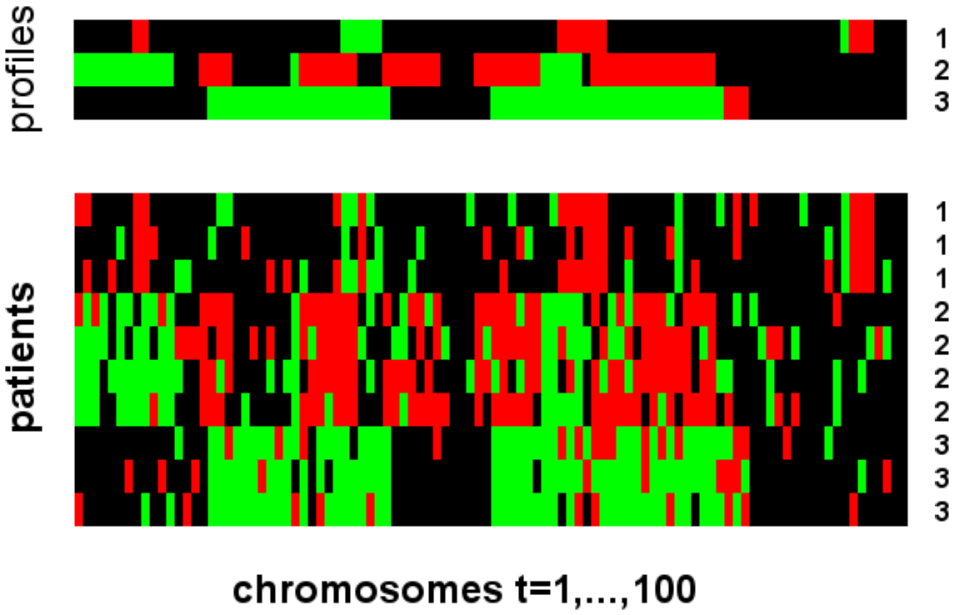



Figure 1.2: On top we show the recurrent CNAs that characterize the disease profiles $M^{1:G}_{1:T}$ found. Red corresponds to losses and green to gains. On the bottom we show the states $Z^{1:P}_{1:T}$ that are inferred by the *hmmmix-hard* algorithm from [Sha08] as well as the final clustering for the patients. The cluster indices are shown in the right margin (based on the profiles indices above) and the patients have been reordered so that those in the same cluster are together.

By clustering the patients using a model that takes into consideration the sequential structure of the data, we are able to exploit the fact that CNAs come in segments. This provides a good advantage over distance-based methods that can only take advantage of this fact indirectly by smoothing the imputed states after inference.

Bear in mind that in this section we presented data sampled from the very model that [Sha08] introduced to solve the problem. This circular demonstration is intended here as a means to illustrate the problem tackled by [Sha08] and us, and not as a justification for the *hmmmix* model itself. The *hmmmix* model has been validated by experiments with real data presented

in [Sha08] and [SJJ+09]. In this thesis we are mostly interested in finding ways to use the model in a better way.

### 1.1.3 The hmmmix generative model

The *hmmmix* model is a generative model that has $G$ hidden Markov chains and $P$ patients, with every patient assigned to one of the chains (we assume that $P \gg G$). Figure 1.3 shows the graphical representation of the model.

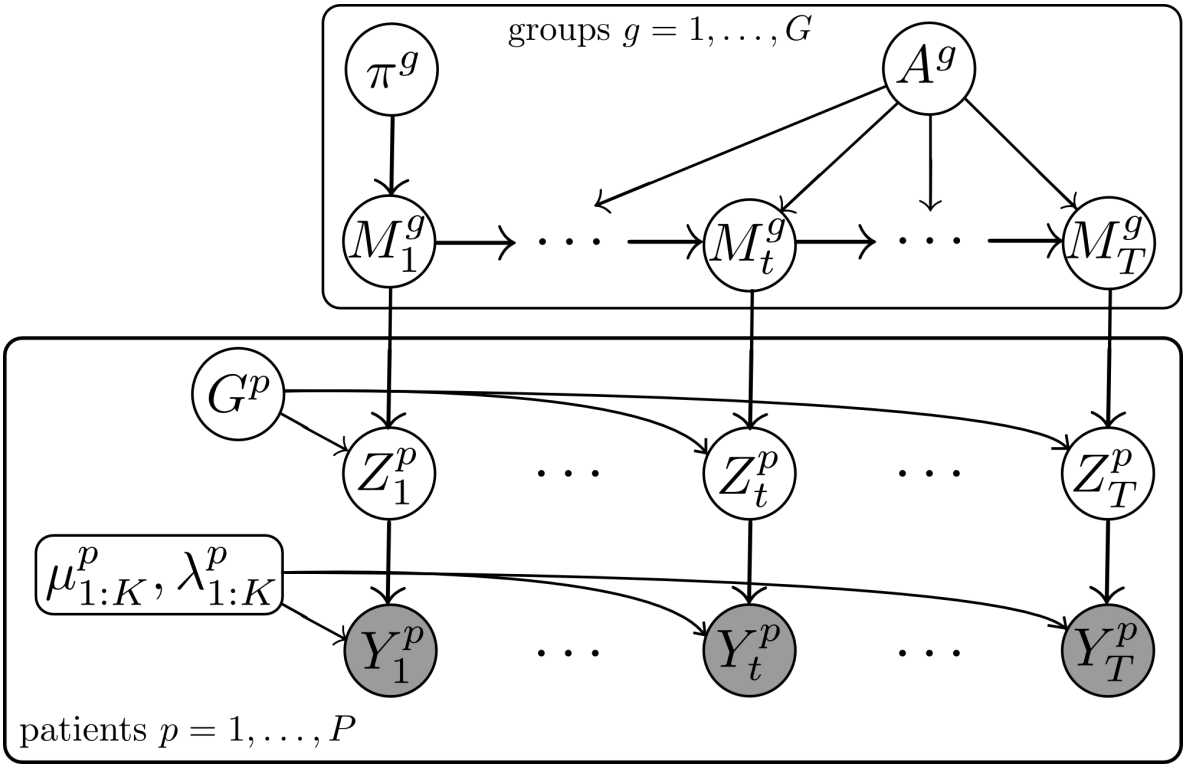



Figure 1.3: The *hmmmix* model. When generating data, the parameters $\pi^{1:G}, A^{1:G}, \mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$ are fixed. The parameters $\alpha_L, \alpha_N, \alpha_G$ are not shown here.

We now describe how the model (conceptually) generates data. For every group $g = 1, \ldots, G$, we have a corresponding hidden Markov chain of length $T$ whose states are represented by $M_{1:T}^g$. These chains have $K$ discrete states that, for the purposes of this work, can take one of $K = 3$ discrete values corresponding to different copy number alterations : {*1=loss, 2=background, 3=gain*}. For every $g$, we sample one full sequence $M_{1:T}^g$ from a hidden Markov chain with corresponding prior $\pi^g$ and transition matrix $A^g$. These parameters should be chosen to encourage the normal ($k = 2$) state and discourage state transitions.

For every patient $p = 1, \ldots, P$, we assign that patient to one of the groups $\{1, \ldots, G\}$ with equal probability. We let $G^p$ be the index identifying to which hidden Markov chain $p$ belongs. We then take the associated chain $M^g_{1:T}$ (for $G^p = g$) and we sample a sequence of hidden states $Z^p_{1:T}$ associated to that patient. Note that the $Z^p_{1:T}$ sequence is not a Markov chain. The discrete value of $Z^p_t$ is sampled from a multinomial whose parameters are given by one of $K$ predefined vectors $\alpha_L, \alpha_N, \alpha_G$ depending on the state $\{$*1=loss, 2=background, 3=gain*$\}$ of the source $M^g_t$.

$$p(Z^p_t = k | M^g_t = j, G^p = g) = \frac{\alpha_j(k)}{\alpha_j(1) + \alpha_j(2) + \alpha_j(3)} \tag{1.1}$$

We usually define the parameter vectors as $\alpha_L = (a_L, 1, 1), \alpha_N = (1, a_N, 1), \alpha_G = (1, 1, a_G)$ for an appropriate choice of $a_L, a_G \geq a_N \geq 1$. These parameters determine how likely the patients are to have the same hidden states as the ones found in the group to which they belong, that is, the "purity" of a given column. We referred to the copy number state $M^g_t = 2$ as *background* but to the state $Z^p_t = 2$ as *neutral*. By using a small value of $\alpha_N \geq 1$, the hidden states $M^g_t =$ *background* become agnostic about the corresponding states $Z^p_t$ of patients $p$ belonging to profile $g$. This is meant to encourage a certain sparsity for the CNAs identified. The states $Z^p_t =$ *neutral*, however, simply represent the fact that the corresponding observations $Y^p_t$ will be most likely found to higher than those from $Z^p_t =$ *loss* and lower than those from $Z^p_t =$ *gain*.

We generate the observations $Y^{1:P}_{1:T}$ given $Z^{1:P}_{1:T}$ according to

$$p(Y^p_t | Z^p_t = k) = \text{Student}\left(Y^p_t | \text{mean}{=}\mu^p_k, \text{precision}{=}\lambda^p_k, \text{degrees of freedom}{=}\nu^p_k\right) \tag{1.2}$$

where

$$\text{Student}\,(y|\mu, \lambda, \nu) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)} \sqrt{\frac{\lambda}{\nu\pi}} \left[1 + \frac{\lambda}{\nu}(y - \mu)^2\right]^{-\frac{\nu+1}{2}} \tag{1.3}$$

In [Sha08] the constant $\alpha_N$ is referred to as $\alpha_0$. The model handles data from probes expected to be in the *background* state a bit differently, but we don't do this distinction here during training. The *hmmmix* model is also defined in [Sha08] using a "compound Dirichlet-Multinomial" where the essential parameters are marginalized. We chose to omit this step as the resulting distribution simplifies into a basic multinomial.

### 1.1.4 Biclustering

As mentioned in the previous sections, we are be using a model-based approach instead of a distance-based approach in order to exploit the structure of the data. The method of "biclustering" introduced by [CC00] in this context is another popular tool for expression

data analysis. It tries to find relations between genes and conditions, represented by the rows and columns of a matrix.

In [MTSc+07] is a fast algorithm to perform biclustering based on the sequential structure of the data. It operates under the assumption that the biclusters found will be contained in contiguous columns. This is similar to the construction of *hmmmix* that encourages the hidden chains representing CNAs to have segments where the state is constant. However, it assumes that the data is discrete and noise-free.

We will not be discussing biclustering in this thesis, though.

## 1.2 Our contributions

Our contributions to the *hmmmix* model are three-fold. First, we use variational methods to train the model. The most interesting consequence of this is the possibility to partially assign patients to multiple clusters. We expect this to improve on hard clustering, just as EM for Gaussian mixtures improves on K-means. We derive the new update formulas to train the model and we analyze its runtime performance.

Secondly, we show how variational methods open the way to other initialization methods than the k-medoids method used in [Sha08]. This was a serious limitation in their model as it required the number of clusters to be selected a priori. We show how the Bayesian information criterion (BIC) fails in this particular setting and how the minimum description length (MDL), the silhouette coefficient and the gap statistic can be used instead. We also discuss the addition of scaling constants to the variational entropy terms in order to smooth out the objective function.

Finally, we give a low-memory implementation made to handle large datasets. We compare the performance of both on small and large datasets. We have also rewritten the forwards-backwards algorithm in C to speed up by a factor between 10 and 100 the original code from [Sha08], which was written in Matlab.

## 1.3 Outline of the thesis

In chapter 2, we describe how the hmmmix model is trained in [Sha08]. We present an experiment with the silhouette coefficient and one with BIC to determine the best number of clusters to use.

In chapter 3 we show how it is possible with variational methods to extend the model to remove some of the constraints that [Sha08] has. We derive the formulas relevant to our method.

In chapter 4 we compare our method with that of [Sha08] using his benchmarks on artificial data and by comparing visually the results on real data. We present an experiment with MDL and the gap statistic to determine if the true number of clusters can be recovered practically with either [Sha08]'s method or our own. We also analyze the computational/memory complexity of our algorithm and discuss how it scales to accommodate the very large datasets that are common in biology.

Finally, in chapter 5, we conclude with a general discussion and point to potential improvements.

# Chapter 2

# Description of the current model

In this section we explain how the hmmmix model is used in [Sha08] and [SJJ+09].

We follow certain naming conventions in order to tell apart the model in [Sha08] from ours, and to tell the difference between the statistical models themselves and the various methods to train them. What we refer to as *hmmmix* is a generative model with various parameters. Exact inference in that model is very hard and various approximations can be done to simplify this task. In [Sha08] a certain technique is used for that, and we will refer to it as *hmmmix-hard* as it performs hard cluster assignments to patients and uses the Viterbi algorithm to impute values to the hidden Markov chains.

In this thesis, we start again from the *hmmmix* model but we solve the learning problem by using variational inference. We call our method *hmmmix-soft* in reference to the partial cluster assignments for patients. The underlying model is still the *hmmmix* model but we approach it differently.

The core of the *hmmmix* model was presented in section 1.1.3. We introduce priors in section 2.1 for some of the parameters. An important component of the model comes in choice of the emission distributions for the observations and the method we use learn the parameters. For this, [Sha08] borrows from [Arc05] but the actual formulas from the latter have to be adapted to our particular case.

We discuss in section 2.3 the choice of objective function in the perspective of model selection to identify the best number of clusters to use. We present in section 2.4 an experiment to determine the number of clusters using the silhouette coefficient. In section 2.5, we show why BIC cannot be used for this task.

## 2.1 Priors for the parameters

There are two places in this model where priors are placed on parameters. The first place is when estimating the transition matrices involved in the hidden chains. We put a Dirichlet prior on every row of $A^g$ with parameters picked to encourage same-state transitions. This

essentially corresponds to putting pseudocounts on the transition matrices. More details on this are found in section 3.1 when we put the forwards-backwards algorithm in context.

The second place where priors are used when we are estimating the parameters $\theta = \{\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}\}$ that take part in $p(Y_{1:T}^{1:P}|Z_{1:T}^{1:P}, \theta) = \prod_t \prod_p p(Y_t^p|Z_t^p, \theta)$. There are slight divergences in notation between [Sha08] and [Arc05] as well as some imprecision concerning the roles that the hyperparameters play. We decided to follow [Arc05] whenever possible and to be explicit about the density functions whenever confusion could arise. This is important as we will be using these definitions in section 3.3.4 when addressing how to learn the parameters from *hmmmix-soft*. It should be understood that these priors are there to give the model some flexibility and there might be better ways to train the values of $\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$ when we have domain-specific knowledge.

We factor the prior as $p(\mu, \lambda) = p(\mu|\lambda)p(\lambda)$. On the precisions $\lambda_k^p$, we put a Gamma prior with hyperparameters $\gamma_k^p, S_k^p$ also indexed by $k, p$ :

$$p\left(\lambda_k^p|\gamma_k^p, S_k^p\right) = \text{Gamma}\left(\lambda_k^p|\gamma_k^p, S_k^p\right) \tag{2.1}$$

where

$$\text{Gamma}\left(x|a, b\right) = \frac{1}{\Gamma(a)} b^a x^{a-1} \exp\left(-bx\right). \tag{2.2}$$

On the means $\mu_k^p$, we put a normal prior with hyperparameters $m_k^p, \eta_k^p$ :

$$p\left(\mu_k^p|m_k^p, \eta_k^p, \lambda_k^p\right) = \mathcal{N}\left(\mu_k^p|\text{mean}{=}m_k^p, \text{precision}{=}\eta_k^p\lambda_k^p\right) \tag{2.3}$$

where

$$\mathcal{N}\left(x|\text{mean}{=}\mu, \text{precision}{=}\lambda\right) = \sqrt{\frac{\lambda}{2\pi}} \exp\left(-\frac{\lambda}{2}\left(x-\mu\right)^2\right). \tag{2.4}$$

It should be noted that, in [Sha08], the hyperparameters $m_k^p, \nu_k^p$ are only indexed as $m_k, \nu_k$. From personal correspondence with the author, we learned that *hmmmix-hard* actually used patient-specific parameters so we thought it would be better to present the model in this fashion. As those values are not learned during inference, we can always assign the same values across patients and it will be equivalent to using the $m_k, \nu_k$ indexing.

