Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Modelling of genomic annotations using discrete phase-type distributions

Bachelor Thesis

2024
Hana Derková

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Modelling of genomic annotations using discrete phase-type distributions

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	Mgr. Askar Gafurov, PhD.

Bratislava, 2024
Hana Derková

20225703

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Hana Derková
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Modelling of genomic annotations using discrete phase-type distributions
Modelovanie genomických anotácií pomocou diskrétnych distribúcií fázového typu

Anotácia: Genomické anotácie predstavujú jednotný spôsob kódovania informácií o častiach genómu. Príklady zahŕňajú umiestnenie génov, exónov, úsekov s nízkou zložitosťou, väzobne miesta proteínov, regulačné regióny, atď. Anotácia je zakódovaná ako sada intervalov pozdĺž genómu. Dostupnosť takýchto dát sa rýchlo zvýšila v posledných rokoch. Množstvo týchto dát vyvoláva potrebu efektívneho, ale zároveň verného modelovania.

Diskrétne distribúcie fázového typu vznikajú ako distribúcia absorpčného času Markovovho reťazca. Poskytujú tým prirodzený spôsob modelovania dĺžok intervalov a medzier v anotácii. Diskrétne distribúcie fázového typu sú schopné aproximovať ľubovoľné diskrétne distribúcie a taktiež umožňujú výpočtovo efektívne spracovanie vďaka vlastnostiam Markovových reťazcov.

Cieľ: Hlavným cieľom tejto práce je vytvoriť softvérový nástroj umožňujúci modelovanie dĺžok skutočných genomických anotácii pomocou diskrétnych distribúcií fázového typu. Vedľajším cieľom je aplikovať tento nástroj na rôzne genomické anotácie vo verejných databázach..

Vedúci: Mgr. Askar Gafurov, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 30.10.2023

Dátum schválenia: 30.10.2023 doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

20225703

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Hana Derková
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Modelling of genomic annotations using discrete phase-type distributions

Annotation: Genomic annotations are a unified way to encode information about parts of a genome. Examples include the location of genes, exons, low-complexity regions, protein binding sites, promoter regions, etc. An annotation is encoded as a set of intervals alongside the genome. The availability of such data has increased rapidly over the recent years. The amount of the data brings the necessity of an efficient yet faithful way of modelling of them.

Discrete phase-type distributions arise as a distribution of absorbing time of a Markov chain. This provides a natural way to model the interval and gap lengths in an annotation. The discrete phase-type distributions are able to approximate arbitrary discrete distributions, and also admit computationally efficient processing of them, thanks to the properties of Markov chains.

Aim: The primary goal of this thesis is to create a software tool allowing to fit discrete phase-type distributions to real genomic annotations. The secondary goal is to apply that tool to various genomic annotations in public databases.

Supervisor:	Mgr. Askar Gafurov, PhD.
Department:	FMFI.KI - Department of Computer Science
Head of department:	prof. RNDr. Martin Škoviera, PhD.
Assigned:	30.10.2023
Approved:	30.10.2023

doc. RNDr. Dana Pardubská, CSc.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Acknowledgments: Firstly, I would like to thank my supervisor Askar Gafurov for endless patience, guidance, advice, and help. Then I thank my whole family and friends for helping me get through. My special thanks belongs to my mom for technical and emotional support, my sister Katka for the food catering and, although sometimes questionable, mental support. And I can't forget my grandma, to whom I have promised to dedicate at least 3 pages as a gift. Then my running buddy Jitka for suffering through both running and my GitHub incompetence. And lastly, to Natali for firmly believing that I can do this, for regularly checking up on me, and for the emotional support.

