Variational Methods for Detecting Copy Number Alterations

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)

June 2009

Abstract

DNA copy number alterations (CNAs) are genetic changes that can produce adverse effects in numerous human diseases, including cancer. Various techniques exist to identify them, but the problem is far from solved and the prohibitive computational costs rule out many approaches.

We show how the current state-of-the-art model *hmmmix* can be trained using variational methods with minimal assumptions. We show how this allows for partial patients clustering and how it partly addresses the difficult issue of determining the number of clusters to use. We compare our technique with the current best and show how it can handle very large 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.

Chapter 1

Introduction

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

We believe Shah's model would perform better and would be more flexible if it were trained using variational methods instead of using its current approximations. We propose to explore this approach and compare it to the one introduced by [Sha08].

1.1 Our contributions

Our contributions to the *hmmmix* model are three-fold. First, we use variational methods can be used to extend the model. The most interesting consequence of this is the possibility to assign patients to multiple clusters. 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 also discuss the addition of scaling constants to the variational entropy terms in order to smooth out the algorithm.

Finally, in addition to providing the normal code for our implementation, 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 considerably the original code from [Sha08].

1.2 Outline of the thesis

In chapter 2, we describe the hmmmix model completely and discuss how [Sha08] learn the parameters. This model is the basis of this thesis.

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.

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 also analyze the computational/memory complexity of the 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 will provide a complete description of the hmmmix model introduced 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.

Although this chapter will cover the *hmmmix* model in details, we will only summarize the *hmmmix-hard* method because it is not the object of our study. The reader can always refer to [Sha08] or [SJJ+09] for more details.

Certain parameters of the model can be given priors, but to avoid putting the ox before the plow, these priors will be discussed in (2.2) after the core of the *hmmmix* model is explained in (2.1). One could also argue that the priors could be considered part of the inference mechanism and not really part of the model (as we are not integrating over them). An important component of the model comes in the definition of $p(Y_{1:T}^{1:P}|Z_{1:T}^{1:P}, \theta)$ and the method to learn the parameters θ. For this, [Sha08] borrows from [Arc05] but the actual formulas from the latter have to be adapted to our particular case.

2.1 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 2.1 shows the graphical representation of the model.

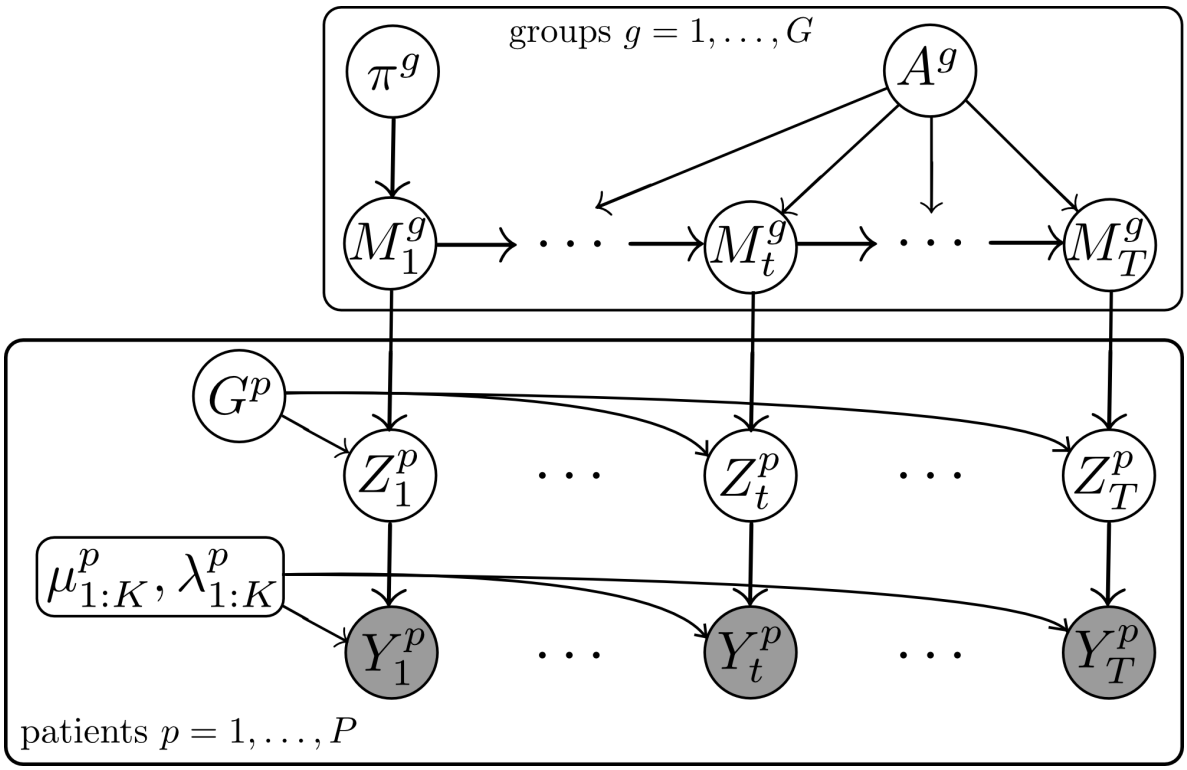


Figure 2.1: The *hmmmix* model. When generating data, the parameters $\pi^{1:G}, A^{1:G}, \mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$ are fixed. The parameters $\alpha_L, \alpha_N, \alpha_G$ are not shown here.

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=normal, 3=gain*$\}$. For every g, we sample one full sequence $M_{1:T}^g$ from a hidden Markov chain with corresponding prior π^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 $M_{1:T}^g$ (for $G^p = g$) and we sample a sequence of hidden states $Z_{1:T}^p$ associated to that patient. Note that the $Z_{1:T}^p$ sequence is not a Markov chain. The discrete value of Z_t^p 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=normal, 3=gain*$\}$ of the source M_t^g.

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.