Finally, there is a certain amount of preprocessing done on the experimental data before it is used to learn the *hmmmix* model. This preprocessing step has more to do with instrument calibration than with statistics. The effects of that are reflected in the values of $m_k^p$ that we use and they should be consistent with our choice of labels for the hidden states. With a good prior respecting at least the equality $m_1^p \leq m_2^p \leq m_3^p$, it should come naturally that $\mu_1^p \leq \mu_2^p \leq \mu_3^p$ without having to enforce it explicitly.

## 2.2 Inference with hmmmix-hard

The patients are initially clustered within a specified number of groups with a method described in [Sha08]. This method imputes values to the variables $Z_{1:T}^{1:P}$ and then uses the k-medoids algorithm to get the initial assignments. From then on, the *hmmmix-hard* algorithm proceeds by the iterative conditional modes (ICM) algorithm, updating the hidden chains $M_{1:T}^{1:G}$, the patients assignments $G^{1:P}$ and the parameters $\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$ until convergence.

These updates are done by assigning patients to the single group whose hidden chain "explains" best the observations. The updates to the hidden chains are done by imputing values to $M_{1:T}^{1:G}$ using the Viterbi algorithm to get the most likely sequence of states explaining the observed values for the patients assigned to that group. We refer the reader to [Sha08] for the actual formulas.

To estimate the parameters, the priors from [Arc05] are used. We are using the same priors in section 3.3.4, although we use a slightly different approach. The reader can again refer to [Sha08] and [SJJ$^+$09] for the essential formulas, to [Arc05] for the original derivations or simply to section 3.3.4 of this thesis.

There are two reasons why the k-medoids algorithm is used instead of the classic k-means. The first is that k-means is not defined on discrete data, and the second is that k-medoids does the clustering with a distance matrix that computes distances between points only $\frac{P(P-1)}{2}$ times. The cost of each evaluation is prohibitive when we are dealing with long sequences of genes. The cost of running k-means (or EM) to initialize the algorithm can become as expensive as running the algorithm itself, so multiple restarts of k-means are not an option. With k-medoids, the same distance matrix can be used for multiple restarts of the algorithm.

## 2.3 Objective functions and model selection

For the rest of this section, we will denote by $C$ the number of clusters used (currently denoted by $G$) to avoid potential confusion with the hidden variables $G^{1:P}$. The meaning of $G$ is unambiguous in a context where it is always accompanied by $G^p$ or $G^{1:P}$ when referring to clusters to which patients are assigned, but here we want to be able to drop some indices from the current discussion to avoid clutter. We are interested in determining what is the optimal number of clusters $C$ to use. This becomes a model selection problem.

Conceptually, when learning the *hmmmix* model we are learning the parameters $A^{1:C}, \pi^{1:C}, \theta$

that maximize the likelihood

$$\max_{A,\pi,\theta} \sum_{M} \sum_{G} p(Y, M, G|\theta, A, \pi). \tag{2.5}$$

By using priors on the parameters, we are actually maximizing

$$\max_{A,\pi,\theta} \sum_{M} \sum_{G} p(Y, M, G, \theta, A, \pi), \tag{2.6}$$

but the important feature is that we are integrating over the unknown chains $M_{1:T}^{1:C}$ and patient assignments $G^{1:P}$.

By proceeding by ICM, *hmmmix-hard* imputes values for $M_{1:T}^{1:C}$ and $G^{1:P}$ while maximizing the likelihood with respect to the parameters $A, \pi, \theta$. We are essentially treating the hidden variables $M_{1:T}^{1:C}, G^{1:P}$ as though they were parameters like the rest, and maximizing

$$\max_{A,\pi,\theta} \max_{M} \max_{G} p(Y, M, G, \theta, A, \pi), \tag{2.7}$$

instead of (2.6). There are two issues with this. First, we have to ask ourselves if we are interested in the quantity (2.7) itself instead of the one at line (2.6). Secondly, ICM might be a terrible way to maximize (2.7) in the context of the *hmmmix* model. Without the possibility of maximizing (2.7) exactly, it's hard to determine how efficient *hmmmix-hard* is (we return to that topic in 4.1). We think that *hmmmix-soft* can perform better in situations where *hmmmix-hard* gets stuck quickly in bad local maxima.

Coming back to the first issue, the original problem is to identify CNAs and cluster patients to discover shared patterns. The objects that we study are marginalized in (2.6) and we are left with the parameters $A, \pi, \theta$ that are not of much use. We are very much interested in finding out where the CNA segments are. That information is present in the imputed chains $M_{1:T}^{1:C}$ that we get from *hmmmix-hard*, but not in the MAP estimates of $A, \pi$.

Instead of taking the maximum over the parameters in (2.6), we could integrate them out as

$$\int_{A,\pi,\theta} \sum_{M} \sum_{G} p(Y, M, G, \theta, A, \pi). \tag{2.8}$$

However, we have two objections against integrating out the parameters, not mentioning the objection about using $\sum_G \sum_M$ instead of $\max_G \max_M$ expressed earlier.

First, the forwards-backwards algorithm works with fixed constants $A, \pi$ and not with distributions. If we didn't have a good prior for the transition matrices $A$ and we tried to evaluate (2.8) by sampling values of $A$, most of the transition matrices obtained would be completely incompatible with our data that we expect to contain only rare transitions between states. If

we had a prior that accounted for this, then the posterior distribution would be peaked and a pointwise estimate of $A$ would be fine. This is exactly what we get by using $\max_{A,\pi}$ instead of $\int_{A,\pi}$.

The other objection about marginalizing over the parameters $\theta$ is that we are using specific hyperpriors tailored to our problem. For example, we set the hyperpriors $m_{1:K}^{1:P}$ to the values that we hope the parameters $\mu_{1:K}^{1:P}$ will take in the absence of overwhelming evidence. This defeats the philosophical ideal of integrating over the parameters to keep the model as agnostic as possible.

Those objections notwithstanding, with equation (2.8) we can compare the likelihood of the data under the models with different numbers of hidden chains $M_{1:T}^{1:C}$. The likelihood is given by

$$p(C|Y) \propto p(Y|C) = \int_{A,\pi,\theta} \sum_{M_{1:T}^{1:C}} \sum_{G} p(Y, A, \pi, \theta, M_{1:T}^{1:C}, G). \tag{2.9}$$

As argued earlier, another equally valid but more convenient alternative is

$$p(C|Y) \propto p(Y|C) = \max_{A,\pi,\theta} \sum_{M_{1:T}^{1:C}} \sum_{G} p(Y, A, \pi, \theta, M_{1:T}^{1:C}, G). \tag{2.10}$$

With the *hmmmix-soft* algorithm introduced in chapter 3, we use variational approximations in order to maximize a lower bound approximation on (2.10). We get distributions $h(M_{1:T}^{1:C})$ and $h(G^{1:P})$ that behave somewhat like the quantities $M_{1:T}^{1:C}, G^{1:P}$ imputed by ICM by *hmmmix-hard* to maximize (2.7). We update them in the same fashion as *hmmmix-hard* and when they are peaked they are similar to the values from *hmmmix-hard*.

We explain in section 3.4 one method to convert the variational approximations from *hmmmix-soft* to imputed values $M_{1:T}^{1:C}, G^{1:P}$. This allows us to assess *hmmmix-soft*'s performance in terms of maximizing (2.7) as *hmmmix-hard* does, and it also illustrates why the latter can be improved.

If we could work with (2.9) instead of (2.10), we could use the distribution $p(C|Y) \propto p(Y|C)$ to determine the true number of clusters with the Bayesian equivalent of Occam's razor. Assuming that this were also true with (2.10), nothing guarantees that our variational lower bound on (2.10) obtained by *hmmmix-soft* is tight (it should not be since we made assumptions about the factoring). It would be hard to argue that, upon getting that $p(C = 10|Y) = 0.3$ and $p(C = 11|Y) = 0.28$ we should select $C = 10$ and risk being one cluster short when the penalty for being one short is potentially worse than for being one over.

The Bayesian approach to $p(C|Y) \propto p(Y|C)$ doesn't work with equation (2.7) because more clusters just lead to overfitting. In the worst case, we could wind up with one patient per

cluster. In section 2.5 we discuss the use of BIC to solve this problem, but before that we show in section 2.4 how [Sha08] use the silhouette coefficient to determine the best number of clusters.

## 2.4 Determining the number of clusters with the silhouette coefficient

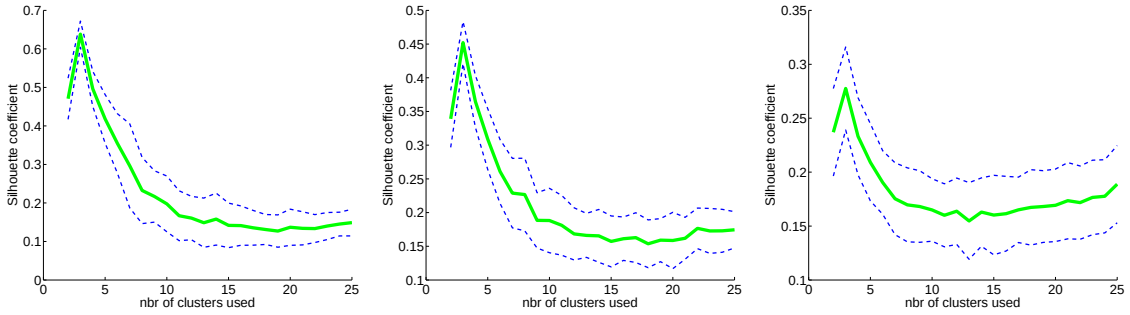
One of the important limitations of the *hmmmix-hard* approach is that we have to specify in advance the number of clusters to be used. This is essential for the k-medoids algorithm and the resulting clustering contains only hard assignments to clusters.

To get around this issue of determining the number of clusters, [Sha08] use the *silhouette* coefficient as a measure of quality (see appendix B for definition). They impute the values of the hidden $Z_{1:T}^{1:P}$ and use them as input for k-medoids. The quality of the initial clustering naturally depends on the quality of the imputed values of $Z_{1:T}^{1:P}$.

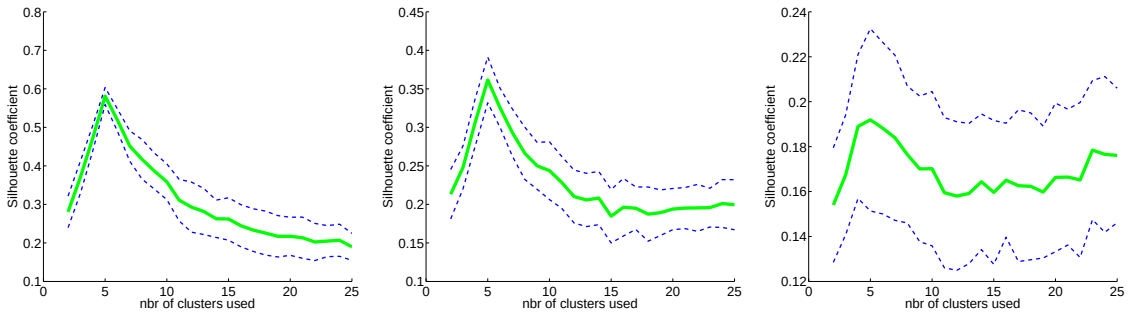
We ran our own simulation to see if we could rely on the silhouette coefficient to estimate the number of clusters in the case of the synthetic data generated by [Sha08]. We used their code to generate data with different numbers of clusters $G = 3, 5, 10$ and different lengths $L = 25, 50, 75$ for the copy number alterations. We looked at the silhouette coefficients obtained when selecting the best clustering from k-medoids with $G = 2, \ldots, 25$ clusters. We used the same parameters as [Sha08] to explore the nine combinations and averaged over 50 datasets each time. The results are in Figure 2.1.

From this we see that in many of the simpler cases, the silhouette coefficient with k-medoids identifies correctly the true number of clusters. It peaks at the true values and the variance in the results is relatively small. However, in 2.1(f) the results are unreliable due to high variance, in (h) the peak is hard to identify even when averaging multiple runs and in (i) the silhouette coefficient is off track, probably because the k-medoids algorithm fails to produce any good clustering. This is somewhat counter-intuitive as we would have expected better results with larger values of $L$. It might be related with the fact that the k-medoids clustering is done with imputed values of $Z_{1:T}^{1:P}$ instead of the $Y_{1:T}^{1:P}$ and the quality of those values $Z_{1:T}^{1:P}$ might decrease when the length of the inserted CNAs increases.

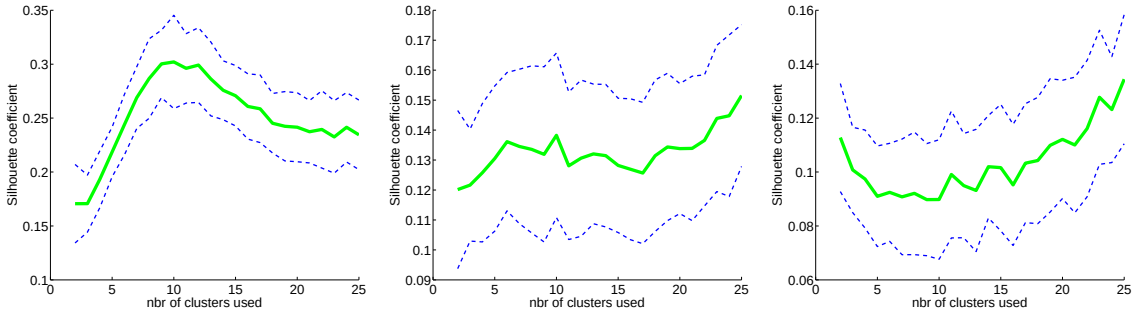
These graphs were produced from artificial datasets obtained by taking real data and inserting mutations in them of appropriate lengths. Experimental data does not have a true number of clusters and the value found by the silhouette coefficient does not guarantee that we will do any good for any other kind of measure of performance (such as imputing the $Z$, the $M$ or predicting $Y$).




(a) G=3,L=25 (b) G=3,L=50 (c) G=3,L=75




(d) G=5,L=25 (e) G=5,L=50 (f) G=5,L=75




(g) G=10,L=25 (h) G=10,L=50 (i) G=10,L=75

Figure 2.1: The silhouette coefficient (vertical axis) depending on the number of clusters used (horizontal axis) for different artificial datasets. The number G is the true number of clusters in the datasets and L is the length of the alterations inserted. The solid green curve is the average of the results for 50 datasets and the dotted blue curves show the standard deviation. The silhouette coefficient is a good way to detect the real number of clusters for those artificial datasets when the values of (G,L) are small, but it's not reliable in cases (f), (h), (i) as the curves don't peak at the true values of G.

## 2.5 Bayesian information criterion and minimum description length

The Bayesian information criterion (BIC) is introduced in [Sch78] and defined as

$$\log p_{\mathcal{H}}(X_1, \ldots, X_n | \hat{\theta}^{\mathrm{MLE}}_{\mathcal{H}}) - \frac{1}{2} k \log(n) \tag{2.11}$$

where $\{X_1, \ldots, X_n\}$ is the data observed, $\log p_{\mathcal{H}}(X_1, \ldots, X_n | \hat{\theta}^{\mathrm{MLE}}_{\mathcal{H}})$ is the maximum value for the log-likelihood under model $\mathcal{H}$ and $k$ is the number of parameters that model $\mathcal{H}$ has. To select the best model $\mathcal{H}_1, \mathcal{H}_2, \ldots$, we pick the one that maximizes BIC. The size penalty for the model acts like Occam's razor that selects the simplest model that explains the data.

There is something unsatisfying about the silhouette coefficient in section 2.4 because it does not fit into the statistical framework of the model. With BIC we add a penalty term to the log-likelihood of the data in order to do model selection, but we can't do the equivalent for the silhouette coefficient. In fact, when [Sha08] compare *hmmmix-hard* to existing methods with their benchmark test that uses synthetic data, they omit this step and assume that the true number of clusters G is known by their k-medoids clustering algorithm.

A first objection against BIC is that it requires us to train models of various sizes to determine the best number of clusters to use ultimately. This is prohibitive in our case because we are dealing with large datasets. One approach could be to reuse partial results from models of different sizes to train others, but this first issue isn't the real problem with BIC and the *hmmmix* model.

The fundamental problem is the fact that the parameters of *hmmmix* are redundant. As explained in section 2.3, the *hmmmix-hard* algorithm treats the hidden chains $M^{1:G}_{1:T}$ and patient assignments $G^{1:P}$ as parameters. The BIC penalty term is then

$$\frac{1}{2}(GT + GK^2 + P + KP) \log(PT) \approx \frac{1}{2} GT \log(PT). \tag{2.12}$$

Considering that $T$ dominates all other terms (in our particular application), the penalty term is essentially $\frac{1}{2} GT \log(PT)$.