Abstrakt

Genomická anotácia kóduje významné časti genómu ako súbory intervalov. Signifikantný prekryv týchto intervalov môže naznačovať biologické súvislosti. Táto práca stavia na predchádzajúcej metóde, ktorá používa dvojstavové Markovove reťazce na výpočet p-hodnôt pre tieto prekrytia. Dvojstavové Markovove reťazce, však generujú geometrické rozdelenie dĺžok intervalov aj medzier. Využitím diskrétnych distribúcií fázového typu, ktoré môžu aproximovať ľubovoľné diskrétne distribúcie, odstránime toto obmedzenie. V práci hodnotíme rôzne architektúry Markovových reťazcov z hľadiska výkonu a teoretickej sily. Súčasne vyvíjame softvérový nástroj na modelovanie reálnych genomických anotácií. Nakoniec začleníme absorpčné Markovove reťazce do pôvodného softvéru na výpočet p-hodnôt pre prekrývajúce sa genomické anotácie.

Kľúčové slová: Markovovský reťazec, absorbčný Markovovský reťazec, diskrétna distribúcia fázového typu, genomická anotácia

Abstract

Genomic annotation encodes significant parts of the genome as sets of intervals. Significant overlap of these intervals, can suggest biological connections. This thesis builds upon a previous method that uses two-state Markov chains to compute p-values for these overlaps. However, two-state Markov chain generates geometric a distribution of interval and gap lengths. By employing discrete phase-type distributions, which can approximate arbitrary discrete distributions, we mitigate these limitations. We evaluate various Markov chain architectures for performance and expressive power. We develop a software tool to fit these distributions to real genomic data. Finally, we incorporate absorbing Markov chains into the original tool for computing the p-values for overlapping genome annotations.

Keywords: Markov chain, absorbing Markov chain, discrete phase-type distribution, genomic annotation

Contents

List of Figures

List of Tables

Introduction

Genomic annotation provides a unified way to encode significant parts of a genome alongside the genome sequence. Mathematically, annotation is represented as a set of intervals. If two annotations resemble each other, it may indicate an underlying biological connection and suggest further research focus.

We need a statistical measure to determine whether observed annotation overlaps occurred by pure chance or truly indicate a biological connection. Such a measure is the p-value under a suitable null hypothesis.

This thesis builds upon the article *Markov chains improve the significance computation of overlapping genome annotations*, where the authors propose a null hypothesis that one of the annotations is generated by a two-state Markov chain. The properties of Markov chains then allow for the analytical computation of the corresponding p-value. However, a two-state Markov chain generates a geometric distribution of interval and gap lengths, which does not accurately model some annotations, thereby oversimplifying them.

The discrete phase-type distribution represents the distribution of absorption times of an absorbing Markov chain. Discrete phase-type distributions can approximate arbitrary discrete distributions, providing a natural way to model the interval and gap lengths in an annotation.

The goal of this thesis is to develop a software tool to fit discrete phase-type distributions to real genomic annotations.

Firstly, in chapter 1 we introduce terminology and mathematical background. In chapters 2 and 3 we examine different Markov chain architectures to determine which is most suitable for this task. We evaluate their performance and expressive power and present techniques we developed to make the training process more effective. We then show the results of fitting real genomic data in chapter 4. Lastly, in chapter 5 we incorporate our phase-type distribution into the original tool called MCDP from the *Markov chains improve the significance computation of overlapping genome annotations* article. Chapter 6 contains implementation details.

Chapter 1

Preliminaries

In this chapter we define the concept of genomic annotation and explain the need for software tool that can model them accurately. Given the central importance role of Markov chains in our project, we will properly define terms, Markov chain, absorbing Markov chain, discrete phase-type distributions, and discuss its properties. Additionally, we will offer an overview of existing research concerning the modeling of genomic annotations and statistical testing in this domain.

1.1 Biological context and motivation

1.1.1 Basic Terminology

Firstly, we establish the basic concepts and terminology essential for understanding genomic annotation problem. All the following definitions are from [28, 3].

Deoxyribonucleic acid, or *DNA*, is an organic molecule responsible for encoding genetic information. The molecule consists of two strands that form a double helix. Each strand consists of sugar (deoxyribose) and phosphate groups together forming a backbone. Attached along the backbone are *nucleobases*, nucleic acids: adenine (A), guanine (G), cytosine (C), and thymine (T). The sequence of nucleobases encodes genetic information, i.e. protein or RNA sequences.