Finally, we generate the observations $Y_{1:T}^{1:P}$ given $Z_{1:T}^{1:P}$.

$$p(Y_t^p|Z_t^p = k) = \text{Student}\left(Y_t^p|\text{mean}{=}\mu_k^p, \text{precision}{=}\lambda_k^p, \text{degrees of freedom}{=}\nu_k^p\right) \tag{2.1}$$

where

$$\text{Student}\left(x|\mu,\lambda,\nu\right) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)}\sqrt{\frac{\lambda}{\nu\pi}}\left[1+\frac{\lambda}{\nu}\left(\mu-x\right)^2\right]^{-\frac{\nu+1}{2}} \tag{2.2}$$

2.2 Priors for learning the parameters

There are two places in this model where priors are used to learn the 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 3.3.3 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 λ_k^p, we put a Gamma prior with hyperparameters γ_k^p, S_k^p also indexed by k, p :

$$p\left(\lambda_k^p|\gamma_k^p, S_k^p\right) = \text{Gamma}\left(\lambda_k^p|\gamma_k^p, S_k^p\right) \tag{2.3}$$

where

$$\text{Gamma}\left(x|a,b\right) = \frac{1}{\Gamma(a)} b^a x^{a-1}\exp\left(-bx\right). \tag{2.4}$$

On the means μ_k^p, we put a normal prior with hyperparameters m_k^p, η_k^p :

$$p\left(\mu_k^p|m_k^p,\eta_k^p,\lambda_k^p\right) = \mathcal{N}\left(\mu_k^p|\text{mean}{=}m_k^p, \text{precision}{=}\eta_k^p\lambda_k^p\right) \tag{2.5}$$

where

$$\mathcal{N}\left(x|\text{mean}{=}\mu, \text{precision}{=}\lambda\right) = \sqrt{\frac{\lambda}{2\pi}}\exp\left(-\frac{\lambda}{2}\left(x-\mu\right)^2\right). \tag{2.6}$$

It should be noted that, in [Sha08], the hyperparameters m_k^p, ν_k^p are only indexed as m_k, ν_k. From personal correspondence with the author, we learned that this was how his *hmmmix-hard* was implemented and 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, ν_k indexing.

Finally, there is a certain amount of preprocessing done on the experimental data $Y_{1:T}^{1:P}$ 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 representation {*1=loss, 2=normal, 3=gain*}.

2.3 Inference in the hmmmix-hard model

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 iterative conditional means, 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 3.3.3, 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 3.3.3 of this thesis.

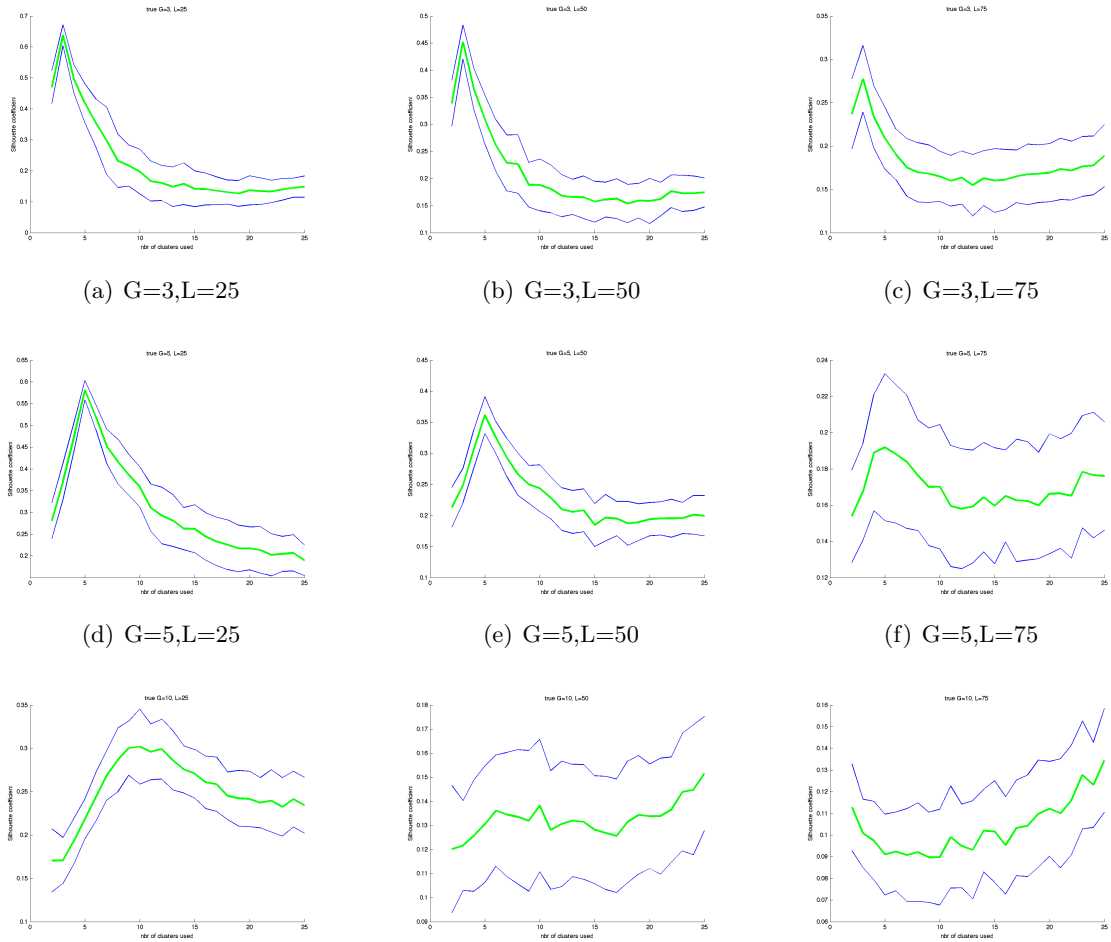
2.4 Determining the number of clusters

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. 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 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. For $G_0 = 2, \ldots, 25$, we looked at the silhouette coefficients obtained when selecting the best clustering from k-medoids with G_0 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 2.2.



(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 2.2: 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 green curve is the average of the results for 50 datasets and the 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.

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).

There is also something unsatisfying about the silhouette coefficient because it does not fit into the statistical framework of the model. The bayesian information criterion (BIC) is a natural way to 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.

The reason why the k-medoids algorithm is used instead of the classic k-means is because it 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.