However, this is misleading because the chains $M^g_{1:T}$ can be encoded with much less than $T$ parameters due to the fact that state transitions are uncommon. Their rare occurrence is part of the assumptions that we make based on the fact that we are modeling CNAs. Individual patients have all kinds of CNVs in their chromosomes, but the CNAs that we want to identify in patient cohorts are supposed to be sparse.

Hypothetically, we could encode the chains imputed by *hmmmix-hard* with one parameter for the initial state and two parameters for each transition, indicating when the transition

occurs and to what state we are switching. This would translate into representations using a varying number of parameters, with "most" sequences (in the probabilistic sense) being represented by very few parameters. With this in mind, we should leave BIC behind and use the minimum description length (MDL) instead. We discuss MDL in section 4.1 after *hmmmix-soft* is introduced.

We conclude here with a numerical example to support our case against BIC. We train *hmmmix-hard* using data sampled from the *hmmmix* model itself. The values of the hyperparameters used to sample the data are also given to *hmmmix-hard*. Using $G = 10$ groups as ground truth, we compare in figure 2.2 the log-likelihood contributions for model sizes $G = 6, \ldots, 16$ to the same log-likelihood that includes the BIC penalty term. We used k-medoids to initialize the clusters. The plot shows that the BIC penalty term is of the wrong order of magnitude. It overshadows the log-likelihood term $\log p(Y_{1:T}^{1:P} | \cdots)$ as it makes the solid curve in figure 2.2 decrease monotonically without anything special happening around the true value of $G = 10$.

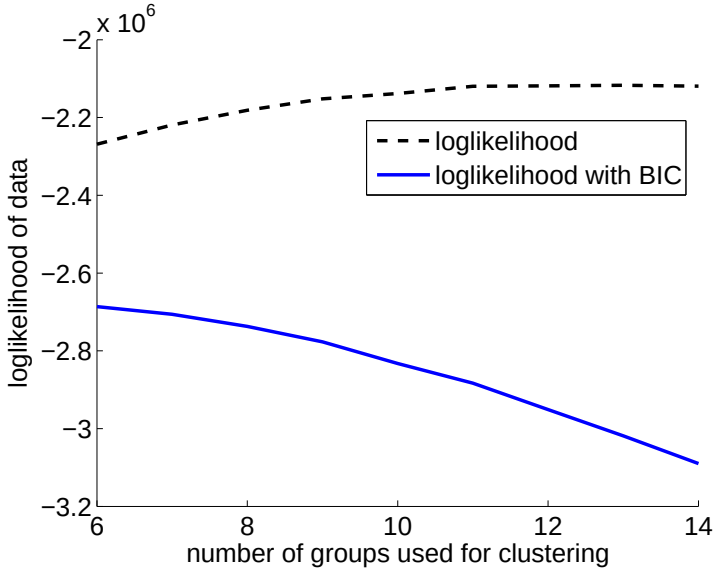



Figure 2.2: We show here the log-likelihood for data generated by the *hmmmix* model using $G = 10$ clusters, $P = 100$ patients and sequences of length $T = 10\ 000$. The dotted curve shows the log-likelihood alone while the solid curve shows the log-likelihood with the BIC penalty term. The model was trained by *hmmmix-hard* with initial clustering by k-medoids (using $G = 6, \ldots, 14$ clusters). To get a smooth curve, we averaged the results over 30 trials. If BIC was a good way to recover the true number of groups, we would expect the solid curve to peak at $G = 10$.

# Chapter 3

# Generalization of hmmmix

Inference is difficult in *hmmmix* model mostly due to the fact that the hidden chains are coupled because of *explaining away* (see figure 3.1). This makes learning intractable unless we are willing to make concessions. [Sha08] imputes values to the hidden $G^{1:P}$ to break the ties between the hidden chains. With the chains uncoupled, the Viterbi algorithm is used to select the most likely sequence of states for hidden chains separately.

The *hmmmix-hard* model gets good results when used with a decent initialization of patients assignments, but it tends to get stuck in very bad local maxima when not initialized properly. In this chapter, we will show how we can use variational methods to make it more flexible and better in some cases (but never worse). The actual benchmarks are found in the next chapter.

An interesting consequence of our approach is that we will be dealing with soft patient assignments. The difference between hard patients assignments and soft patients assignments is very similar to the case of k-means versus Gaussian mixture models learned by EM.

A very insightful paper on variational methods used for a related model called *factorial hidden Markov chains* is [GJ97]. We are dealing with a model a bit more involved than in [GJ97] so we have to re-derive the formulas for our particular case.

At the core of the *hmmmix-soft* algorithm is a variational lower bound for the log-likelihood of the data that we want to maximize. This is a technique that maximizes the log-likelihood itself, but we are more interested in the variational approximations to the hidden chains because they are supposed to represent the underlying biological phenomenon that we are studying. The final log-likelihood quantity whose value we optimize marginalizes over these hidden chains. Having optimal values for the transition matrices $A^{1:G}$ is not half as interesting as knowing where the state transitions are located. Again, we can draw a comparison with the case of Gaussian mixture models trained with EM where we are very interested in seeing which data points lie in which cluster, but we might not even care about the final log-likelihood. The relationships between the different quantities optimized are discussed in section 2.3.

A crucial optimization step in section 3.3.2 requires a good eye to identify certain quantities

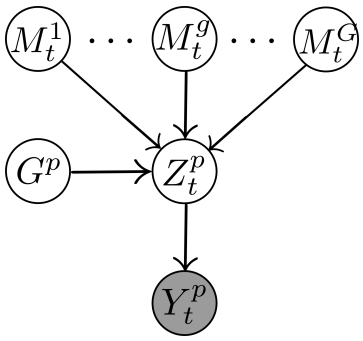



Figure 3.1: If we take a slice at time step $t$ of the graphical model found in figure 1.3, we can see how the hidden chains $M_t^1, M_t^g, M_t^G$ are dependent given the observed value $Y_t^p$. We omitted parameters here to simplify the graph. This coupling of the hidden chains is what makes exact inference intractable.

as being the log-likelihood of a hidden Markov chain with given observations. To simplify the exposition of our model, we felt that it would be appropriate to devote section 3.1 to derive certain formulas in a context where we have only one chain. That way, it will be easier to untangle expressions involving $G$ chains. Section 3.1 should simply be viewed a collection of useful lemmas.

Section 3.2 sets up the context for our model. In section 3.3 we derive the formulas for updating the parameters. We show in section 3.4 one possible way to get good imputed values from the variational approximations. In section 3.5, we sketch a map of the approximations that were used to make the original model tractable and discuss in more depths those pertaining to the parameters $\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$.

## 3.1 Note on forwards-backwards

Let $M_{1:T}$ be a Markov chain with $K$ states, with transition matrix $A$ and initial state priors given by a multinomial of parameter $\pi$. Moreover, for all hidden states $M_t$ at time $t \in \{1, \ldots, T\}$, let $R_t[k]$ be the log-likelihood of the observed data for state $k \in \{1, \ldots, K\}$. This encapsulation of the log-likelihood values enables us to focus on the chains without thinking about the rest of the model at this point. The complete data log-likelihood for $M_{1:T}$ can be written as

$$\sum_{t=1}^{T}\sum_{k=1}^{K} \mathbb{I}(M_t = k) R_t(k) + \sum_{k=1}^{K} \mathbb{I}(M_1 = k) \log \pi_k + \sum_{t=1}^{T}\sum_{j=1}^{K}\sum_{k=1}^{K} \mathbb{I}(M_{t-1} = j, M_t = k) \log A(j,k).$$

Later on we will encounter this expression integrated over the whole state space $M_{1:T}$ with a probability distribution $h(M_{1:T})$. It is important to be able to see that, if we want to

maximize with respect to $h(M_{1:T})$ the following quantity :

$$\int h(M_{1:T}) \left[ \sum_{t=1}^{T} \sum_{k=1}^{K} \mathbb{I}(M_t = k) R_t(k) + \text{const}_1 \right] + \int h(M_{1:T}) \left[ \sum_{k=1}^{K} \mathbb{I}(M_1 = k) \log \pi_k + \sum_{t=1}^{T} \sum_{j=1}^{K} \sum_{k=1}^{K} \mathbb{I}(M_{t-1} = j, M_t = k) \log A(j,k) \right] - \int h(M_{1:T}) \log h(M_{1:T}) + \text{const}_2, \tag{3.1}$$

that maximal value is achieved when $h(M_{1:T})$ follows the same distribution as the one induced on a first-order Markov chain by observations with log-likelihood values $R_t[k]$. This is true when $h(M_{1:T})$ is free to be any joint distribution on $(M_1, \ldots, M_T) \in \{1, \ldots, K\}^T$. This follows from the general principle that, for any given distribution $P$, the unique distribution $Q$ that minimizes $\mathrm{KL}(Q\|P)$ is $Q = P$ if we don't impose any constraints on $Q$.

The optimal $h(M_{1:T})$ can be obtained by running the *forwards-backwards* algorithm (see [Bis06] chap 13.2). It can be expressed analytically with the help of a vector of marginals $h(M_1)$ and a set of $T-1$ square matrices called the *two-slice marginals* that are denoted by $\xi_{t-1,t}(j,k) = h(M_{t-1} = j, M_t = k)$. We can sample from $h(M_{1:T})$ easily with those quantities and we get the smoothed marginals $h(M_t)$ as a bonus from the forwards-backwards algorithm.

The transition matrix $A$ can have a conjugate prior that takes the form of pseudocounts $A_{\mathrm{PC}}(j,k) \geq 0$ with a pdf proportional to

$$p(A|A_{\mathrm{PC}}(j,k)) \propto \prod_{j=1}^{K} \prod_{k=1}^{K} A(j,k)^{A_{\mathrm{PC}}(j,k)}.$$

This can be achieved more formally by taking the rows of $A$ to be distributed independently according to $\mathrm{Dirichlet}(A_{\mathrm{PC}}(j,1), \ldots, A_{\mathrm{PC}}(j,K))$ for rows $j = 1, \ldots, K$. The MLE with respect to $A$ of equation (3.1) is given by the forwards-backwards algorithm as an average of the two-slice marginals $\xi_{t-1,t}$.

We can find the MAP estimate of $A$ with pseudocounts simply by adding those to the two-slice marginals before renormalizing the rows (to get a proper transition matrix).

$$\hat{A}^{\mathrm{MAP}}(j,k) = \frac{\sum_{t=2}^{T} \xi_{t-1,t}(j,k) + A_{\mathrm{PC}}(j,k)}{\sum_{k'} \sum_{t=2}^{T} \xi_{t-1,t}(j,k') + A_{\mathrm{PC}}(j,k')} \tag{3.2}$$

These are the quantities that maximize (3.1) with respect to $A$ when all other variables are fixed. A similar reasoning shows that the MLE for the initial state priors $\pi_{1:K}$ is given by the normalized smoothed marginals $M_1(1{:}K)$.

## 3.2 Notation and assumptions for variational methods

The most important hidden quantities in our model are the patients assignments $G^{1:P}$, the chains $M_{1:T}^{1:G}$ and the hidden states $Z_{1:T}^{1:P}$ of the patients. We will use $f$ to refer to the original pdf from the *hmmmix* model and $h$ to refer to the pdf of the variational approximation to the hidden variables. Thus, we will be interested in maximizing a quantity of the form $\mathcal{L}(h) = \int h(W) \log f(Y, W|\theta, A, \pi) dW - \int h(W) \log h(W)$ where $W$ represent here the hidden variables, $Y$ are the observed variables and $\theta, A, \pi$ are the parameters of the function $f$.

The tractability of our model comes from restricting the possibilities of $h$ to a subfamily where the factorization $h(M_{1:T}^{1:G}, G^{1:P}) = h(M_{1:T}^{1:G})h(G^{1:P})$ holds. This allows us to maximize iteratively $h(M_{1:T}^{1:G})$ and $h(G^{1:P})$. Moreover, we will assume that we can further factorize the factors as $h(M_{1:T}^{1:G}) = \prod_g h(M_{1:T}^g)$ and $h(G^{1:P}) = \prod_p h(G^p)$, both of those assumptions being quite reasonable since patients are not supposed to be related and neither should the CNA profiles be. We will not assume that $h(M_{1:T}^g) = \prod_t h(M_t^g)$ but, for purposes other than for estimating the transition matrices $A^g$ and initial state priors $\pi^g$, the distributions $h(M_{1:T}^g)$ will propagate through the model in the same way as if we assumed that $h(M_{1:T}^g) = \prod_t h(M_t^g)$. We decided to marginalize out analytically the hidden variables $Z_{1:T}^{1:P}$ from the model. This is why we don't have a variational term $h(Z_{1:T}^{1:P})$, but $Z_{1:T}^{1:P}$ still play a part in the formulas.

The expected complete data log-likelihood to be maximized is

$$\mathcal{L}(h) = \int h(M_{1:T}^{1:G})h(G^{1:P}) \log f(Y_{1:T}^{1:P}, M_{1:T}^{1:G}, G^{1:P}, A^{1:G}, \pi^{1:G}, \theta) dG^{1:P} dM_{1:T}^{1:G} - \int h \log h. \tag{3.3}$$

where we use $\theta$ as shorthand for $(\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P})$ and where $f$ decomposes as

$$\begin{aligned} f(Y_{1:T}^{1:P}, M_{1:T}^{1:G}, G^{1:P}, A^{1:G}, \pi^{1:G}, \theta) =& f(Y_{1:T}^{1:P}|M_{1:T}^{1:G}, G^{1:P}, \theta) f(M_{1:T}^{1:G}|A^{1:G}, \pi^{1:G}) f(G^{1:P}) \\ & f(A^{1:G}|A_{\text{PC}}) f(\pi^{1:G}) f(\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}|m_{1:K}^{1:P}, \eta_{1:K}^{1:P}, \gamma_{1:K}^{1:P}, S_{1:K}^{1:P}). \end{aligned} \tag{3.4}$$

It will be convenient also to let $\mathcal{C}_g^p$ be the probability under $h(G^{1:P})$ that patient $p$ is assigned to cluster $g$. This is well-defined because we assume that the assignments to clusters between patients are independent. It also means that $\sum_{g=1}^G \mathcal{C}_g^p = 1$ for all patients $p$.

Some formulas contain entropy terms $-\int h(M_{1:T}^{1:G}) \log h(M_{1:T}^{1:G})$ that we will denote by $\mathbb{H}[h(M_{1:T}^{1:G})]$ to lighten the notation. Due to assumptions in our variational approximation, we have that

$$\mathbb{H}[h(M_{1:T}^{1:G})] = \sum_{g=1}^G \mathbb{H}[h(M_{1:T}^g)] \qquad \text{and} \qquad \mathbb{H}[h(G^{1:P})] = \sum_{p=1}^P \mathbb{H}[h(G^p)] \tag{3.5}$$

## 3.3 Training

### 3.3.1 Updates and initialization

We train our model by updating iteratively the variational approximations $h(M_{1:T}^{1:G})$, $h(G^{1:P})$ and the model parameters $A^{1:G}, \pi^{1:G}, \mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$. Due to the particular structure of the *hmmmix* model (as represented in figure 1.3), the parameters $A^{1:G}, \pi^{1:G}$ and the parameters $\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$ are independent when $h(M_{1:T}^{1:G})$ and $h(G^{1:P})$ are fixed. Traditional variational EM would have us update all the parameters $A^{1:G}, \pi^{1:G}, \mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$ simultaneously, but for our particular problem it is more natural to perform the updates to $h(M_{1:T}^{1:G})$ and $A^{1:G}, \pi^{1:G}$ together. We illustrate in figure 3.2 in what order the updates are performed.

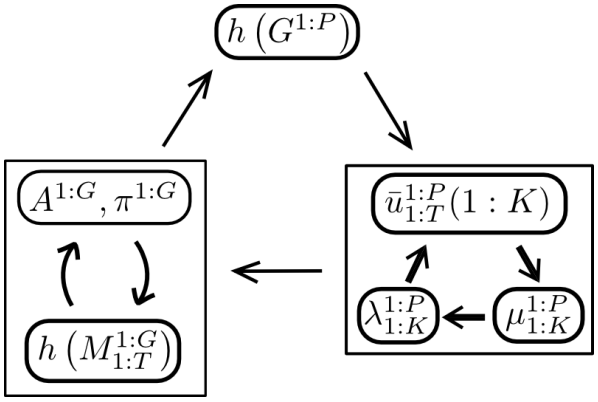



Figure 3.2: Our algorithm updates variational approximations and parameters iteratively. This picture shows an overview of the order in which the updates are performed. Within rectangular boxes, we have cycles of updates that should be performed until the quantities stabilize within a certain tolerance.