Chromosomes are thread-like structures made of single molecule of DNA and associated proteins. The chromosomes reside in the nucleus of the cell and act as carriers of the genetic information encoded in DNA.

A *gene* is a distinct sequence of nucleobases within a DNA molecule that serves as the basic unit of inheritance, since they specify physical or biological traits. Genes contain the instructions for synthesizing specific proteins or protein segments. Approximately $20,000$ protein-coding genes are present in the human genome.

A *genome* is the complete set of genetic material present in an organism. It includes all of the organism's genes, as well as non-coding sequences of DNA. The exact sequence

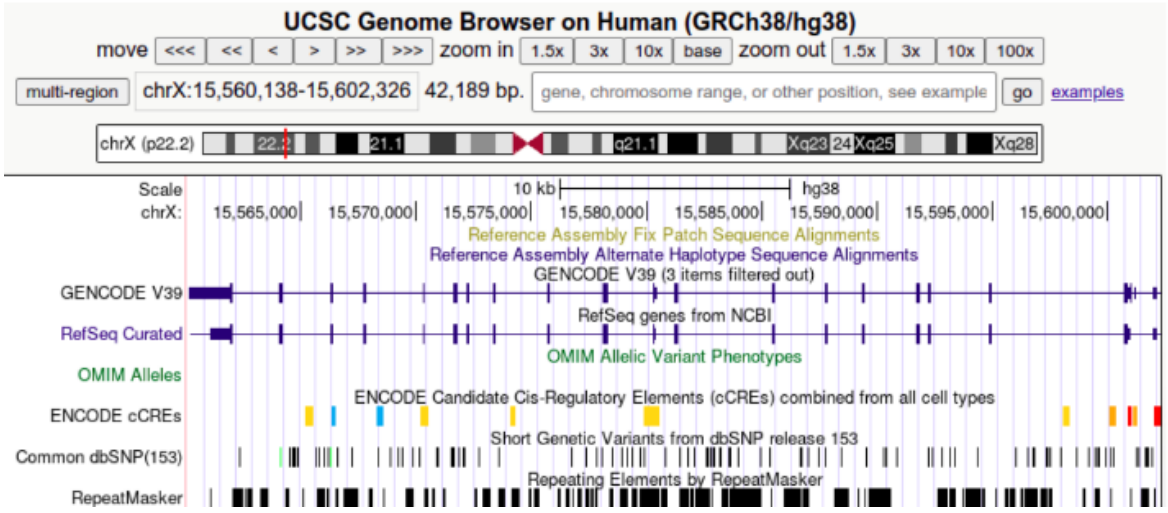

Figure 1.1: Example of a genome annotations (source: UCSC Genome Browser, `http://genome.ucsc.edu`)

of bases, genetic code, is unique for each organism. Determining this exact order of bases is called *sequencing*.

1.1.2 Genomic annotations

A genomic annotation is a unified way to represent parts of a genome, sharing particular property or function. As examples we can use low-complexity regions, protein binding sites, gene locations or coding regions.

Annotations are encoded as intervals along genome sequence (see Figure 1.1). From the mathematical point of view, an annotation can be represented either by a set of non-overlapping intervals or a sequence of 0 and 1 indicating on which genomic positions the annotation occurs.

The degree of overlap between two annotations could indicate underlying biological connection and therefore could suggest further biological research focus. The determination of the significance of such overlaps is sometimes referred to as *colocalization analysis*. For example, if one annotation contains nucleosomes with H3K4Me3 histone modifications, another contains known promoter regions, we can count the overlap between these regions. A significant number of overlapping regions implies a potential role for H3K4Me3 modifications in the gene transcription regulation, which has been indeed proven in [11].

However, on a scale of the human genome, some overlaps are due to appear simply by random chance. Therefore a statistical model that determines the significance of observed overlap is needed.

1.2 Statistical testing

There are many ways to create a probabilistic model in order to assign statistical significance of the observed overlaps. Given two annotations *reference* and *query*, we can either measure the *overlap statistics*, which is counting the number of intervals in reference that overlap with intervals in query, the other approach is the *the shared bases statistics*, counting the number of bases both reference and the query cover [8].