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*. This makes learning intractable unless we are willing to make concessions. [Sha08] assigns temporary 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 independently.

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 case for hard patients assignments versus soft patients assignments is very similar to the case of k-means versus gaussian mixture models learned by EM. We hope that the patients will put their weights in single clusters depending on their CNA profiles, but we can only hope that the real data can be explained by such a clustering.

A very insightful paper on variational methods used for coupled hidden Markov chains is [GJ97]. Chapter 10 of [Bis06] also contains relevant material to justify optimizing the variational lower bound, but they focus more on indirected models and generalization to Power EP. Hidden Markov models are always special cases where we can do certain things in linear time so [GJ97] is the reference that we'll be following more closely. 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.

A crucial optimization step in section 3.3.1 requires a good eye to identify certain quantities 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 return briefly to the topic of cluster initialization in 3.4 and explain in 3.5 how the updates should be performed. In section 3.6, 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 π. 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

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

Thus, if we are faced with an expression of the type

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

that we want to maximize with respect to $h(M_{1:T})$ taken to be any joint distribution on $(M_1, \ldots, M_T) \in \{1, \ldots, K\}^T$, 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 follows from the uniqueness of the minimizing distribution Q in $\mathrm{KL}(Q\|P)$ when Q is allowed to take the value P.

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 Dirichlet$(A_{\mathrm{PC}}(j,1),\ldots,A_{\mathrm{PC}}(j,K))$ for row $j = 1,\ldots,K$.

The MLE with respect to A of equation (3.1) is given by the fowards-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).

$$\left(A^{\mathrm{MLE}}(j,1),\ldots,A^{\mathrm{MLE}}(j,K)\right) \propto \sum_{t=2}^{T} \xi_{t-1,t}(j,1{:}K) + A_{\mathrm{PC}}(j,1{:}K)$$

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 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 $\mathcal{L}(h) = \int h(W)\log f(Y,W|\tau)dW - \int h(W)\log h(W)$ where W are the hidden variables, Y are the observed variables and τ 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 makes the chains independent and 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 other purposes than for estimating the transition matrices A^g and initial state priors π^g, the distributions $h(M^g_{1:T})$ will propagate through the *hmmmix-soft* model in the same way as if we assumed that $h(M^g_{1:T}) = \prod_t h(M^g_t)$. We decided to marginalize out analytically the hidden variables $Z^{1:P}_{1:T}$ from the model. This is why we don't have a variational term $h(Z^{1:P}_{1:T})$, but $Z^{1:P}_{1:T}$ still play a part in the formulas.

The full data log-likelihood to be maximized is

$$\mathcal{L}(h) = \int h(M^{1:G}_{1:T})h(G^{1:P}) \log f(Y^{1:P}_{1:T}, M^{1:G}_{1:T}, G^{1:P}, A^{1:G}, \pi^{1:G}, \theta) dG^{1:P} dM^{1:G}_{1:T} - \int h \log h. \tag{3.2}$$

where we use θ as shorthand for $(\mu^{1:P}_{1:K}, \lambda^{1:P}_{1:K})$ and where f decomposes as

$$\begin{aligned} f(Y^{1:P}_{1:T}, M^{1:G}_{1:T}, G^{1:P}, A^{1:G}, \pi^{1:G}, \theta) =& f(Y^{1:P}_{1:T}|M^{1:G}_{1:T}, G^{1:P}, \theta) f(M^{1:G}_{1:T}|A^{1:G}, \pi^{1:G}) f(G^{1:P}) \\ & f(A^{1:G}|A_{\text{PC}}) f(\pi^{1:G}) f(\mu^{1:P}_{1:K}, \lambda^{1:P}_{1:K}|m^{1:P}_{1:K}, \eta^{1:P}_{1:K}, \gamma^{1:P}_{1:K}, S^{1:P}_{1:K}). \end{aligned} \tag{3.3}$$

It will be convenient also to let $\mathcal{C}^p_g$ 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 $\sum_{g=1}^G \mathcal{C}^p_g = 1$ for all patients p.

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

$$\mathbb{H}[h(M^{1:G}_{1:T})] = \sum_{g=1}^G \mathbb{H}[h(M^g_{1:T})] \qquad \text{and} \qquad \mathbb{H}[h(G^{1:P})] = \sum_{p=1}^P \mathbb{H}[h(G^p)] \tag{3.4}$$

3.3 Training

3.3.1 Hidden chains

If we focus on maximizing (3.2) with respect to $h(M^{1:G}_{1:T})$, we can omit the unrelated terms and rewrite (3.2) as

$$\int h(M^{1:G}_{1:T})h(G^{1:P}) \left[\log f(Y^{1:P}_{1:T}|M^{1:G}_{1:T}, G^{1:P}, \theta) + \log f(M^{1:G}_{1:T}|A^{1:G}, \pi^{1:G})\right] dG^{1:P} dM^{1:G}_{1:T} + \mathbb{H}[h(M^{1:G}_{1:T})] \tag{3.5}$$

We start by reformulating the first term in the previous expression (3.5) to separate as much as possible the contributions of the G groups. We have that

$$\int h(M^{1:G}_{1:T})h(G^{1:P}) \log f(Y^{1:P}_{1:T}|M^{1:G}_{1:T}, G^{1:P}, \theta) dG^{1:P} dM^{1:G}_{1:T} \tag{3.6}$$

$$= \int h(M^{1:G}_{1:T}) \int h(G^{1:P}) \sum_t \log f(Y^{1:P}_t|M^{1:G}_t, G^{1:P}, \theta) dG^{1:P} dM^{1:G}_{1:T} \tag{3.7}$$

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

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

$$= \int h(M_{1:T}^{1:G}) \sum_t \sum_g R_{t,g}[M_t^g] dM_{1:T}^{1:G} \tag{3.10}$$

where, in that last line, we defined

$$R_{t,g}[M_t^g] = \sum_p \mathcal{C}_g^p \log f(Y_t^p | M_t^g, G^p = g, \theta). \tag{3.11}$$