To initialize the algorithm, we start by assigning patients to clusters using k-medoids or by setting $h(G^{1:P})$ to uniform random values. Throughout the experiments that we did, results were always better with k-medoids than with a random initialization. The extent to which the k-medoids initializations are better depends on the particular case, though. We set the means $\mu_{1:K}^{1:P} = m_{1:K}^{1:P}$ and let the precisions $\lambda_{1:K}^{1:P}$ be relatively small to give the model some initial slack. We then start the updating scheme with the hidden chains and we iterate as shown in figure 3.2 until convergence.

If we start with good initialization values for the patients clustering (e.g. from k-medoids), it is important that we do not lose these values by updating them prematurely while we still have crude values of $h(M_{1:T}^{1:G})$ and $\theta$. In that particular case, we recommend skipping a few updates to $h(G^{1:P})$ while the rest of the model adjusts itself to the initial patient assignments. The arrows from figure 3.2 suggest one possible order in which we can do the updates. We

have tried a few variants but they are roughly equivalent.

### 3.3.2 Hidden chains

If we focus on maximizing (3.3) with respect to $h(M_{1:T}^{1:G})$, we can omit the terms not involving $M_{1:T}^{1:G}$ to get

$$\int h(M_{1:T}^{1:G})h(G^{1:P}) \log f(Y_{1:T}^{1:P}|M_{1:T}^{1:G}, G^{1:P}, \theta) dG^{1:P} dM_{1:T}^{1:G}$$
$$+ \int h(M_{1:T}^{1:G})h(G^{1:P}) \log f(M_{1:T}^{1:G}|A^{1:G}, \pi^{1:G}) dG^{1:P} dM_{1:T}^{1:G} + \mathbb{H}[h(M_{1:T}^{1:G})]. \tag{3.6}$$

We start by reformulating the first term in the previous expression (3.6) to separate as much as possible the contributions of the $G$ groups. We have that

$$\int h(M_{1:T}^{1:G})h(G^{1:P}) \log f(Y_{1:T}^{1:P}|M_{1:T}^{1:G}, G^{1:P}, \theta) dG^{1:P} dM_{1:T}^{1:G} \tag{3.7}$$

$$= \int h(M_{1:T}^{1:G}) \int h(G^{1:P}) \sum_t \log f(Y_t^{1:P}|M_t^{1:G}, G^{1:P}, \theta) dG^{1:P} dM_{1:T}^{1:G} \tag{3.8}$$

$$= \int h(M_{1:T}^{1:G}) \sum_t \int h(G^{1:P}) \sum_p \sum_g \mathbb{I}(G^p = g) \log f(Y_t^p|M_t^g, G^p = g, \theta) dG^{1:P} dM_{1:T}^{1:G} \tag{3.9}$$

$$= \int h(M_{1:T}^{1:G}) \sum_t \sum_g \sum_p \mathcal{C}_g^p \log f(Y_t^p|M_t^g, G^p = g, \theta) dG^{1:P} dM_{1:T}^{1:G} \tag{3.10}$$

$$= \int h(M_{1:T}^{1:G}) \sum_t \sum_g R_{t,g}[M_t^g] dM_{1:T}^{1:G} \tag{3.11}$$

where, in that last line, we defined

$$R_{t,g}[M_t^g] = \sum_p \mathcal{C}_g^p \log f(Y_t^p|M_t^g, G^p = g, \theta). \tag{3.12}$$

Continuing on (3.11), we get

$$\int h(M_{1:T}^{1:G}) \sum_t \sum_g R_{t,g}[M_t^g] dM_{1:T}^{1:G} = \sum_g \int h(M_{1:T}^g) \sum_t R_{t,g}[M_t^g] dM_{1:T}^g. \tag{3.13}$$

We now come back to the original expression (3.6) using (3.13) and the entropy term decomposition (3.5) to rewrite our formula of interest (3.6) as

$$\sum_g \left[ \int h(M_{1:T}^g) \left[ \sum_t R_{t,g}[M_t^g] + \log f(M_{1:T}^g|A^g, \pi^g) \right] dM_{1:T}^g - \int h(M_{1:T}^g) \log h(M_{1:T}^g) dM_{1:T}^g \right]. \tag{3.14}$$

Because the distributions $f\left(M^g_{1:T}|A^g, \pi^g\right)$ from the *hmmmix* model are Markov chains, we can see that (3.14) is essentially a sum of $G$ independent hidden Markov chains as found in section (3.1). It follows that, to maximize our variational lower bound (3.3) with respect to $h(M^{1:G}_{1:T})$, we should run forwards-backwards on the chains independently for $g = 1, \ldots, G$ with $R_{t,g}[M^g_t = 1, \ldots, K]$ as observation log-likelihoods. The difference from "regular" forwards-backwards is that we use these quantities $R_{t,g}[k]$ derived from the current variational approximations $h(G^{1:P})$.

When implementing the algorithm, we represent internally $h(M^g_{1:T})$ as a collection of two-slice marginals $\xi_{t-1,t}(1{:}K, 1{:}K)$ along with a vector for the smoothed distribution of the first state $h(M_1)$.

It can easily be shown that, to maximize the original variational lower bound (3.3) with respect to $(A^{1:G}, \pi^{1:G})$, we can use the two-slice marginals and $h(M_1)$ to find the new MLE values. We can add pseudocounts to $A^{1:G}$ by proceeding as explained in section 3.1.

We alternate between optimization with respect to $h(M^{1:G}_{1:T})$ and $(A^{1:G}, \pi^{1:G})$ because they depend on each other. We pick a tolerance threshold to determine when the values have stabilized to avoid excessive computations in the first steps in the algorithm. Once we are satisfied with the values MLE values of $(A^{1:G}, \pi^{1:G})$, we no longer need the two-slice marginals of $h(M^{1:G}_{1:T})$ and it is sufficient to keep only the smoothed marginals $\{h(M^g_t)|t = 1 \ldots T, g = 1 \ldots G\}$ as a K-by-T-by-G matrix.

### 3.3.3 Partial patients assignments

We now turn to optimizing our lower bound $\mathcal{L}(h)$ with respect to the distributions $h(G^{1:P}) = \prod_p h(G^p)$.

As the variational parameters $h(G^p)$ are essentially parameters for multinomial distributions (of one draw), we can represent them by values $\mathcal{C}^p_g$ corresponding to the weight of the partial assignment of patient $p$ in group $g$. Dropping all unrelated terms from (3.3), we get

$$\mathcal{L}(h) = \int h(M^{1:G}_{1:T}) h(G^{1:P}) \log f(Y^{1:P}_{1:T}|M^{1:G}_{1:T}, G^{1:P}, \theta) f(G^{1:P}) dM^{1:G}_{1:T} dG^{1:P} + \mathbb{H}\left(G^{1:P}\right) \quad (3.15)$$

$$= \int h(G^{1:P}) \left[ \int h(M^{1:G}_{1:T}) \log f\left(Y^{1:P}_{1:T}|M^{1:G}_{1:T}, G^{1:P}, \theta\right) dM^{1:G}_{1:T} + \log f(G^{1:P}) \right] dG^{1:P} + \mathbb{H}\left(G^{1:P}\right) \quad (3.16)$$

$$= \int h(G^{1:P}) \left[ \int h(M^{1:G}_{1:T}) \sum_p \sum_g \mathbb{I}(G^p = g) \log f\left(Y^p_{1:T}|M^g_{1:T}, G^p = g, \theta\right) \right] + \int h(G^{1:P}) \log f(G^{1:P}) + \mathbb{H}\left(G^{1:P}\right) \quad (3.17)$$

$$= \int h(G^{1:P}) \left[ \sum_p \sum_g \int h(M_{1:T}^{1:G}) \mathbb{I}(G^p = g) \log f\left(Y_{1:T}^p | M_{1:T}^g, G^p = g, \theta\right) \right] + \int h(G^{1:P}) \log f(G^{1:P}) + \mathbb{H}\left(G^{1:P}\right) \tag{3.18}$$

$$= \sum_p \int h(G^p) \sum_g \int h(M_{1:T}^g) \log f\left(Y_{1:T}^p | M_{1:T}^g, G^p = g, \theta\right) + \sum_p \int h(G^p) \log f(G^p) + \sum_p \mathbb{H}\left(G^p\right) \tag{3.19}$$

There are different ways to handle the $\int h(G^p) \log f(G^p)$ terms. The simplest of those is to use a uniform prior that allows us to ignore them. Another possibility is to take a Dirichlet$(\tau, \ldots, \tau)$ prior on $f(G^p)$. This prior is such that we have, for every $p$, that

$$\begin{aligned} \mathbb{H}\left(G^p\right) + \int h(G^p) \log f(G^p) &= -\sum_g \mathcal{C}_g^p \log \mathcal{C}_g^p - \sum_g \mathcal{C}_g^p (\tau - 1) \log \mathcal{C}_g^p + \text{cst}(\tau) \\ &= \tau \mathbb{H}\left(G^p\right) + \text{cst}(\tau) \end{aligned} \tag{3.20}$$

With this prior, we develop (3.19) further. It can be written as a sum of $P$ independent expressions, so the maximization problem is solved separately for the many $h(G^p)$. For every patient $p$, we get expressions of the form

$$\int h(G^p) \log(\sigma_g^p) - \int h(G^p) \log h(G^p) dG^p \tag{3.21}$$

where

$$\sigma_g^p = \exp\left(\tau^{-1} \int h(M_{1:T}^g) \log f\left(Y_{1:T}^p | M_{1:T}^g, G^p = g, \theta\right) dM_{1:T}^g\right). \tag{3.22}$$

The $\tau^{-1}$ in 3.22 comes from the $\tau$ coefficient in (3.20) by which we can divide the whole of expression (3.19). It follows that the desired distributions of $h(G^p)$ are softmaxima scaled by a factor of $\tau^{-1} > 0$. A stronger prior belief that cluster assignments should be uniform (corresponding to a larger $\tau$) translates into more uniform softmaxima terms due to lighter evidence (being divided by $\tau$).

The updated quantities for $h(G^{1:P})$ are given by

$$\mathcal{C}_g^p = \frac{\exp\left(\tau^{-1} \int h(M_{1:T}^g) \log f\left(Y_{1:T}^p | M_{1:T}^g, G^p = g, \theta\right) dM_{1:T}^g\right)}{\sum_{g'} \exp\left(\tau^{-1} \int h(M_{1:T}^{g'}) \log f\left(Y_{1:T}^p | M_{1:T}^{g'}, G^p = g', \theta\right) dM_{1:T}^{g'}\right)} \tag{3.23}$$

$$= \frac{\exp\left(\tau^{-1} \sum_{t=1}^T \sum_{k=1}^K h(M_t^g = k) \log f\left(Y_{1:T}^p | M_{1:T}^g, G^p = g, \theta\right)\right)}{\sum_{g'} \exp\left(\tau^{-1} \sum_{t=1}^T \sum_{k=1}^K h(M_t^{g'} = k) \log f\left(Y_{1:T}^p | M_t^{g'}, G^p = g', \theta\right)\right)} \tag{3.24}$$

The uniform prior on $f(G^p)$ corresponds to picking $\tau = 1$. By adding a prior we are changing the *hmmmix* model slightly. We discuss this further in section 4.9. This $\tau$ parameter should be seen as a tool to add flexibility to the model. It can be used to tune the convergence speed of the whole algorithm. Note also that we only need the smoothed distributions $h(M_t^g)$ to update the $\mathcal{C}_g^p$ and we don't use the two-slice marginals. This has to do with the fact that the hidden states $Z_{1:T}^{1:P}$ are not connected as Markov chains.

### 3.3.4 Observation parameters

When comes to the time to maximize $\mathcal{L}(h)$ with respect to $\theta = \{\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}\}$, we use the priors from section 3.3.1 of [Arc05]. We decided to handle the hidden variables $Z_{1:T}^{1:P}$ in a slightly different way than [Arc05], though, by integrating over $Z_t^p$ when evaluating the quantity $R_t^p[k]$. As [Arc05] covers a variety of models with different flavors, we define everything explicitly in here. The technique used to get the parameter update formulas is not part of our contributions, but it is still necessary for us to give out the details of the work for our particular case.

We want to maximize the component of the lower bound of $\mathcal{L}(h)$ that depends on $\theta$. This quantity is

$$L(\theta) = \int h(M_{1:T}^{1:G}) h(G^{1:P}) \log f(Y_{1:T}^{1:P} | M_{1:T}^{1:G}, G^{1:P}, \theta) f(\theta)$$

$$= \int h(M_{1:T}^{1:G}) h(G^{1:P}) \left[ \log \int f(Y_{1:T}^{1:P} | Z_{1:T}^{1:P}, \theta) f(Z_{1:T}^{1:P} | M_{1:T}^{1:G}, G^{1:P}) dZ_{1:T}^{1:P} + \log f(\theta) \right] \tag{3.25}$$

$$\geq \int \left[ h(M_{1:T}^{1:G}) h(G^{1:P}) f(Z_{1:T}^{1:P} | M_{1:T}^{1:G}, G^{1:P}) dM_{1:T}^{1:G} dG^{1:P} \right] \left[ \log f(Y_{1:T}^{1:P} | Z_{1:T}^{1:P}, \theta) f(\theta) \right] dZ_{1:T}^{1:P} \tag{3.26}$$

Now we will maximize a lower bound on $L(\theta)$ instead of maximizing that quantity itself. A legitimate alternative would have been to start out with a variational parameter $h(Z_{1:T}^{1:P})$ to go along $h(M_{1:T}^{1:G}) h(G^{1:P})$, but we opted against this approach. In any case, the effects of the relaxation from (3.25) to (3.26) are limited to the optimization of the parameter $\theta$. This is the main reason why our final formulas are slightly different from those of [Arc05].

The derivations for the update formulas of the parameters $\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$ are not too insightful. We provide them in appendix A and give the essential formulas here. These formulas involve temporary variables $\bar{u}_t^p(k)$ that can be discarded once we settle on a choice of $\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$. We select a tolerance threshold and we iteratively update the quantities according to the following formulas until they stabilize. From our experience, this generally took at most a dozen iterations.

$$\bar{u}_t^p(k) = \frac{\nu_k^p - 1}{\lambda_k^p \left( Y_t^p - \mu_k^p \right)^2 + \nu_k^p} \tag{3.27}$$

$$\mu_k^p = \frac{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k) Y_t^p + \eta_k^p m_k^p}{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k) + \eta_k^p} \tag{3.28}$$

$$(\lambda_k^p)^{-1} = \frac{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k)\left(Y_t^p - \mu_k^p\right)^2 + \eta_k^p\left(\mu_k^p - m_k^p\right)^2 + 2S_k^p}{\sum_t \rho_t^p(k) + 2\gamma_k^p - 1} \tag{3.29}$$

## 3.4 Projecting to hard assignments

With *hmmmix-soft* we are maximizing the variational lower bound $\mathcal{L}(h)$ (equation (3.3)) with respect to the parameters $A^{1:G}, \pi^{1:G}, \theta$. We get variational approximations $h(M_{1:T}^{1:G}), h(G^{1:P})$ as well as MAP estimates for $A^{1:G}, \pi^{1:G}, \theta$. With *hmmmix-hard* we get imputed values for the chains $M_{1:T}^{1:G}$ and patient assignments $G^{1:P}$. There are a few simple methods that can be used to get hard values from the variational approximations, but not all of them are equally good. We don't necessarily want to get those values, so the method explained here is not really the "final step" of *hmmmix-soft*, but rather just something that could be done if desired.

The natural way to get hard values instead of a posterior over chains is to use the Viterbi algorithm like in *hmmmix-hard* instead of the forwards-backwards used by *hmmmix-soft*. The quantities $R_{t,g}[M_t^g = k]$ (equation 3.12)) can be used by the Viterbi algorithm in the same way that they are used by forwards-backwards as log-likelihoods of observations. With the values of $M_{1:T}^{1:G}$ obtained, we then use equation (3.24) to compute the quantities $\mathcal{C}_g^p$. We assign each patient $p$ to the cluster $g = \text{argmax}_g \mathcal{C}_g^p$. The code from the regular *hmmmix-soft* updates can be reused with the hard $M_{1:T}^{1:G}$ values if we encode them as 1-of-K binary vectors that essentially represent a point-mass distribution taking the role of the usual $h(M_{1:T}^{1:G})$ term.