Since certain overlaps or shared bases are bound to occur purely by random chance, we need a metric to determine if the observed overlap is of any statistical significance. Such metric is a *P-value* under a suitable *null hypothesis*.

The *null hypothesis* H_0 proposes that there's no underlying relationship between query and reference annotations. If the null hypothesis holds true, any observed overlaps did occur by random chance and do not pose a statistical significance [14]. The *P-value* is the probability of observing the number of overlaps, at least as extreme as the observed value under the null hypothesis.

There are two general ways to compute a P-value. The first is an analytical approach, where the P-value is computed using a formula. The second is a sampling approach, where we randomly sample annotations from the distribution prescribed by the chosen null hypothesis and estimate the P-value by counting the number of samples with a test statistic more extreme than the observed one.

1.2.1 Null hypotheses for the colocalization analysis

The choice of the null hypothesis is not inherently clear, and different options lead to different computational challenges, impacting both time and memory consumption. Essentially, we need to define what constitutes a "random" annotation. This requires creating a random generator for query annotations where the generated annotations are similar to the original input query annotation but still randomized.

Permutation test

One popular way to define a null hypothesis is to assume that the query annotation is chosen uniformly at random from the set of all possible rearrangements of the intervals in the input query annotation. This approach is referred to as *permutation test* and it has been implemented in [9] and [13]. It has been shown in [7] that the computation of P-value for common test statistics under this null hypothesis is NP-hard.

In [9] authors implement *R/Bioconductor package* a permutation test framework designed to work with genome regions.

In [13] authors randomly place intervals alongside a given sequence accessible for the simulation. The overlap of each simulation is recorded and the average from each

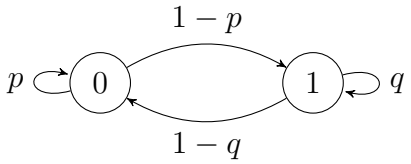

Figure 1.2: Two state Markov chain

simulation serves as an empirical P-value.

Defining a null hypothesis against randomly permuted intervals yields accurate outcomes only within a restricted space of all potential random samples. In human-sized genomes, naive sampling methods are computationally inefficient [25].

Null hypothesis based on Markov chains - MCDP

In articles [8] and [7], the authors proposed a novel null hypothesis based on Markov chains (see section 1.3 for detailed definitions). This new formulation admits an efficient algorithm to compute the P-value and also approximates the P-values obtained from the often used permutation null hypothesis.

Consider a Markov chain with two states, labeled zero and one respectively (see Figure 1.2). State 0 represents "outside" of the interval, state 1 represents "inside" of the interval. Such a Markov chain would generate a sequence of zeros and ones, which represents an annotation query (see subsection 1.1.2).

The transition weights of the Markov chain are chosen in such a manner that the expected interval and gap lengths in a generated annotation match the corresponding average lengths in the input annotation. This formulation allowed the authors to propose an exact quadratic time algorithm for the computation of the P-value, which was exploiting the properties of the Markov chains and the matrix representations. For details of this algorithm see [7].

The Markov chain used in this null hypothesis forces the intervals and gaps in a generated annotation to have the *geometric distribution* of lengths. However, some real-life annotations don't follow that distribution (see the datasets in Chapter 4 for examples). Thus, the Markov chain null hypothesis does not always model the input annotations faithfully.

In this thesis, our goal is to extend the Markov chain null hypothesis in a way which will model the distribution of interval and gap lengths more closely to the input data.

1.3 Markov chains

In this section, we provide a concise summary of general properties of Markov chains and absorbing Markov chains, which we will use in our work. This section can be

skipped if the reader already has experience in that area. The material in this section is based on [5, 29, 23].

Stochastic process is a sequence of random variables, where the sequence can be interpreted as time steps [24].