Continuing on (3.10), we get

$$\int h(M_{1:T}^{1:G}) \sum_t \sum_g R_{t,g}[M_t^g] dM_{1:T}^{1:G} = \sum_g \int h(M_{1:T}^g) \sum_t R_{t,g}[M_t^g] dM_{1:T}^g. \tag{3.12}$$

We now come back to the original expression (3.5) using (3.12) and the entropy term decomposition (3.4) to rewrite our formula of interest (3.5) as

$$\sum_g \left[\int h(M_{1:T}^g) \left[\sum_t R_{t,g}[M_t^g] + \log f(M_{1:T}^g | A^g, \pi^g) \right] dM_{1:T}^g - \int h(M_{1:T}^g) \log h(M_{1:T}^g) dM_{1:T}^g \right]. \tag{3.13}$$

Because the distributions $f\left(M_{1:T}^g | A^g, \pi^g\right)$ from the *hmmmix* model are Markov chains, we can see that (3.13) 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 (3.2) with respect to $h(M_{1:T}^{1:G})$, we should run forwards-backwards on the chains independently for $g = 1, \ldots, G$ with $R_{t,g}[M_t^g = 1, \ldots, K]$ as observation log-likelihoods.

When implementing the algorithm, we represent internally $h(M_{1:T}^g)$ 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 (3.2) 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 use iterative conditional mode to optimize with respect to $h(M_{1:T}^{1:G})$ 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 $h(M_{1:T}^{1:G})$ and it is sufficient to keep only the smoothed marginals $\{h(M_t^g) | t = 1 \ldots T, g = 1 \ldots G\}$ as a K-by-T-by-G matrix.

3.3.2 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 $\mathcal{C}_g^p$ corresponding to the weight of the partial assignment of patient p in group g. Dropping all unrelated terms from (3.2), we get

$$\mathcal{L}(h) = \int h(M_{1:T}^{1:G}) h(G^{1:P}) \log f(Y_{1:T}^{1:P} | M_{1:T}^{1:G}, G^{1:P}, \theta) f(G^{1:P}) dM_{1:T}^{1:G} dG^{1:P} + \mathbb{H}\left(G^{1:P}\right) \quad (3.14)$$

$$= \int h(G^{1:P}) \left[\int h(M_{1:T}^{1:G}) \log f\left(Y_{1:T}^{1:P} | M_{1:T}^{1:G}, G^{1:P}, \theta\right) dM_{1:T}^{1:G} + \log f(G^{1:P}) \right] dG^{1:P} + \mathbb{H}\left(G^{1:P}\right) \quad (3.15)$$

$$= \int h(G^{1:P}) \left[\int h(M_{1:T}^{1:G}) \sum_p \sum_g \mathbb{I}(G^p = g) \log f\left(Y_{1:T}^{1:P} | M_{1:T}^{1:G}, G^p = g, \theta\right) \right] + \int h(G^{1:P}) \log f(G^{1:P}) + \mathbb{H}\left(G^{1:P}\right) \quad (3.16)$$

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

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

There are different ways to handle the $\int h(G^p) f(G^p)$ 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

$$\mathbb{H}\left(G^p\right) + \int h(G^p) \log f(G^p) = \sum_g \mathcal{C}_g^p \log \mathcal{C}_g^p + \sum_g \mathcal{C}_g^p (\tau - 1) \log \mathcal{C}_g^p + \mathrm{cst}(\tau) = \tau \mathbb{H}\left(G^p\right) + \mathrm{cst}(\tau) \quad (3.19)$$

With this prior, we develop (3.18) 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

$$\int h(G^p) \log(\sigma_g^p) - \int h(G^p) \log h(G^p) dG^p \quad (3.20)$$

where

$$\sigma_g^p = \exp\left(\tau^{-1} \int h(M_{1:T}^g) \log f\left(Y_{1:T}^p | M_{1:T}^g, G^p = g, \theta\right) dM_{1:T}^g\right). \tag{3.21}$$

The τ^{-1} in 3.21 comes from the τ coefficient in (3.19) by which we can divide the whole of expression (3.18). 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 τ) translates into more uniform softmaxima terms due to lighter evidence (being divided by τ).

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

$$\mathcal{C}_g^p = \frac{\exp\left(\tau^{-1} \int h(M_{1:T}^g) \log f\left(Y_{1:T}^p | M_{1:T}^g, G^p = g, \theta\right) dM_{1:T}^g\right)}{\sum_{g'} \exp\left(\tau^{-1} \int h(M_{1:T}^{g'}) \log f\left(Y_{1:T}^p | M_{1:T}^{g'}, G^p = g', \theta\right) dM_{1:T}^{g'}\right)} \tag{3.22}$$

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

The uniform prior on $f(G^p)$ corresponds to picking $\tau = 1$. This τ 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 into a Markov chain.

3.3.3 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 same prior as [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 $R_t^p[k]$.

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

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

$$= \int h(M_{1:T}^{1:G}) h(G^{1:P}) \left[\log \int f(Y_{1:T}^{1:P} | Z_{1:T}^{1:P}, \theta) f(Z_{1:T}^{1:P} | M_{1:T}^{1:G}, G^{1:P}) dZ_{1:T}^{1:P} + \log f(\theta)\right] \tag{3.24}$$

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

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 (3.24) to (3.25) are limited to the optimization of the parameter θ. We discuss this further in section 3.6.

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

$$q(Z_{1:T}^{1:P}) = \int h(M_{1:T}^{1:G})h(G^{1:P})f(Z_{1:T}^{1:P}|M_{1:T}^{1:G}, G^{1:P})dM_{1:T}^{1:G}dG^{1:P} \tag{3.26}$$

from equation (3.25). We can rewrite (3.25) 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 :

$$\begin{aligned}
\rho_t^p(k) &:= \int h(M_{1:T}^{1:G})h(G^{1:P})f(Z_t^p = k|M_{1:T}^{1:G}, G^{1:P})dM_{1:T}^{1:G}dG^{1:P} && (3.27)\\
&= \int h(M_t^{1:G})h(G^{1:P})f(Z_t^p = k|M_t^{1:G}, G^{1:P})dM_t^{1:G}dG^{1:P} && (3.28)\\
&= \sum_{g=1}^{G} \mathcal{C}_g^p \sum_{j=1}^{K} h(M_t^g = j)f(Z_t^p = k|M_t^g = j, G^p = g) && (3.29)
\end{aligned}$$

As illustrated at line (3.28), 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 (3.29) we find a more "algorithmic" expression. Note that the values of $f(Z_t^p = k|M_t^g = j, G^p = g)$ 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 (3.29).