The values $M_{1:T}^{1:G}$,$G^{1:P}$ obtained in this fashion are not guaranteed to be good maximizers of

$$\sum_M \sum_G \max_{A,\pi,\theta} p(Y, M, G, \theta, A, \pi), \tag{3.30}$$

and that quantity could probably be improved by a few rounds of ICM. If we launched *hmmmix-hard* again at this point, the whole *hmmmix-soft* procedure would serve as some kind of initialization method for *hmmmix-hard*. We compare in section 4.1 the values of (3.30) achieved by *hmmmix-hard* to those obtained by this method after running *hmmmix-soft*. Experimentally, those obtained with *hmmmix-hard* are higher than those from *hmmmix-soft*, as one might expect.

## 3.5 Flow of approximations

On top of the original assumptions about the factorization of our variational distribution $h(M_{1:T}^{1:G}, G^{1:P}) = h(M_{1:T}^{1:G})h(G^{1:P}) = \prod_g \prod_p h(M_{1:T}^g)h(G^p)$, we used a few heuristics to learn

the parameters $\theta = \mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$. An overview of all the approximations used is shown in figure 3.3.

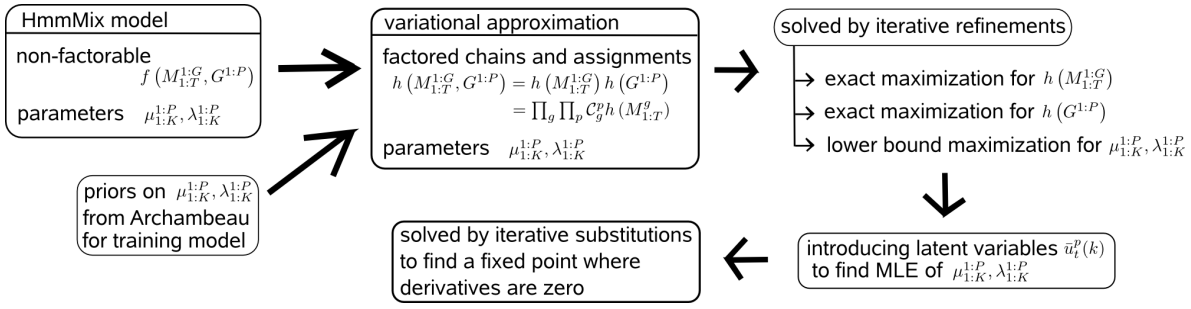



Figure 3.3: The relations between the different parts of the algorithm in terms of approximations.

We never showed that the values of $\bar{u}_t^p(k), \mu_k^p, \lambda_k^p$ that we get by iterative substitutions in section 3.3.4 were a maximum and not an minimum. However, we can always compute the value of either (3.25) or (3.26) to determine if we want to accept a proposed update or to reject it.

In our implementation, we compute the values of $f(Y_t^p | Z_t^p = k, \theta)$ and store them in a matrix of size (K,T,P). By storing $\rho_t^p(k)$ also in a matrix of size (K,T,P), it relatively straightforward to evaluated the two quantities used as safeguards against bad parameter updates.

The original log-likelihood lowerbound (3.25) is given by

```
sum(sum(log(sum(rho_KTP .* YgivenZ_KTP,1))))
```
(3.31)

and the lower bound (3.26) on (3.25) is given by

```
sum(sum(sum(rho_KTP .* log(YgivenZ_KTP)))).
```
(3.32)

If we trust the parameter updates, we can speed up the algorithm greatly by not computing the log-likelihood. As explained in section 4.7 and confirmed by our runtime profiling, the performance bottleneck comes from the evaluations of $p(Y_t^p | Z_t^p = k, \theta)$.

# Chapter 4

# Results and implementation

In this chapter we present three experiments done to evaluate the performance of *hmmmix-soft*. We then analyze the complexity of the algorithm and discuss the potential hardware issues that arise when the size of datasets exceeds the amount of available memory. We point out how the algorithm could be parallelized and we come back to the topic of the free energy scaling constant first introduced in section 3.3.3.

## 4.1 A minimum description length experiment

We argued in section 2.5 that BIC was not a good way to find the number of clusters as it fails to take consideration the fact that the chains that model CNAs have rare state transitions. The minimum description length (MDL) criterion was suggested as alternative. We present here an experiment done with synthetic data sampled from the *hmmmix* model itself. Like in section 2.5, we are using using $G = 10$ chains, $P = 100$ patients and sequences of length $T = 10\ 000$. We supply *hmmmix-hard* and *hmmmix-soft* with the true hyperparameters with which the data was generated. By the MDL principle, we should seek to minimize the quantity

$$-\log_2 f(Y_{1:T}^{1:P}|M_{1:T}^{1:G}, G^{1:P}, \mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}) + L(M_{1:T}^{1:G}, A^{1:G}, \pi^{1:G}, G^{1:P}, \mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P})$$

where $L(M, A, \pi, G, \mu, \lambda)$ is the code length necessary to encode the parameters given the known fixed hyperparameters. As pointed out by [HY01], we don't need to describe the coding scheme explicitly to use the MDL principle. The cost of sending an event $x$ with probability $P(x)$ is $-\log_2 P(x)$. This means that we want to minimize

$$-\log f(Y_{1:T}^{1:P}|M_{1:T}^{1:G}, G^{1:P}, \mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}) - \log f(M_{1:T}^{1:G}, A^{1:G}, \pi^{1:G}, G^{1:P}, \mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}|\text{ hyperparams}).$$

No prior is defined in the original *hmmmix* model for the patient assignments $G^{1:P}$ so we exclude $\log f(G^{1:P})$ from the code length. We also exclude the term $\log f(A, \pi|\text{pseudocounts})$ for practical reasons (after verifying that its contribution was negligible).

Thus, we compare *hmmmix-hard* and *hmmmix-soft* on their ability to maximize the quantity

$$\log f(Y_{1:T}^{1:P}|M_{1:T}^{1:G}, G^{1:P}, \theta) + \log f(M_{1:T}^{1:G}|A^{1:G}, \pi^{1:G})$$

$$+\log f(\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P} | m_{1:K}^{1:P}, \eta_{1:K}^{1:P}, \gamma_{1:K}^{1:P}, S_{1:K}^{1:P}). \tag{4.1}$$

We use the method described in section 3.4 to get hard values for the chains $M_{1:T}^{1:G}$ and patients assignments $G^{1:P}$ from *hmmmix-soft* instead of variational distributions $h(M_{1:T}^{1:G})$ and $h(G^{1:P})$. We are not only interested in comparing the two methods here, but we are also interested in seeing if the MDL values obtained by each method can be used to find the true number of clusters used to generate the data.

For both *hmmmix-hard* and *hmmmix-soft*, we use a random initialization of the patients (and hidden states $Z_{1:T}^{1:P}$, in the case of *hmmmix-hard*) as well as a k-medoids initialization. The random initializations are shown as solid curves in figure 4.1 while the k-medoids variants are shown as dotted curves. We plot the values of the quantity (4.1) obtained by each method.

The values obtained by *hmmmix-hard* are significantly higher than those obtained by *hmmmix-soft*. In all cases, we get an improvement by using k-medoids to initialize the patient assignments. In the case of figure 4.1(a), it's quite clear that we can identify the true number of clusters with the results of *hmmmix-hard* or *hmmmix-soft* with k-medoids. The results are more ambiguous in the case of $G = 10$ and they would very unreliable in all cases if we didn't average them over 30 trials to get smooth curves.

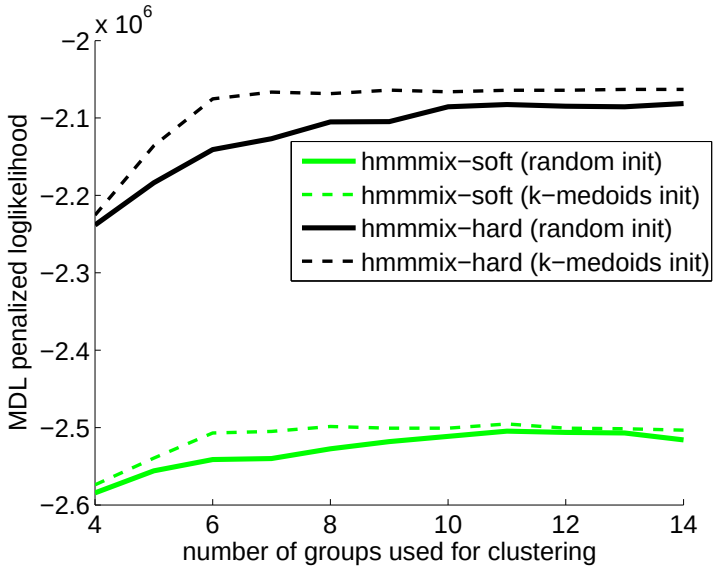
## 4.2 Gap statistic

The *gap statistic* introduced by [TWH01] is another way to estimate the true number of clusters. One of its special features is its ability to detect when there is no need for more than one cluster, but this is not the reason why we are using it. The results that we got with the gap statistic are not convincing enough to justify a detailed exposition of the theory. Instead, we present in appendix B the method used in our particular case and we refer the reader to the original [TWH01] or to [Koe08] for a good summary.

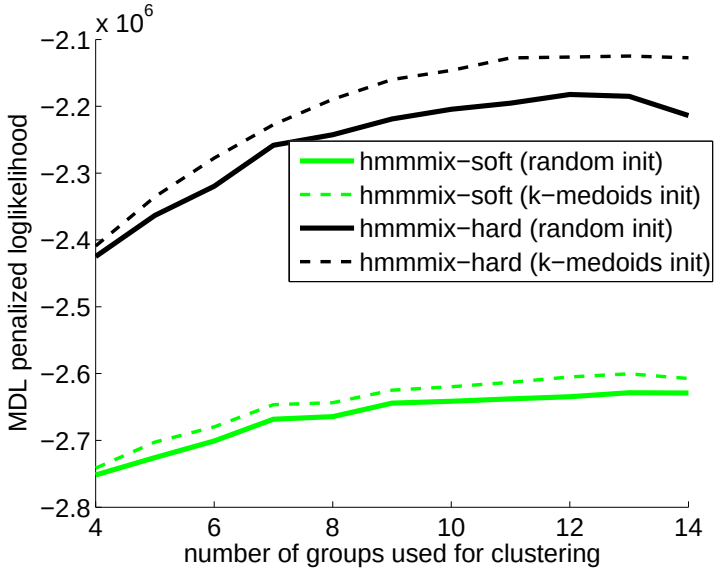
At the core of the gap statistic is the notion of *within cluster spread*

$$\text{spread}(E) = \frac{1}{2|E|} \sum_{x,y \in E} \text{dist}(x, y). \tag{4.2}$$

For a given clustering with $C \in \{1, \ldots, C_{\text{MAX}}\}$ clusters, we define the spread of the final set of clusters as the sum of the spreads of the individual clusters. We then compare the results for all the values of $C$ studied. We should expect spreads values to decrease fast as we increase $C$ up to the point when $C = G$, where G is the real number of clusters. After $C > G$, the decrease should be less significant because there are no more clusters with far away points to break up.




(a) true number of clusters $G = 6$




(b) true number of clusters $G = 10$

Figure 4.1: We show here the results for the log-likelihood of the data penalized by the MDL term (see equation 4.1). We use $G = 6, 10$ hidden chains respectively to generate the data and distributed $P = 100$ patients among those $G$ clusters. The length of the sequences is $T = 10\ 000$. In green we have the results from *hmmmix-soft* and in black we have those from *hmmmix-hard*. The solid lines represent the values obtained with a random initialization while the dotted curves represent those obtained by k-medoids. These results come from averaging 30 trials. Ideally, we would want the curves to grow monotonously with the number of groups used (clusters) up until the true value of $G$. At that point, we would like the curves should be flat, indicating that adding further clusters will not help us to model the data.

Due to the very prohibitive cost of this method, we tested it only on *hmmmix-soft* with random initializations, using the datasets generated in the same way as in 4.1 but with $T = 1000$ instead of $T = 10\ 000$. We generated datasets with $G = 10$ clusters each and we explored values of $C = 6, \ldots, 14$. Out of 28 trials, the number of clusters reported by the gap statistic was

- 6 : 6 occurrences
- 9 : 21 occurrences
- 12 : 1 occurrence.

Starting *hmmmix-soft* with $C$ clusters doesn't guarantee that we end up with $C$ clusters being used in the final patient assignments, though. Any convincing experiment to validate the gap statistic would have to be done on a larger scale and the cost is just too great. The 9 clusters estimate here is close to the target value 10, but if the method was valid we would expect wrong guesses to be more uniform than this. The fact that we obtained 6 occurrences is suspicious because this was the smallest value of $C$ explored.

## 4.3 Benchmark with synthetic data not from *hmmmix*

In [Sha08] *hmmmix-hard* is compared to the current state-of-the-art methods using a benchmark test with artificial data. This data is not generated from the *hmmmix* model itself (like in sections 4.1 and 4.2), but comes from real aCGH data into which segments of the data has been modified to look like CNAs. We reproduced the experiment from [Sha08] with *hmmmix-hard* and ran *hmmmix-soft* on the same data to see how our method would compare.

As mentioned in section 2.4, one of the limitations of *hmmmix-hard* is its reliance on good cluster initialization for the patients. For this reason, we have spent most of our efforts trying to beat *hmmmix-hard* on the synthetic data benchmark without having to resort to a k-medoids initialization that knows the real number of clusters a priori. This turns out to be really hard with the data used except in cases where k-medoids fails to produce anything useful.

The measure of success chosen for this experiment is the Jaccard coefficient (see appendix B). It requires knowledge of the true cluster assignments to be computed and this is why it can only be used on artificial data. The Jaccard coefficient is always in $[0, 1]$, higher being better, and a value of 1 can be achieved if and only if we find cluster assignments equivalent to the

true clustering. The order or labels of the clusters does not affect the result, but using the wrong number of clusters gives a smaller Jaccard coefficient.

We used the same data as [Sha08] and explored the combinations of values from $G = 3, 5, 10$ and $L = 25, 50, 75$ where $L$ corresponds to the length of the mutations introduced. As *hmmmix-hard* performs quite well in most of the cases with $G = 3, 5$, we decided to focus on $G = 10$. Our results can be summarized by table 4.3.

Table 4.1: Comparing *hmmmix-soft* to *hmmmix-hard* using Jaccard coefficient. The values are averaged over 50 generated datasets and the standard deviations are included to give an idea of how spread results are.

| | L=25 | L=50 | L=75 |
|---|---|---|---|
| k-medoids initialization | 0.77 ± 0.11 | 0.33 ± 0.01 | 0.15 ± 0.03 |
| *hmmmix-hard* | 0.90 ± 0.13 | 0.61 ± 0.17 | 0.17 ± 0.08 |
| *hmmmix-soft* | 0.93 ± 0.11 | 0.58 ± 0.15 | 0.35 ± 0.09 |

We have in the first row the value of the Jaccard coefficient with the initial clustering obtained from k-medoids. We can see that it's already large in the case of $L = 25$ and that it's almost worthless in the case of $L = 75$. With a completely random initialization of 10 clusters, the Jaccard coefficient is a little bit above 0.10. In the case of $L = 75$, we can see that *hmmmix-soft* can improve on the initial clustering while *hmmmix-hard* cannot. These results are averaged on 50 tries so the difference is statistically significant, but one could argue that 0.35 is still relatively bad.

One of the difficulties that we encountered here was that the Jaccard coefficient does not care about the fact that a soft clustering might be desirable or even very informative. Instead of using the method from section 3.4 to get hard patient assignments, we merged duplicate clusters obtained by *hmmmix-soft* and then assigned patients to their most likely cluster.

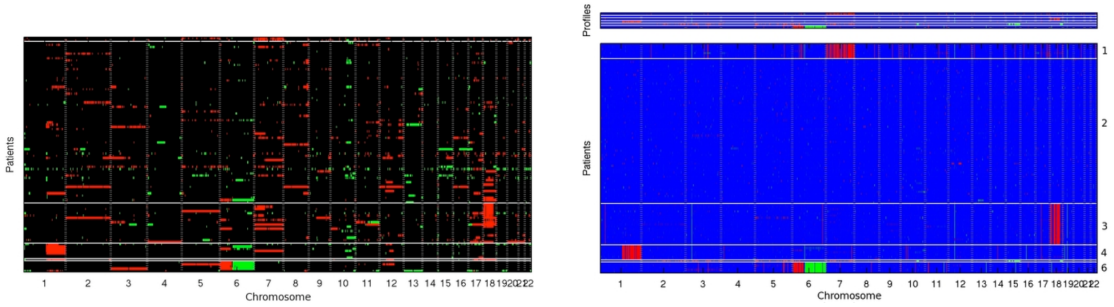
Some other scoring criterion could be used to deal with partial patient assignments, but the purpose of this experiment was to show that we can perform well in the benchmark test from [Sha08].

## 4.4 Comparison with hmmmix-hard on FL data

It is difficult to compare quantitatively the performance of *hmmmix-soft* and *hmmmix-hard* on real data because often there are no target values available as ground truth. Not only are we missing information, but the parameters from the models don't necessarily correspond to any real feature. One way to proceed is to predict biological phenomena from the models and

verify with experts if your predictions are correct (or useful). The is one of ways in which [Sha08] justified the *hmmmix* model.

With our *hmmmix-soft*, we would like to be able to achieve results similar to those of *hmmmix-hard* with real data in terms of imputing hidden values. Our measure of success is how close we can get to the results presented in section 5.4.1 of [Sha08] using the same dataset. The dataset used for this experiment is the follicular lymphoma dataset from [CSS+09] that has $P = 106$ patients, with sequences of length $T = 24\ 881$ spanning over 22 chromosomes. To determine the number of clusters, [Sha08] used weighted k-medoids on imputed values of $Z_{1:T}^{1:P}$ to maximize the silhouette coefficient (they got $G = 6$ clusters). Their original results are included here in figure 4.2.




(a) initial imputed values of $Z_{1:T}^{1:P}$

(b) final values of $Z_{1:T}^{1:P}$ after *hmmmix-hard*

Figure 4.2: The original results from [Sha08] for *hmmmix-hard* on the follicular lymphoma dataset. The patients are reordered for easier visualization. The green segments correspond to a higher copy number while red segments correspond to deletions.

The results that we get from the *hmmmix-soft* algorithm naturally depend on the choices that we make for the free parameters in the model. In particular, choosing $\alpha_L, \alpha_N, \alpha_G$ to be higher forces the hidden chains to "explain" the data better, but it undermines the smoothing effect that the transition matrices are supposed to have in the chains. We experimented with a few different sets of hyperparameters and in figure 4.3 we show one of the closest matches (visually) to the results in [Sha08].

We used a special way to cluster the patients that we describe in section 4.5. We used partial cluster assignments and most of the weight was distributed among 4 main clusters. Although the results shown in figure 4.3 are somewhat similar to those of 4.2(a), we are missing an important CNA segment in chromosome 1.

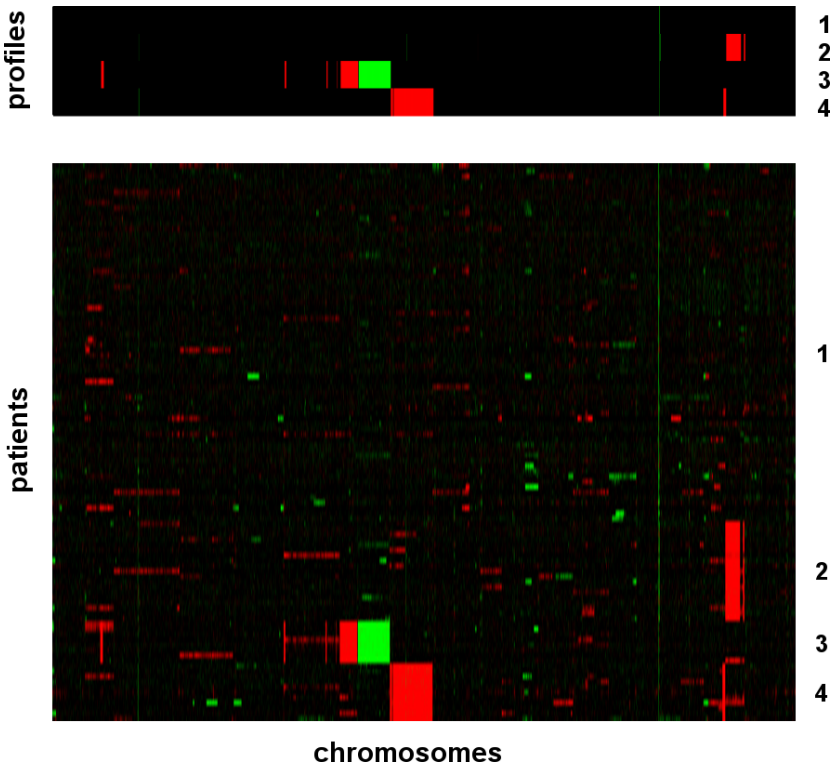



Figure 4.3: Final results for the follicular lymphoma dataset after running *hmmmix-soft* and deriving hard values for $M_{1:T}^{1:G}, G^{1:P}$. The profiles identified are displayed on top and the best values of $Z_{1:T}^{1:P}$ are shown below for every patient. The patients have been reordered so that patients from the same cluster are listed consecutively.

## 4.5 Alternative distance-based clustering

One of the goals of this thesis is to see what can be done with the *hmmmix* model once we can use partial patients assignments. For that reason, to get the results from figure 4.3 we didn't use the k-medoids initialization but we decided instead to see if we could achieve similar results using the classic mixture of Gaussians trained with EM to provide an initial clustering. The usual k-medoids used in other experiments in done with values of $Z_{1:T}^{1:P}$ imputed by a method from [Sha08]. Here we used the observations $Y_{1:T}^{1:P}$ instead to perform the clustering.

The idea was to get a rough partial clustering that distributed the weights of the patients to multiple clusters in a useful way. We didn't want patients to be assigned to a single cluster but we didn't want to have a trivial uniform cluster assignment either. The high dimensionality $T$ of the data tends to make EM assign patients to single clusters. To mitigate the effects of the large $T$, we introduced a scaling factor to the entropy term similar to the $\tau$ from section 3.3.3 (also discussed in section 4.9).

To sketch quickly the details of this modification the EM, we refer the reader to the description of the algorithm in chapter 9 of [Bis06]. With $p(\mathbf{x}_n|\mu_k, \Sigma_k)$ being the likelihood that sample $n$ belongs to Gaussian component $k$, instead of updating the cluster assignments in the E-step according to $\gamma(\mathbf{z}_{nk}) \propto p(\mathbf{x}_n|\mu_k, \Sigma_k)$, we update them with $\gamma(\mathbf{z}_{nk}) \propto p(\mathbf{x}_n|\mu_k, \Sigma_k)^{\tau^{-1}}$. Large values of $\tau$ spread the cluster responsibilities $\gamma(\mathbf{z}_{nk})$ more evenly.

It's hard to justify any value for the scaling $\tau$ factor a priori, but we investigated values from different orders of magnitude and computed the entropy of the resulting EM clustering distributions (merging duplicate clusters). With the follicular lymphoma experiment of section 4.4 there was a clear and identifiable threshold value at which the patients went from having all their weight assigned to single clusters to being spread around more evenly (but not in a trivial uniform way). That threshold value stayed roughly the same when restarting EM multiple times. The particular value is a property of the data and what worked for the FL dataset might not work elsewhere (the threshold value should change, for example).

The cost of running EM on a large dataset is high, but an iteration of EM is cheaper than an iteration of *hmmmix-soft* itself. If we refrain from doing multiple restarts of EM, it will not be the performance bottleneck. If it's still somewhat prohibitive, we can perform the initial clustering with EM using only a subsequence of the indices $t = 1 : T$ and we certainly don't need to iterate until the EM stabilizes within a very small tolerance bound.

It would have been possible also to use a mixture of student-t distributions instead of gaussians, but we haven't explored that alternative. It would be a little more complicated in that case to learn the parameters for the distributions with EM.

## 4.6 Multiple chromosomes

The description of the *hmmmix* model was simplified a bit here to deal with only one chain for every group/cluster/cohort. When dealing with real data, we have observations associated to multiple chromosomes and we don't necessarily want to treat them all as one long chain. As mentioned in [Sha08], there are good reasons to believe that the best values for the transition matrices vary from chromosome to chromosome within any given group.

Multiple chromosomes can be handled very easily by running the forwards-backwards algorithm separately for each chromosome, keeping track of the parameters MLE (a transition matrix and a vector of beliefs about the initial states) for every chromosome and every group.

The complexity analysis from section 4.7 only has terms involving the chain lengths $T$ linearly so we don't get penalized in terms of computational costs by doing to the forwards-backwards on chromosomes separately. In fact, if we were dealing with an enormous dataset, we could analyze the different chromosomes in parallel when solving for the multi-chromosome equivalent of $h(M_{1:T}^{1:G})$. The update formulas for $h(G^{1:P})$ and $\theta$ are not even aware of the chain structure of the hidden Markov chains $M_{1:T}^{g}$. They stay the same when dealing with multiple chromosomes. We can store the data for all the chromosomes one after the other as long as we keep track of the cuts when doing forwards-backwards on the sections.

## 4.7 Complexity analysis

When analyzing the complexity of each of the components of the algorithm, we have to remember that we are not taking into consideration the number of iterations that they will take to stabilize (within a certain tolerance). Experimentally, we observed that it usually takes a bit less than 10 iterations (update $M$, update $\theta$, update $G$, update $\theta$, ...) to have patients assignments stabilize. By using a large value of $\tau$ for the assignment prior (from section 3.3.3), this slowed down the algorithm so that convergence took around 50 such iterations, but we got quantitatively better results (this is just a heuristic rule, though).

With this in mind, it is still relevant to analyze the complexity of one such iteration to get a good picture of what is involved. The main loop of the algorithm goes over the following important steps, which correspond to one cycle in the circuit from figure 3.2.

- compute the values of $R_{t,g}[k]$ with equation (3.12) for all chains : $\mathcal{O}\left(GPTK^2\right)$
- do the following until stabilization

  - update the chains $h(M^g_{1:T})$ using the values of $R_{t,g}[k]$, running forwards-backwards : $\mathcal{O}\left(GTK^2\right)$
  - update the parameters $A^g, \pi^g$ for all chains : $\mathcal{O}\left(GTK^2\right)$
- update the patient assignments $h(G^p)$ using $R_{t,g}[k]$ and the current chains $h(M^g_{1:T})$ with equation (3.24): $\mathcal{O}\left(GPTK\right)$
- compute $\rho^p_t\left(k\right)$ with equation (A.6) : $\mathcal{O}\left(GPTK^2\right)$
- update the parameters $\theta$ by updating the following values with formulas (3.27), (3.28), and (3.29) until stabilization :
  - $\bar{u}^p_t(k)$ : $\mathcal{O}\left(PTK\right)$
  - $\mu^{1:P}_{1:K}$ : $\mathcal{O}\left(PTK\right)$
  - $\lambda^{1:P}_{1:K}$ : $\mathcal{O}\left(PTK\right)$

We are usually using $K = 3$ and $G \approx 10$ so the most important terms are $P$ and $T$. The whole *hmmmix-soft* algorithm run as $\mathcal{O}\left(GPTK^2\right)$ times the number of iterations that we do externally (usually between 10 and 50). In our experiments, the update steps that involve some indefinite number of iterations internally (to stabilize within a certain tolerance) have never taken more than a dozen iterations.

It should also be noted that we need to take logarithms to compute $R_{t,g}[k]$, but all the other operations involve only basic arithmetic. The runtime profiling confirms that the performance bottleneck comes from the evaluations of $p(Y^p_t|Z^p_t = k, \theta)$ and $R_{t,g}[k]$. With the exception of the possible random initialization, the algorithm is deterministic so there are no hidden costs from random number generation.

It is interesting to note that, due to the hard patients assignments in *hmmmix-hard*, that algorithm can do certain things in $\mathcal{O}\left(PTK^2\right)$ that would take *hmmmix-soft* $\mathcal{O}\left(GPTK^2\right)$ operations to do. The overall cost of *hmmmix-hard* is still $\mathcal{O}\left(GPTK^2\right)$ because patient assignments are done by checking how well every patient fits every cluster.

## 4.8 Scalability

As explained in section 4.7, the *hmmmix-soft* algorithm has an asymptotic computational cost of $\mathcal{O}(GPTK^2)$. Our basic implementation has memory requirements proportional to $\mathcal{O}(GTK^2 + PTK)$ simply because of the size of the quantities found in the formulas of section 3.3. The algorithm scales linearly in terms of increasing the number of groups, patients or

length of the sequences. It would be hard to expect better, but the memory requirements can still be a problem when dealing with very large datasets.

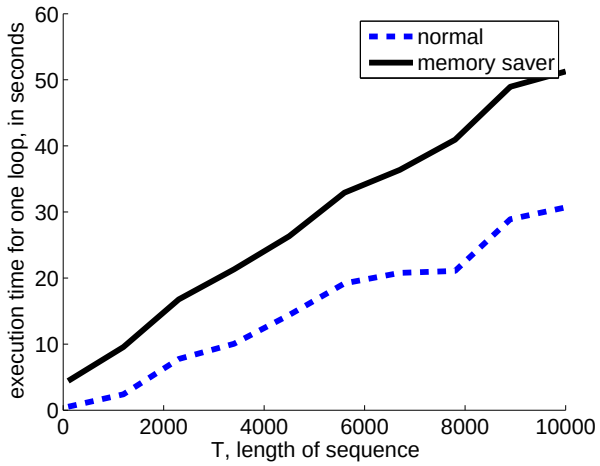
### 4.8.1 Hardware issues

We ran *hmmmix-soft* on a dual-core Mac Mini with 2 gigabytes of RAM for various sequence lengths ranging from $T = 200$ to $T = 1,000,000$. We used artificial data with $G = 10$ groups and $P = 100$ patients. In figure 4.4(a) and 4.4(b) we show with the dotted blue curve the execution time for one iteration of the main loop. We can see in the second plot 4.4(b) that the line breaks at $T = 80,000$ when Matlab runs out of memory.

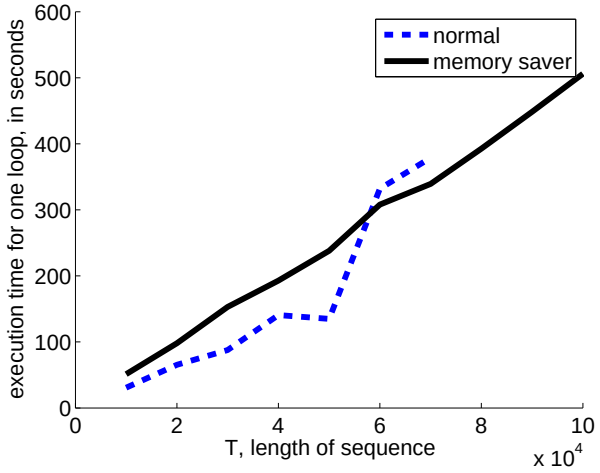
We ran more tests on a PC with a dual-core AMD 64-bit processor with 4 gigabytes of RAM and 12 gigabytes of swap memory. The goal was to see if the swap could be used to allow the algorithm to handle larger datasets. When running *hmmmix-soft* on a large dataset with $G = 10, P = 100, T = 250\ 000$, the available memory wasn't enough to satisfy the algorithm and there were a lot of page misses. In fact, the processor was reported to be only running at around 20% efficiency because most of the execution time was spent transferring variables to and from the swap (the swap being on a regular hard drive). One time, Matlab issued an out of memory error while processing a dataset with $T = 100\ 000$ even though the swap could easily have handled this. This is clearly unacceptable and the swap memory solution is a temporary fix that will break if we multiply the number of patients or do anything else to increase the datasets by an order of magnitude.

After accepting the fact that, by using the swap memory, we were going to do reads and writes to the hard drive during the execution, we wrote a version of the algorithm that keeps nothing in memory except what it needs at a given moment. After splitting the initial dataset of size $(P, T)$ into $P$ sequences stored on the hard drive, the total memory usage for the algorithm falls to $\mathcal{O}(TK^2)$. We decided to draw the line there, but we could have gone further and stored the chromosomes separately.