We rewrite our lower bound (3.25) as

$$\sum_k \sum_p \sum_t \rho_t^p(k) \log f(Y_t^p|Z_t^p = k, \mu_k^p, \lambda_k^p) + \sum_k \sum_p \log f(\mu_k^p, \lambda_k^p) \tag{3.30}$$

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 (3.30) 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 μ, λ. We follow that procedure, adopting the notation from [Arc05]. This essentially amounts to decomposing $\mathcal{A}_1$ as

$$\sum_k \sum_p \sum_t \rho_t^p(k) \log \int \mathcal{N}\left(Y_t^p|\mu_k^p, u\lambda_k^p\right) \text{Gamma}\left(u|\frac{\nu_k^p}{2}, \frac{\nu_k^p}{2}\right) du \tag{3.31}$$

and maximizing instead the quantity

$$\mathcal{A}_1^* = \sum_k \sum_p \sum_t \rho_t^p(k) \log \mathcal{N}\left(Y_t^p|\mu_k^p, \bar{u}_t^p(k)\lambda_k^p\right) + \log \text{Gamma}\left(\bar{u}_t^p(k)|\frac{\nu_k^p}{2}, \frac{\nu_k^p}{2}\right). \tag{3.32}$$

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 2.2 to be

$$\mathcal{A}_2 = \sum_k \sum_p \log \mathcal{N}\left(\mu_k^p | m_k^p, \text{precision} = \lambda_k^p \eta_k^p\right) + \log \text{Gamma}\left(\lambda_k^p | \gamma_k^p, S_k^p\right). \tag{3.33}$$

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

$$\sum_k \sum_t \sum_p \rho_t^p(k) \left[\frac{1}{2}\log\left(\bar{u}_t^p(k)\lambda_k^p\right) - \frac{\bar{u}_t^p(k)\lambda_k^p}{2}\left(Y_t^p - \mu_k^p\right)^2 + \left(\frac{\nu_k^p}{2} - 1\right)\log \bar{u}_t^p(k) - \frac{\nu_k^p}{2}\bar{u}_t^p(k)\right] + \sum_k \sum_p \left[\frac{1}{2}\log\lambda_k^p - \frac{\eta_k^p \lambda_k^p}{2}\left(\mu_k^p - m_k^p\right)^2 + \left(\gamma_k^p - 1\right)\log\lambda_k^p - S_k^p \lambda_k^p\right]. \tag{3.34}$$

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

$$0 = \rho_t^p(k)\left[\frac{1}{2}\bar{u}_t^p(k)^{-1} - \frac{\lambda_k^p}{2}\left(Y_t^p - \mu_k^p\right)^2 + \left(\frac{\nu_k^p}{2} - 1\right)\bar{u}_t^p(k)^{-1} - \frac{\nu_k^p}{2}\right] \tag{3.35}$$

$$\bar{u}_t^p(k)^{-1}\left[\frac{1}{2} + \left(\frac{\nu_k^p}{2} - 1\right)\right] = \frac{\lambda_k^p}{2}\left(Y_t^p - \mu_k^p\right)^2 + \frac{\nu_k^p}{2} \tag{3.36}$$

$$\bar{u}_t^p(k) = \frac{\frac{1}{2} + \frac{\nu_k^p}{2} - \frac{2}{2}}{\frac{\lambda_k^p}{2}\left(Y_t^p - \mu_k^p\right)^2 + \frac{\nu_k^p}{2}} = \frac{\nu_k^p - 1}{\lambda_k^p\left(Y_t^p - \mu_k^p\right)^2 + \nu_k^p} \tag{3.37}$$

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

$$0 = \sum_t \rho_t^p(k)\left[-\frac{2}{2}\bar{u}_t^p(k)\lambda_k^p\left(Y_t^p - \mu_k^p\right)(-1)\right] - \frac{2\eta_k^p\lambda_k^p}{2}\left(\mu_k^p - m_k^p\right) \tag{3.38}$$

$$0 = \sum_t \rho_t^p(k)\,\bar{u}_t^p(k)\lambda_k^p Y_t^p + \eta_k^p\lambda_k^p m_k^p + \sum_t \rho_t^p(k)\,\bar{u}_t^p(k)\lambda_k^p(-1)\mu_k^p - \eta_k^p\lambda_k^p\mu_k^p \tag{3.39}$$

$$\mu_k^p = \frac{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k)\lambda_k^p Y_t^p + \eta_k^p\lambda_k^p m_k^p}{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k)\lambda_k^p + \eta_k^p\lambda_k^p} = \frac{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k)Y_t^p + \eta_k^p m_k^p}{\sum_t \rho_t^p(k)\,\bar{u}_t^p(k) + \eta_k^p} \tag{3.40}$$

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

$$0 = \sum_t \rho_t^p(k)\left[\frac{1}{2}(\lambda_k^p)^{-1} - \frac{\bar{u}_t^p(k)}{2}\left(Y_t^p - \mu_k^p\right)^2\right] + \frac{1}{2}(\lambda_k^p)^{-1} - \frac{\eta_k^p}{2}\left(\mu_k^p - m_k^p\right)^2 + \left(\gamma_k^p - 1\right)(\lambda_k^p)^{-1} - S_k^p \tag{3.41}$$

$$0 = (\lambda_k^p)^{-1} \left[\frac{1}{2} \sum_t \rho_t^p(k) + \frac{1}{2} + (\gamma_k^p - 1) \right] - \left[\sum_t \rho_t^p(k) (-\frac{1}{2}) \bar{u}_t^p(k) \left(Y_t^p - \mu_k^p \right)^2 - \frac{\eta_k^p}{2} \left(\mu_k^p - m_k^p \right)^2 - S_k^p \right] \tag{3.42}$$

$$(\lambda_k^p)^{-1} = \frac{\sum_t \rho_t^p(k) \bar{u}_t^p(k) \left(Y_t^p - \mu_k^p \right)^2 + \eta_k^p \left(\mu_k^p - m_k^p \right)^2 + 2S_k^p}{\sum_t \rho_t^p(k) + 2\gamma_k^p - 1} \tag{3.43}$$

We iterate a few times updating the quantities in the order (3.37), (3.40), (3.43) 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 θ.