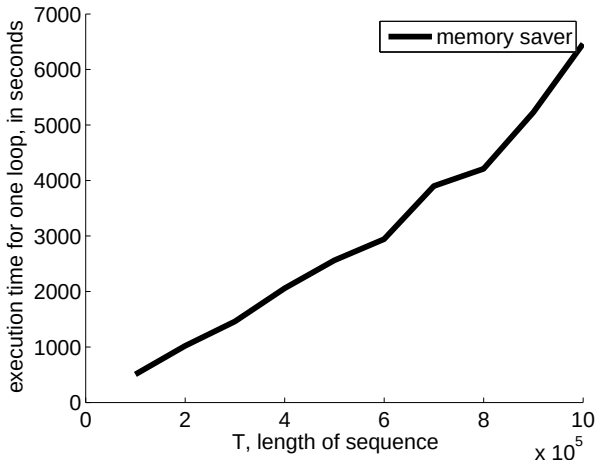
To compare the two approaches, we ran the low-memory implementation on the same Mac Mini. The results are again show in figure 4.4 with the solid black curve. We can see in 4.4(a) how the disk I/O operations slow down the execution significantly for short sequences. It becomes more acceptable in 4.4(b) in the context of larger datasets, and it is absolutely necessary in 4.4(c) with huge datasets. The low-memory implementation also reports that the Mac Mini's two cores are both used to their maximum capacity, unlike the 20% efficiency that we got earlier on by delegating the task of memory management to the kernel. The issue might be more complicated and there might be a way to help Matlab manage its memory better. In any case, we recommend the low-memory implementation for larger datasets.




(a) short sequences




(b) T=10,000 to 100,000




(c) T=100,000 to 1,000,000

Figure 4.4: The execution time for the basic implementation of *hmmmix-soft* (dotted blue curve) compared to a low-memory implementation (solid black curve).

With figure 4.4 we can also confirm our asymptotic analysis claiming that the algorithm scales linearly with $T$. Discussions about hard drive seek times, contiguous memory on the disk, flash hard drives and network file sharing access speeds fall outside of our domain of expertise.

### 4.8.2 Parallelization

The *hmmmix-soft* algorithm can be parallelized by at least a factor of $G$. The updates to the $P$ patients assignments $\mathcal{C}_g^p$ are independent and so are the updates to the $G$ chains. In theory, the only part of the algorithm that doesn't lend itself naturally to parallelization is the forwards-backwards algorithm that runs on sequences of length $T$. Although it seems sequential in nature, it's still possible to do something to parallelize it when the number of states $K$ is small compared to the number of processors by adapting the approach from [BMR97]. We don't think it's worth the trouble, though, considering that the bottleneck in the algorithm is elsewhere and that the sequences are already separated into multiple chromosomes that can be processed in parallel. From a more practical standpoint, the main obstacle to parallelization is the large quantities of data that needs to be passed around. This makes it hard for us to implement a simple parallelization scheme that doesn't depend on the particular hardware available.

Matlab has problems dealing with non-vectorized operations on arrays and it's impossible to vectorize trivially the forwards-backwards algorithm. By rewriting that part of Sohrab Shah's code, we gained a 100-fold increase in speed in forwards-backwards, resulting in a $\approx$ 20-fold increase for *hmmmix-hard*. The exact factor depends on the size of the data because it determines the characteristics of the new bottleneck.

There are various places in the algorithm (e.g. during patient assignments) where we could theoretically speed things up by using only a subset of the full sequence $t = 1, \ldots, T$ of data. If we were to do that, though, it would make more sense to incorporate it into the pre-processing step where experts with domain-specific knowledge can point to certain regions where we should draw more samples. To use sequential data with varying step sizes we need to use non-stationary transition matrices in the forward-backwards algorithm. Our implementation of forwards-backwards supports this, but it hasn't been incorporated into the *hmmmix-soft* algorithm.

## 4.9 Free energy variant

We haven't explained in section 3.3.3 how to choose the constant $\tau$ in the Dirichlet $(\tau, \ldots, \tau)$ prior on the parameters defining the distributions $f(G^p)$. It really depends on the nature of

the data and on the objectives.

We find in [Ros98] a very interesting discussion about the use of an entropy term scaled by a factor $S^{-1}$ to control the formation and fusion of clusters. This is similar to the annealing process in metallurgy where we heat metals just above the re-crystallization temperature and then let the metal cool down in a controlled way. This gets rid of the crystal defects and the internal stresses which they cause [Wik09]. In our case, our value of $\tau$ in the prior corresponds exactly to a scaling of $S^{-1} = \tau$ to the entropy terms $\mathbb{H}(G^p)$.

Justifying that scaling term as a prior hyperparameter is easy within a Bayesian framework, but tinkering with the entropy term in variational methods is a bit harder to justify as it affects the maximum (in terms of $\theta$) that the original likelihood had. A "safe" thing to try is to start at equilibrium with $\tau = 1$, let $\tau$ grow slowly, have the clusters move around and then come back to $\tau = 1$ hoping to get better final clusterings. That isn't to say that it is justified to change priors as we iterate in EM, but changing the scaling factor of the entropy terms can smooth out the objective function and allow us to escape from local maxima. A compelling example of this is presented in [UN98]. They start with a small value of $S < 1$ and they let $S$ increase monotonically. At $S = 1$ they obtain the original objective function, but the final variational parameters obtained are closer to the global optimum than they would have been otherwise.

Empirically, in many of the cases where we used the prior $\tau = 1$, the algorithm took less than 10 iterations to converge, assigning many patients to one chain each. Selecting a higher value of $\tau$ slowed the convergence but the same final hard assignments often resulted. We didn't change the value of $\tau$ while running *hmmmix-soft* and preferred to view $\tau$ as part of the prior instead of a modified entropy term. If it wasn't for the success of [UN98] and [Ros98], though, we might not have defined a prior on $G^{1:P}$ as we did in section 3.3.3.

Although we haven't defined a Dirichlet prior on the states of the chains $M_t^{1:G}$, we could proceed by adding a scaling factor of $S^{-1}$ to the entropy terms $\mathbb{H}\left(M_{1:T}^g\right)$ in (3.14). This would have the effect of scaling the $R_{t,g}[k]$ terms by $S$ as well as raising the transition matrices $A^g$ and initial states priors $\pi^g$ to the power $S$ (renormalizing appropriately). It is also possible to play with the definition of the model to limit this scaling effect to only the $R_{t,g}[M_t^g]$ terms (without affecting the transition matrices). Smaller values of $S$ result in the chains being less affected by evidence when tuned by forwards-backwards. If we rewrite the complete data log-likelihood (3.3) with scaled entropy terms on both $G^{1:P}$ and $M_{1:T}^{1:G}$, we get

$$\mathcal{L}(h) = \int h(M_{1:T}^{1:G})h(G^{1:P}) \log f(Y, M, G, A, \pi, \theta) + S_M \mathbb{H}[h(M_{1:T}^g)] + S_G \mathbb{H}[h(G^{1:P})] \quad (4.3)$$

One of the approaches tried (to reduce the influence of bad initial clustering) involved having one cluster for every patient and then letting the clusters merge by running *hmmmix-soft*.

The clusters would only merge if given enough incentives by way of scaling the entropy terms sufficiently. In one particular experiment, we observed that smaller values of $S$ (i.e. large $\tau$) led to better results by having the clusters merge progressively instead of suddenly collapsing from 100 clusters to 13. The conclusions of many experiments were heuristic rules like this that were not really generalizable.

If we choose to scale the entropy terms by such factors, to select $\tau$ we have to take into consideration the length $T$ of the sequences. We can see in (3.24) how the $\tau^{-1}$ factor is fighting against the sum $\sum_{t=1}^{T}$. A good value of $\tau$ when $T = 10^5$ might not be good when $T = 10^8$. The same applies to the pseudocounts on the transition matrices. They should be chosen based on how many transitions we expect in the whole sequence.

# Chapter 5

# Conclusion and future work

## 5.1 Summary

In this thesis we showed how the *hmmmix* model from [Sha08] can be trained using variational methods. This gives the model more flexibility in terms of allowing partial assignments of patients to clusters and controlling the weight that evidence has on the training process. We studied the effects of using different numbers of clusters, compared k-medoids cluster initialization to random cluster initialization, and tried a few methods to determine the best number of clusters to use.

We compared the performance of our approach to that of [Sha08] on synthetic data using their benchmark test and with data sampled from the model. We showed that our method generally leads to better results, with bigger improvements in cases where k-medoids fails to produce good cluster initializations. We also compared the two methods on real data in terms of the visual representation of the segments where mutations occur.

Finally, we analyzed the computational complexity and memory usage of our algorithm and showed how a low-memory implementation can be used to handle very large datasets without paying an unreasonable price on performance. We briefly addressed the issue of parallelization and other optimization techniques.

## 5.2 Future work

Our initial goal was to have a method that can completely do away with the need for good clustering initialization. In most experiments, the results were still better using k-medoids instead of a random initialization. If we had a real dataset on which k-medoids was useless, we could check if *hmmmix-soft* can get good results.

We briefly mentioned the possibility of selecting special subsets of the gene sequences to focus on more promising segments. This is one of the next natural steps and some portions of our code already support non-stationary transition matrices for this. An experiment could be

done to quantify the performance losses incurred when using significantly less observations than are available. It would be easier in that case to control the trade-off between the quality of the results and the execution time as determined by the size of the datasets.

There is also work to be done to implement a more efficient version of *hmmmix-soft* that exploits parallelism. Multicore processors are now the norm and one could imagine how *hmmmix-soft* could be made to run on Amazon's Elastic Compute Cloud. It would be interesting to know what is actually worth doing in practice given specific hardware.

As mentioned in [RIF+06], the deletion mutations are on average three times shorter than the higher copy number mutations. It could be possible to pick our model parameters taking that fact into consideration. It would have been possible also to study what set of values for hyperparameters generate data that comes closest to real experimental data. Maybe it would have been possible to use in some way the observation from [RIF+06] that the proportion of chromosomes susceptible to CNVs varies from 6% to 19%. These are just examples of possible improvements to the model based on domain knowledge. There are probably many others, and we don't really know if we'd get significantly better results by complicating the model. One of the good features of the original *hmmmix* model was that it could discover interesting patterns in an unsupervised way without much assumptions. With domain specific priors, we can't argue a model's validity by the discovered patterns if those patterns were programmed into the model in the first place.

Various methods exist to accelerate the convergence of the EM algorithm (see [SRG03], [SR03], [BKS97], [MvD97]). As observed in section 4.9, however, the choice of a large value for $\tau$ slowed down the algorithm but generally improved the quality of the solutions found. We spent limited time exploring heuristic methods to accelerate the convergence of *hmmmix-soft*, but it might be possible to use a clever optimization technique to reach the fixed point of the parameter updates (3.27), (3.28), (3.29) faster, for example. In that case, we might still be able to use an appropriate value of $\tau$ while speeding up some of the internal loops illustrated in figure 3.2.

# Bibliography


[Arc05] Cédric Archambeau, *Probabilistic models in noisy environments*, Ph.D. thesis, Université catholique de Louvain, 2005.

[Bis06] C. Bishop, *Pattern recognition and machine learning*, Springer, 2006.

[BKS97] E. Bauer, D. Koller, and Y. Singer, *Batch and on-line parameter estimation in Bayesian networks*, 1997.

[BMR97] J. Binder, K. Murphy, and S. Russell, *Space-efficient inference in dynamic probabilistic networks*, International Joint Conference on Artificial Intelligence, vol. 15, Citeseer, 1997, pp. 1292–1296.

[CC00] Y. Cheng and G.M. Church, *Biclustering of expression data*, Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology table of contents, AAAI Press, 2000, pp. 93–103.

[CSS+09] K.J.J. Cheung, S.P. Shah, C. Steidl, N. Johnson, T. Relander, A. Telenius, B. Lai, K.P. Murphy, W. Lam, A.J. Al-Tourah, et al., *Genome-wide profiling of follicular lymphoma by array comparative genomic hybridization reveals prognostically significant DNA copy number imbalances*, Blood **113** (2009), no. 1, 137.

[GJ97] Z. Ghahramani and M.I. Jordan, *Factorial hidden Markov models*, Machine learning **29** (1997), no. 2, 245–273.

[HY01] M.H. Hansen and B. Yu, *Model selection and the principle of minimum description length*, Journal of the American Statistical Association **96** (2001), no. 454, 746–774.

[JAVDD+06] MCJ Jongmans, RJ Admiraal, KP Van Der Donk, L. Vissers, AF Baas, L. Kapusta, JM van Hagen, D. Donnai, TJ de Ravel, JA Veltman, et al., *CHARGE syndrome: the phenotypic spectrum of mutations in the CHD7 gene*, Journal of medical genetics **43** (2006), no. 4, 306–314.

[Koe08] Hoyt Koepke, *Bayesian Cluster Validation*, Master's thesis, University of British Columbia, 2008.

[LR95] C. Liu and D.B. Rubin, *ML estimation of the t distribution using EM and its extensions, ECM and ECME*, Statistica Sinica **5** (1995), no. 1, 19–39.

[MTSc+07] S.C. Madeira, M.C. Teixeira, I. Sá-correia, A.L. Oliveira, A.L. Oliveira, and K. Dis, *Identification of regulatory modules in time-series gene expression data using a linear time biclustering algorithm*, IEEE/ACM Transactions on Computational Biology and Bioinformatics (2007).

[MvD97] X. L. Meng and D. van Dyk, *The EM algorithm — an old folk song sung to a fast new tune (with Discussion)*, J. Royal Stat. Soc. B **59** (1997), 511–567.

[RIF+06] R. Redon, S. Ishikawa, K.R. Fitch, L. Feuk, G.H. Perry, T.D. Andrews, H. Fiegler, M.H. Shapero, A.R. Carson, W. Chen, et al., *Global variation in copy number in the human genome*, Nature **444** (2006), no. 7118, 444–454.

[RLHR+06] A. Rovelet-Lecrux, D. Hannequin, G. Raux, N. Le Meur, A. Laquerrière, A. Vital, C. Dumanchin, S. Feuillette, A. Brice, M. Vercelletto, et al., *APP locus duplication causes autosomal dominant early-onset Alzheimer disease with cerebral amyloid angiopathy*, Nature genetics **38** (2006), 24–26.

[Ros98] K. Rose, *Deterministic annealing for clustering, compression, classification, regression, and related optimization problems*, Proceedings of the IEEE **86** (1998), no. 11, 2210–2239.

[Sch78] G. Schwarz, *Estimating the dimension of a model*, The annals of statistics (1978), 461–464.

[SFJ+03] AB Singleton, M. Farrer, J. Johnson, A. Singleton, S. Hague, J. Kachergus, M. Hulihan, T. Peuralinna, A. Dutra, R. Nussbaum, et al., *$\alpha$-Synuclein locus triplication causes Parkinson's disease*, 2003, pp. 841–841.

[Sha08] Sohrab Shah, *Model based approaches to array CGH data analysis*, Ph.D. thesis, University of British Columbia, 2008.

[SJJ+09] S. Shah, K. J. Cheung Jr., N. Johnson, R. Gascoyne, D. Horsman, R. Ng, G. Alain, and K. Murphy, *Model-based clustering of array CGH with*, Bioinformatics (2009).

[SR03] Ruslan Salakhutdinov and Sam Roweis, *Adaptive overrelaxed bound optimization methods*, Proceedings of the International Conference on Machine Learning, vol. 20, 2003, pp. 664–671.

[SRG03] Ruslan Salakhutdinov, Sam T. Roweis, and Zoubin Ghahramani, *Optimization with EM and Expectation-Conjugate-Gradient*, 2003.

[TSK05] P.N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2005.

[TWH01] R. Tibshirani, G. Walther, and T. Hastie, *Estimating the number of clusters in a data set via the gap statistic*, Journal of the Royal Statistical Society. Series B, Statistical Methodology (2001), 411–423.

[UN98] N. Ueda and R. Nakano, *Deterministic annealing EM algorithm*, Neural Networks **11** (1998), 271–282.

[Wik09] Wikipedia, *Annealing (metallurgy) — Wikipedia, the free encyclopedia*, 2009, [Online; accessed 15-May-2009].