3.4 Cluster initialization

Depending on the data given and the choice of hyperparameters, both *hmmmix-hard* and *hmmmix-soft* can be very sensitive to the initial clustering. When using artificial data sampled from the hmmmix model itself, *hmmmix-soft* performed quite well in terms of identifying the hidden chains without any need for a good initialization (i.e. using uniform random noise as initial assignments). The patients assignments usually converge to hard assignments even thought they are not required to do so. It is not uncommon with soft assignments to get duplicate clusters (the T-dimensional equivalent of two cluster centers at the same location), especially when using a number of clusters G larger than the true number of clusters. In those cases, we can identify the copies to merge them before computing the Jaccard coefficient.

More on the topic of cluster initialization is found in sections 4.1, 4.2 and 4.6 when we discuss particular applications.

3.5 Algorithm initialization and updating scheme

Our variational inference approach is about trying to find the best distributions $h(M_{1:T}^{1:G}), h(G^{1:P})$ and parameters $A^{1:G}, \pi^{1:G}, \mu_k^p, \lambda_{1:K}^{1:P}$ that maximize the full data log-likelihood (3.2). There are three main components to this algorithm :

- updating the hidden chains $h(M_{1:T}^{1:G})$ along with their associated parameters $A^{1:G}, \pi^{1:G}$
- updating the partial patients assignments $h(G^{1:P})$
- updating the parameters $\theta = (\mu_k^p, \lambda_{1:K}^{1:P})$

If we start with good initialization values for the patients clusterings (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 θ. In that particular case, we recommend doing a few

updates of $h(M_{1:T}^{1:G})$ and θ, alternating between them, before updating the partial patients assignments $h(G^{1:P})$. After that, we can proceed in any order and see what works best for our particular case. We have tried a few variants but they are roughly equivalent.

3.6 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 tricks to learn the parameters $\theta = \mu_{1:K}^{1:P}, \lambda_{1:K}^{1:P}$. We can get a good overview of all the approximations used in Figure 3.1.

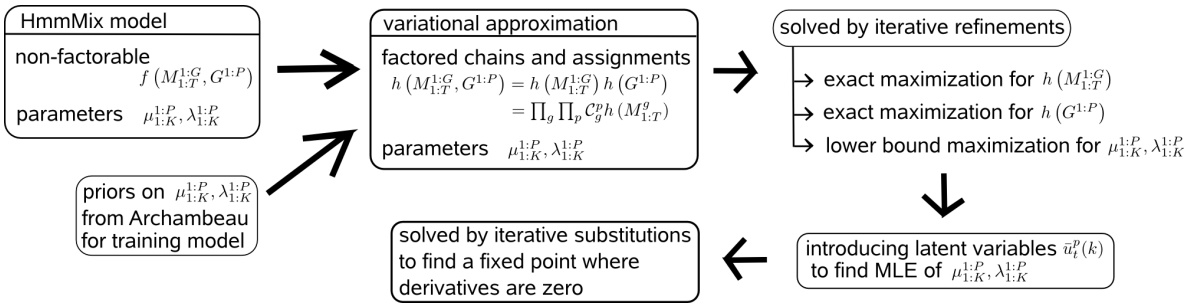


Figure 3.1: The relations between the different parts of the algorithm in terms of approximations.

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 3.3.3 were a maximum and not an minimum. However, we can always compute the value of either (3.24) or (3.25) 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 (3.24) is given by

```
sum(sum(log(sum(rho_KTP .* YgivenZ_KTP,1))))
```
(3.44)

and the lower bound (3.25) on (3.24) is given by

```
sum(sum(sum(rho_KTP .* log(YgivenZ_KTP)))).
```
(3.45)

If we trust the parameter updates, we can speed up the algorithm greatly by not computing the log-likelihood. As explained in section 4.4, 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 describe two 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 3.3.2.

4.1 Benchmark with synthetic data

In [Sha08] *hmmmix-hard* is compared to the current state-of-the-art methods using a benchmark test with artificial data. We reproduced their experiment 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 is the Jaccard coefficient. 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 4.1.

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

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

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.

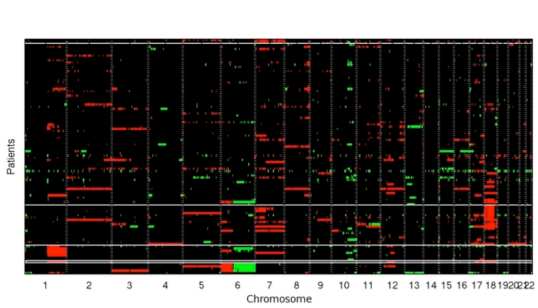
One of the difficulties that we encountered here was that the Jaccard coefficient does not care about the fact that a soft clustering might be desirable or even very informative. We merge identical clusters after running *hmmmix-soft* (or during execution), but some clusters might be relatively similar without being identical. They can also be merged using a tolerance threshold, and if we know the real number number of clusters, we merge clusters until we reach that number. The main problem, however, is the lower quality of local maxima that we reach when we don't start from a good initialization.

4.2 Comparison with hmmmix-hard on FL data

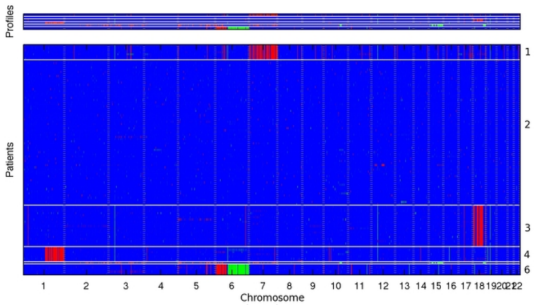
It is difficult to compare quantitatively the performance of *hmmmix-soft* and *hmmmix-hard* on real data because often there are no target values available as ground truth. Not only are we missing information, but the parameters from the models don't necessarily correspond to any real feature. One way 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 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 4.1.