# Appendix A

# Derivation of update formulas for parameters

We show in this section the derivations for the update formulas of the parameters $\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$ found in section 3.3.4. We start from

$$L(\theta) = \int h(M_{1:T}^{1:G}) h(G^{1:P}) \log f(Y_{1:T}^{1:P} | M_{1:T}^{1:G}, G^{1:P}, \theta) f(\theta)$$

$$= \int h(M_{1:T}^{1:G}) h(G^{1:P}) \left[ \log \int f(Y_{1:T}^{1:P} | Z_{1:T}^{1:P}, \theta) f(Z_{1:T}^{1:P} | M_{1:T}^{1:G}, G^{1:P}) dZ_{1:T}^{1:P} + \log f(\theta) \right] \tag{A.1}$$

$$\geq \int \left[ h(M_{1:T}^{1:G}) h(G^{1:P}) f(Z_{1:T}^{1:P} | M_{1:T}^{1:G}, G^{1:P}) dM_{1:T}^{1:G} dG^{1:P} \right] \left[ \log f(Y_{1:T}^{1:P} | Z_{1:T}^{1:P}, \theta) f(\theta) \right] dZ_{1:T}^{1:P} \tag{A.2}$$

where we want to maximize (A.2) with respect to $\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$.

For the rest of this section, we will be dealing with the distribution

$$q(Z_{1:T}^{1:P}) = \int h(M_{1:T}^{1:G}) h(G^{1:P}) f(Z_{1:T}^{1:P} | M_{1:T}^{1:G}, G^{1:P}) dM_{1:T}^{1:G} dG^{1:P} \tag{A.3}$$

from equation (A.2). We can rewrite (A.2) as an expectation

$$\mathbb{E}_{q(Z_{1:T}^{1:P})} \left[ \log f(Y_{1:T}^{1:P} | Z_{1:T}^{1:P}, \theta) f(\theta) \right].$$

However, as $q(Z_{1:T}^{1:P}) = \prod_t \prod_p q(Z_t^p)$, it will be more convenient to define a special notation for the quantities $q(Z_t^p = k)$ and use them to write the formulas. Thus, we define :

$$\rho_t^p(k) := \int h(M_{1:T}^{1:G}) h(G^{1:P}) f(Z_t^p = k | M_{1:T}^{1:G}, G^{1:P}) dM_{1:T}^{1:G} dG^{1:P} \tag{A.4}$$

$$= \int h(M_t^{1:G}) h(G^{1:P}) f(Z_t^p = k | M_t^{1:G}, G^{1:P}) dM_t^{1:G} dG^{1:P} \tag{A.5}$$

$$= \sum_{g=1}^{G} \mathcal{C}_g^p \sum_{j=1}^{K} h(M_t^g = j) f(Z_t^p = k | M_t^g = j, G^p = g) \tag{A.6}$$

As illustrated at line (A.5), these quantities are easily computed from the smoothed marginals $h(M_t^g)$ because the $Z_t^p$ are independent given the chains $h(M_{1:T}^{1:G})$. We need the partial patients

assignments $\mathcal{C}_g^p$ and the parameters $\alpha_L, \alpha_N, \alpha_G$ that govern $f(Z_t^p = k | M_t^g = j, G^p = g)$ to evaluate $\rho_t^p(k)$. At line (A.6) we find a more "algorithmic" expression. Note that the values of $f(Z_t^p = k | M_t^g = j, G^p = g)$ defined by equation (1.1) don't depend on the chain, the patient or the time so we can define a K-by-K matrix $S(j,k) := f(Z_t^p = k | M_t^g = j, G^p = g)$ to precompute and cache the values in (A.6).

We rewrite our lower bound (A.2) as

$$\sum_k \sum_p \sum_t \rho_t^p(k) \log f(Y_t^p | Z_t^p = k, \mu_k^p, \lambda_k^p) + \sum_k \sum_p \log f(\mu_k^p, \lambda_k^p) \tag{A.7}$$

that we now want to maximize with respect to $\theta = (\mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P})$. We will temporarily refer to the first term of (A.7) as $\mathcal{A}_1$ and to the second as $\mathcal{A}_2$ for convenience. Note that, among all the hyperparameters $(\nu_{1:K}^{1:P}, \eta_{1:K}^{1:P}, m_{1:K}^{1:P}, \gamma_{1:K}^{1:P}, S_{1:K}^{1:P})$, $\mathcal{A}_1$ depends on $\nu_{1:K}^{1:P}$ and $\mathcal{A}_2$ depends on $\eta_{1:K}^{1:P}$, $m_{1:K}^{1:P}$, $\gamma_{1:K}^{1:P}$,$S_{1:K}^{1:P}$.

As Archambeau explains in [Arc05], there is no closed form solution for estimating the parameters of a Student-t, but we can always use a EM trick from Liu and Rubin [LR95] involving a latent variable $u$ to estimate $\mu, \lambda$. We follow that procedure, adopting the notation from [Arc05]. This essentially amounts to decomposing $\mathcal{A}_1$ as

$$\sum_k \sum_p \sum_t \rho_t^p(k) \log \int \mathcal{N}\left(Y_t^p | \mu_k^p, u\lambda_k^p\right) \text{Gamma}\left(u | \frac{\nu_k^p}{2}, \frac{\nu_k^p}{2}\right) du \tag{A.8}$$

and maximizing instead the quantity

$$\mathcal{A}_1^* = \sum_k \sum_p \sum_t \rho_t^p(k) \log \mathcal{N}\left(Y_t^p | \mu_k^p, \bar{u}_t^p(k)\lambda_k^p\right) + \log \text{Gamma}\left(\bar{u}_t^p(k) | \frac{\nu_k^p}{2}, \frac{\nu_k^p}{2}\right). \tag{A.9}$$

We use $\bar{u}_t^p(k)$ to denote the fact that we have such a latent variable $u$ for all patients $1{:}P$, all time steps $1{:}T$ and all states $1{:}K$.

The prior $\mathcal{A}_2$ was defined in section 2.1 to be

$$\mathcal{A}_2 = \sum_k \sum_p \log \mathcal{N}\left(\mu_k^p | m_k^p, \text{precision} = \lambda_k^p \eta_k^p\right) + \log \text{Gamma}\left(\lambda_k^p | \gamma_k^p, S_k^p\right). \tag{A.10}$$

We are now looking for a set of values $\bar{u}_t^p(k), \mu_k^p, \lambda_k^p$ for which the derivatives of $\mathcal{A}_1^* + \mathcal{A}_2$ with respect to all these variables are equal to zero. There are no constraints other than $\lambda_{1:K}^{1:P} > 0$ because the precision is a positive quantity. Dropping all constants not involving $\mu_k^p, \lambda_k^p, \bar{u}_t^p(k)$,

we get that $\mathcal{A}_1^* + \mathcal{A}_2$ is equal to

$$\sum_k \sum_t \sum_p \rho_t^p(k) \left[ \frac{1}{2} \log\left(\bar{u}_t^p(k)\lambda_k^p\right) - \frac{\bar{u}_t^p(k)\lambda_k^p}{2}\left(Y_t^p - \mu_k^p\right)^2 + \left(\frac{\nu_k^p}{2} - 1\right)\log \bar{u}_t^p(k) - \frac{\nu_k^p}{2}\bar{u}_t^p(k) \right] + \sum_k \sum_p \left[ \frac{1}{2}\log \lambda_k^p - \frac{\eta_k^p \lambda_k^p}{2}\left(\mu_k^p - m_k^p\right)^2 + \left(\gamma_k^p - 1\right)\log \lambda_k^p - S_k^p \lambda_k^p \right]. \tag{A.11}$$

We now set all the derivatives to zero and see what equalities must hold for us to have a maximum. First, we set $\frac{\partial(\mathcal{A}_1^* + \mathcal{A}_2)}{\partial \bar{u}_t^p(k)} = 0$ :

$$0 = \rho_t^p(k)\left[\frac{1}{2}\bar{u}_t^p(k)^{-1} - \frac{\lambda_k^p}{2}\left(Y_t^p - \mu_k^p\right)^2 + \left(\frac{\nu_k^p}{2} - 1\right)\bar{u}_t^p(k)^{-1} - \frac{\nu_k^p}{2}\right] \tag{A.12}$$

$$\bar{u}_t^p(k)^{-1}\left[\frac{1}{2} + \left(\frac{\nu_k^p}{2} - 1\right)\right] = \frac{\lambda_k^p}{2}\left(Y_t^p - \mu_k^p\right)^2 + \frac{\nu_k^p}{2} \tag{A.13}$$

$$\bar{u}_t^p(k) = \frac{\frac{1}{2} + \frac{\nu_k^p}{2} - \frac{2}{2}}{\frac{\lambda_k^p}{2}\left(Y_t^p - \mu_k^p\right)^2 + \frac{\nu_k^p}{2}} = \frac{\nu_k^p - 1}{\lambda_k^p\left(Y_t^p - \mu_k^p\right)^2 + \nu_k^p} \tag{A.14}$$

Then, we set $\frac{\partial(\mathcal{A}_1^* + \mathcal{A}_2)}{\partial \mu_k^p} = 0$ :

$$0 = \sum_t \rho_t^p(k)\left[-\frac{2}{2}\bar{u}_t^p(k)\lambda_k^p\left(Y_t^p - \mu_k^p\right)(-1)\right] - \frac{2\eta_k^p\lambda_k^p}{2}\left(\mu_k^p - m_k^p\right) \tag{A.15}$$

$$0 = \sum_t \rho_t^p(k)\,\bar{u}_t^p(k)\lambda_k^p Y_t^p + \eta_k^p\lambda_k^p m_k^p + \sum_t \rho_t^p(k)\,\bar{u}_t^p(k)\lambda_k^p(-1)\mu_k^p - \eta_k^p\lambda_k^p\mu_k^p \tag{A.16}$$

$$\mu_k^p = \frac{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k)\lambda_k^p Y_t^p + \eta_k^p\lambda_k^p m_k^p}{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k)\lambda_k^p + \eta_k^p\lambda_k^p} = \frac{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k)Y_t^p + \eta_k^p m_k^p}{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k) + \eta_k^p} \tag{A.17}$$

Finally, we set $\frac{\partial(\mathcal{A}_1^* + \mathcal{A}_2)}{\partial \lambda_k^p} = 0$ to have :

$$\begin{aligned} 0 = \quad & \textstyle\sum_t \rho_t^p(k)\left[\frac{1}{2}(\lambda_k^p)^{-1} - \frac{\bar{u}_t^p(k)}{2}\left(Y_t^p - \mu_k^p\right)^2\right] \\ & + \tfrac{1}{2}(\lambda_k^p)^{-1} - \tfrac{\eta_k^p}{2}\left(\mu_k^p - m_k^p\right)^2 + \left(\gamma_k^p - 1\right)(\lambda_k^p)^{-1} - S_k^p \end{aligned} \tag{A.18}$$

$$\begin{aligned} 0 = \quad & (\lambda_k^p)^{-1}\left[\tfrac{1}{2}\textstyle\sum_t \rho_t^p(k) + \tfrac{1}{2} + (\gamma_k^p - 1)\right] \\ & - \left[\textstyle\sum_t \rho_t^p(k)\,(-\tfrac{1}{2})\bar{u}_t^p(k)\left(Y_t^p - \mu_k^p\right)^2 - \tfrac{\eta_k^p}{2}\left(\mu_k^p - m_k^p\right)^2 - S_k^p\right] \end{aligned} \tag{A.19}$$

$$(\lambda_k^p)^{-1} = \frac{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k)\left(Y_t^p - \mu_k^p\right)^2 + \eta_k^p\left(\mu_k^p - m_k^p\right)^2 + 2S_k^p}{\sum_t \rho_t^p(k) + 2\gamma_k^p - 1} \tag{A.20}$$

We iterate a few times updating the quantities in the order $(A.14), (A.17), (A.20)$ until they stabilize. We don't need to update the values for $\rho_t^p(k)$ as we iterate as they don't depend on $\theta$.

# Appendix B

# Measures of performance for clusterings

## B.1 Silhouette coefficient

The silhouette coefficient is a measure of clustering quality. By performing clustering with different numbers of clusters and evaluating the silhouette coefficients, we can pick the best number of clusters as the one that maximizes the silhouette coefficient.

For each data point, we compute its silhouette coefficient as

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{B.1}$$

where $a_i$ is the average distance to points in $i$'s cluster and $b_i$ is the minimum average distance to points in another cluster. The silhouette coefficient for a given clustering is the average of the silhouette coefficients for the points :

$$S = \frac{1}{N} \sum_{i=1}^{N} s_i. \tag{B.2}$$

Values for silhouette coefficients are always found in the interval $[-1, 1]$ and with higher values corresponding to more desirable clustering configurations.

## B.2 Jaccard coefficient

The Jaccard coefficient is a measure of clustering quality that requires knowledge of the true assignments. Its value is in the interval $[0, 1]$ and a value of 1 can be achieved iff we learned the true assignments.

Following the notation of [TSK05], we take the true clustering and we let $C(i, j)$ be 1 if data points $i, j$ are in the same cluster and 0 otherwise. For the predicted clustering, we let $P(i, j)$ be 1 if data points $i, j$ are in the same cluster and 0 otherwise. We define

- $f_{00}$ : the number of pairs of points $(i, j)$ for which $C(i, j) = 0$ and $P(i, j) = 0$
- $f_{01}$ : the number of pairs of points $(i, j)$ for which $C(i, j) = 0$ and $P(i, j) = 1$
- $f_{10}$ : the number of pairs of points $(i, j)$ for which $C(i, j) = 1$ and $P(i, j) = 0$
- $f_{11}$ : the number of pairs of points $(i, j)$ for which $C(i, j) = 1$ and $P(i, j) = 1$.

The Jaccard coefficient is defined as

$$\frac{f_{11}}{f_{01} + f_{10} + f_{11}}. \tag{B.3}$$

## B.3 Gap statistic

This section of the appendix is not meant to be an explanation of the theory behind the gap statistic. It is only a description of the method that we followed to get the results from section 4.2.

We explain how the gap statistic can be used to estimate the true number of clusters for a given dataset $\mathcal{D}$. The experiment has to be repeated for a number of datasets in order for us to reach any reliable conclusion about the use of the gap statistic, but we will turn to that after explaining the method on a dataset $\mathcal{D}$.

For a given clustering $\mathcal{E} = (E_1, \ldots, E_C)$ of the data $\mathcal{D}$, we define the within cluster spread as

$$\text{spread}(E_i) = \frac{1}{2|E_i|} \sum_{x,y \in E_i} \text{dist}(x, y). \tag{B.4}$$

for some choice of distance function. We use here the square Euclidean distance $\text{dist}(x, y) = \|x - y\|_2^2$. We also define the spread for the set of clusters $\mathcal{E}$ as

$$\text{spread}(\mathcal{E}) = \sum_{i=1}^{C} \text{spread}(E_i). \tag{B.5}$$

We need a "null set" to serve as baseline measure so we generate a series of datasets $D_1^1, D_1^2, \ldots, D_1^M$ having a single cluster. For $C = 1, \ldots, C_{\text{MAX}}$, we perform our clustering (using *hmmmix-soft* with random initializations, in this particular case). For every dataset $D_1^i$, we let $B_C^i = \text{spread}(E_1^i, \ldots, E_C^i)$ where $E_1^i, \ldots, E_C^i$ are the resulting clusters from dataset $D_1^i$ with the method using $C$ initial clusters. These values will come into play later in the role of $\frac{1}{M} \sum_{i=1}^{M} \log B_C^i$.

Now we go back to our original dataset $\mathcal{D}$ having $G$ clusters and we use our clustering method again using $C = 1, \ldots, C_{\text{MAX}}$ clusters, repeating this process $N$ times for every value of $C$.

From this we define quantities $A_C^i$ as the spread of the $i^{th}$ set of clusters obtained from the method with $C$ clusters.

We are interested in the estimated expections $\tilde{A_C} = \frac{1}{N}\sum_{i=1}^{N} \log A_C^i$ as well as the estimates of the standard deviations of those quantities defined as

$$s_C = \sqrt{\frac{1}{N+1}\sum_{i=1}^{N}\left(\log A_C^i - \tilde{A_C}\right)^2}. \tag{B.6}$$

The gap statistic is defined as

$$\text{Gap}(C) = \frac{1}{N}\sum_{i=1}^{N} \log A_C^i - \frac{1}{M}\sum_{i=1}^{M} \log B_C^i \tag{B.7}$$

and the estimate for the number of clusters is given by

$$\min_C \left\{C | \text{Gap}(C) \geq \text{Gap}(C+1) - s_{C+1}\right\}. \tag{B.8}$$

This procedure is for a single dataset $\mathcal{D}$. To get a good idea of the reliability of the gap statistic, we have to do this multiple times by generating other datasets with $G$ clusters. The estimates $\frac{1}{M}\sum_{i=1}^{M} \log B_C^i$ can be reused throughout the trials, and since their role is important it makes sense to use a large value of $M$.