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, but we generally get results that range from



(a) values of chains associated to patients after *hmmmix-hard*



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

Figure 4.1: The original results for *hmmmix-hard*. (a) the initial imputed values for $Z_{1:T}^{1:P}$. (b) the values after running *hmmmix-hard* to cluster the patients. The values displayed are those of the hidden chains to which each patient belongs. The patients are reordered for easier visualization. The green segments correspond to $M_t^g = high$ while red segments correspond to $M_t^g = low$.

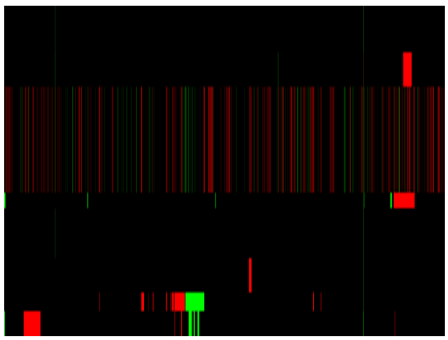
modeling noise too closely to being insensitive to the data. In the transition from one extreme to the other, though, the regions of interest are more or less consistent. We show in figure 4.2 three possible outcomes of *hmmmix-soft*. The closest match to the original is figure 4.2(a).

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. In all three cases, there seems to be certain groups with a good number of patients where we're not really capturing the essence of the data. Using smaller values of $\alpha_L, \alpha_N, \alpha_G$ gives less erratic values in problematic regions, but this weakens the relation between the group hidden chains $M_{1:T}^g$ and the chains $Z_{1:T}^p$. In the extreme case, setting $\alpha_L, \alpha_N, \alpha_G = (1, 1, 1)$ makes the algorithm ignore the data so we'd like to stay with high values whenever possible.

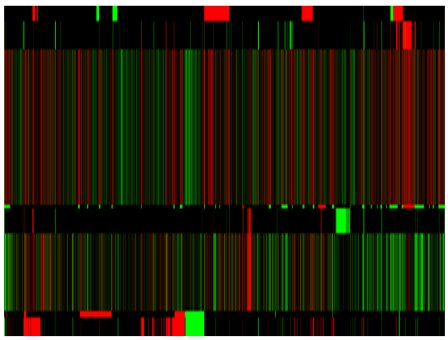
One of the goals of this thesis is to see what can be done with the extended *hmmmix* model once we can use partial patients assignments. For that reason, to get the results from figure 4.2 we didn't use the k-medoids initialization from [Sha08] but we decided instead to see if we could achieve similar results using the classic mixture of gaussians trained with EM.

The idea was to get a rough partial clustering that distributes 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 τ from section 3.3.2 (also discussed in section 4.6).

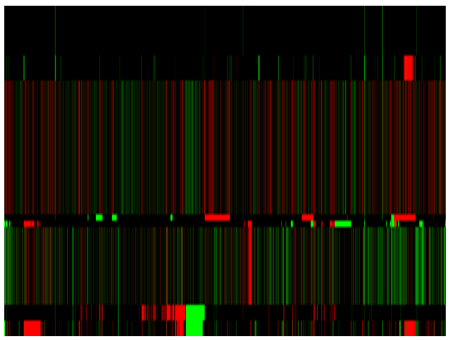
It's hard to justify any value for the scaling factor a priori, but we investigated values from different orders of magnitude and computed the entropy of the resulting EM clustering dis-



(a) $\alpha_L, \alpha_N, \alpha_G = (3, 3, 3)$



(b) $\alpha_L, \alpha_N, \alpha_G = (10, 5, 10)$



(c) $\alpha_L, \alpha_N, \alpha_G = (20, 10, 20)$

Figure 4.2: Three possible sets of imputed values for hidden chains and patients using *hmmmix-soft*. These are a good representation of the kind of results that we get. The choices of parameters $\alpha_L, \alpha_N, \alpha_G$ are shown here, but we refrained from showing the effects of the other free parameters as they are somewhat similar. These values are imputed by a pass of Viterbi because *hmmmix-soft* keeps full distributions $h(M_{1:T}^{1:G})$.

tributions (merging duplicate clusters). 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 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 also use 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.

4.3 Multiple chromosomes

The description of the *hmmmix* model was simplified a bit here to deal with only one chain for every group. When dealing with real data, we have data coming from 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 4.4 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 θ 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 just link the chromosomes one after the other as long as we keep track of the cuts when doing forwards-backwards on the sections.

4.4 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). It usually takes a bit less than 10 iterations (update M, update θ, update G, update θ, ...) to have patients assignments stabilize, but we can always pick a large value of τ for the assignment prior (from section 3.3.2) to slow down the algorithm and have it take around 50 such iterations.

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 :

- compute the values of $R_{t,g}[k]$ 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 $R_{t,g}[k]$: $\mathcal{O}\left(GTK^2\right)$
  - update the parameters A^g, π^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 $h(M^g_{1:T})$: $\mathcal{O}\left(GPTK\right)$
- compute $\rho^p_t\left(k\right)$: $\mathcal{O}\left(GPTK^2\right)$
- update the parameters θ by updating the following values 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 $\mathcal{O}\left(PTK\right)$ that would take *hmmmix-soft* $\mathcal{O}\left(GPTK\right)$ operations to do.

4.5 Scalability

As explained in section 4.4, 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. Without the memory constraints, this means that 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.

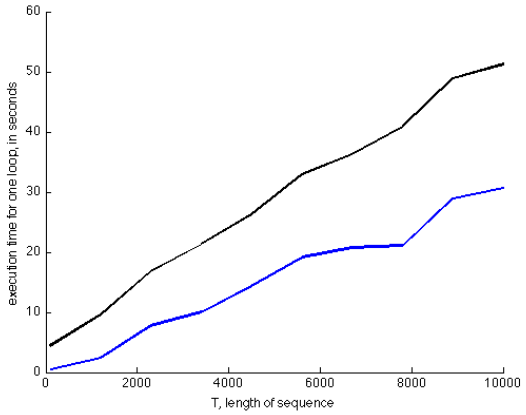
4.5.1 Hardware issues

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 4.3(a) and 4.3(b) we show in blue the execution time for one iteration of the main loop. We can see in the second plot 4.3(b) that the line breaks at $T = 80,000$ when Matlab ran out of memory.

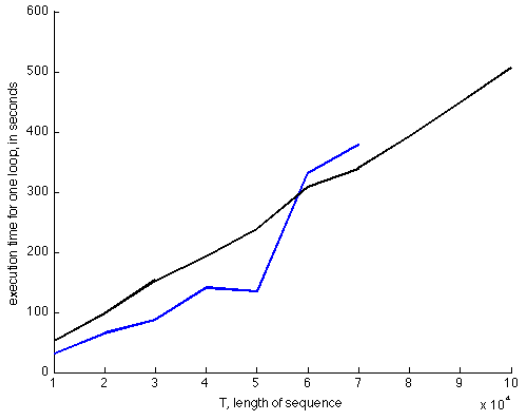
We ran more tests on a PC with a dual-core AMD 64-bit processor with 4 gigabytes of RAM and 12 gigabytes of swap memory. The goal was to see if the swap could be used to allow the algorithm to handle larger datasets. When running *hmmmix-soft* on a large dataset with $G = 10, P = 100, T = 250\ 000$, 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.

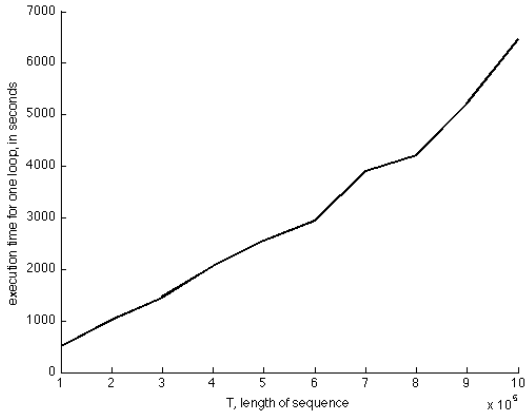
To compare the two approaches, we ran the low-memory implementation on the same Mac Mini. The results are again show in figure 4.3 in black. We can see in 4.3(a) how the disk I/O operations slow down the execution significantly for short sequences. It becomes more acceptable in 4.3(b) in the context of larger datasets, and it is absolutely necessary in 4.3(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



(a) short sequences



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



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

Figure 4.3: The execution time for the basic implementation of *hmmmix-soft* (in blue) compared to a low-memory implementation (in black).

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.

With figure 4.3 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.

4.5.2 Parallelization and other tricks

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 parallization 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. 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.

While not really part of the theoretical *hmmmix-soft* model, there is another contribution worth mentioning. Matlab has problems dealing with non-vectorized operations on arrays and it's impossible to vectorize 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. This has nothing to do with clever optimization techniques, but personal past experience taught that Matlab could be very inefficient in loops.

There are various places in the algorithm (e.g. during patient assignments) where we could 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.

4.6 Free energy variant

We haven't explained in section 3.3.2 how to choose the constant τ in the Dirichlet $(\tau, \ldots, \tau)$ prior on the parameters defining the distributions $h(G^p)$. 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 τ 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 θ) that the original likelihood had. A "safe" thing to try is to start at equilibrium with $\tau = 1$, let τ grow slowly, have the clusters move around and then come back to $\tau = 1$ hoping to get better final clusterings.

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 τ slowed the convergence but the same final hard assignments often resulted. A lot of the experiments were done with a random soft initialization of the patients to many clusters. It was of some use, but ultimately, the best results were with the use of k-medoids as initialization.

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 (3.13). 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 π^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.

One of the approaches tried (to get rid of the need for a good 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. We observed that smaller values of S (i.e. large τ) led to better results by having the clusters merge progressively instead of suddenly collapsing from 100 clusters to 13.

If we choose to scale the entropy terms by such factors, we have to take into consideration the length T of the sequences to select τ. We can see in (3.23) how the τ^{-1} factor is fighting against the sum $\sum_{t=1}^{T}$. A good value of τ when $T = 100$ might not be good when $T = 10^6$.

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.

4.7 Bayesian information criterion

The bayesian information criterion (BIC) is a popular way to do model selection , but there is a problem with it in this context.

We are using variational methods to maximize the log-likelihood of our data with respect to the parameter θ, but we really don't care much about θ itself and we have little use for the original model in which the hidden chains are marginalized. The quantities that are the most interesting are the imputed values for the hidden chains because they represent the biological reality that we are trying to model.

As noted in section 2.4, the silhouette coefficient isn't compatible with BIC, but the variational method by which we learn *hmmmix-soft* doesn't play too well with BIC either. Our variational quantities $h(M_{1:T}^{1:G})$ may be very informative about the copy number alterations in the data, but after using them to learn the parameters of the original distribution f in which the chains are marginalized, the only thing about the chains that we retain are the parameters $\pi^{1:G}, A^{1:G}$. These don't encode the locations of the special regions of interest identified by $h(M_{1:T}^{1:G})$ so they don't model the data too well. Adding a BIC term to a quantity that we don't really consider important is not a good way to pick the number of clusters.

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 mode more flexibility in terms of allowing partial assignments of patients to clusters and controling the weight that evidence has on the training process. Unfortunately, we didn't manage to beat the results of a good initialization with k-medoids, but we showed how we can get better results when k-medoids fails.

We compared the performance of our approach to that of [Sha08] on synthetic data using their benchmark test (we also reproduced their results with their code). We 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 tricks.

5.2 Future work

Our initial goal was to have a method that can completely do away with the need for good clustering initialization. Once we had an implementation of *hmmmix-soft* in hands, many elaborate methods were tried to accomplish this but they all came short of beating a k-medoids initialization that knows how many clusters it should identify. It might be the case that *hmmmix-soft* could perform better on another kind of data where k-medoids never works.

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.

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.

Bibliography

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

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

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

[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.

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

[Sha08] Sohrab Shah, *Model based approaches to array cgh data analysis*, Ph.D. thesis, 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), To appear.

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