Let I be a finite or countable set, where each $i \in I$ is called a state, and I is called the state-space. We consider a stochastic process:

$$\{X^{(n)}, n = 0, 1, 2, \ldots\}$$

Suppose there is a fixed probability P_{ij} independent of time such that

$$P(X^{(n+1)} = i \,|\, X^{(n)} = j, X^{(n-1)} = i_{(n-1)}, \ldots, X^{(0)} = i_0) = P_{ij}, \quad n \geq 0$$

where $i, j, i_0, i_1, \ldots, i_{n-1} \in I$. Then this is called a *Markov chain process.*

One can interpret the above probability as follows: the conditional distribution of any future state $X^{(n+1)}$ given the past states $X^{(0)}, X^{(1)}, \ldots, X^{(n-1)}$ and present state $X^{(n)}$, is independent of the past states and depends on the present state only.

The probability P_{ij} represents the probability that the process will make a transition to state i given that currently the process is in state j. Clearly one has:

$$\sum_{i=0}^{\infty} P_{ij} = 1$$

We call the matrix P containing the transition probabilities P_{ij} the *transition probability matrix.* There is one-to-one correspondence between transition matrices P and state automata with states I and probability of transition between states on corresponding edges. The steps of a corresponding automaton are often thought of as moments in time. The *initial state distribution* $\overline{\pi}$ is the probability distribution of the Markov chain at time 0. In this text we consider only the initial state distribution $\overline{\pi} = (1, 0, \ldots, 0)$, hence at time 0 our Markov chain "starts" in the first state.

We can model the probability of being in a particular state in a particular moment of time simply by simulating our automaton, and transitioning from states with assigned probability [5].

For a given Markov chain with n states $I = \{0, 1, \ldots n-1\}$, transition matrix P and initial distribution $\overline{\pi} = (\pi_0, \ldots, \pi_{n-1})$, the distribution of $X^{(k)}$ is given by:

$$(P(X^{(k)} = 0), P(X^{(k)} = 1), \ldots, P(X^{(k)} = n-1)) = \overline{\pi} P^k.$$

This definition was inspired by and the full proof by induction can be found in [29]. In other words we can compute the distribution of $X^{(n)}$ by matrix multiplication, multiplying the initial vector $\overline{\pi}$ and matrix P^n to the power of n.

P^n can be computed using $O(\log n)$ matrix multiplications, using the *multiplication by square* algorithm. In this method, we decompose n into the sum of powers of two,

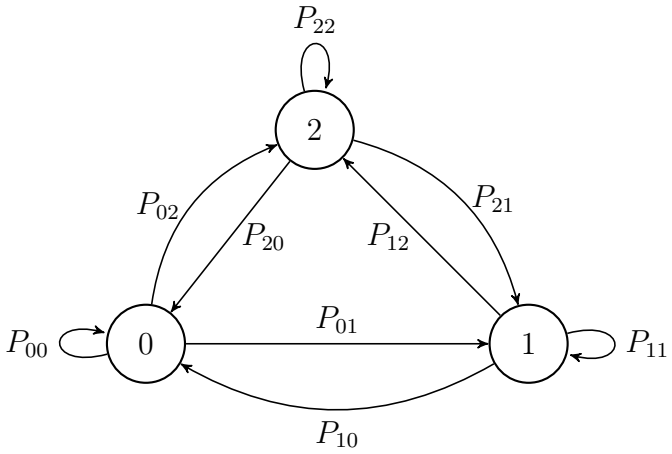

Figure 1.3: Three state Markov chain for transition matrix P

such that $n = x_1 \cdot 2^1 + x_2 \cdot 2^2 + \cdots + x_k \cdot 2^k$. Sub-products of P to the power of $2^1, 2^2, \ldots, 2^k$, are stored in memory and can be computed using a dynamic programming table. Subsequently, the result matrix P^n is computed by multiplying these sub-products. Example of a state automaton in Figure 1.3 for state space $I = \{0, 1, 2\}$ and transition matrix:

$$P = \begin{bmatrix} P_{00} & P_{01} & P_{02} \\ P_{10} & P_{11} & P_{12} \\ P_{20} & P_{21} & P_{22} \end{bmatrix}$$

1.3.1 Absorbing Markov chain. Mean and variance of the absorption time.

An *absorbing state* is a state from where the Markov chain cannot escape. State $i \in I$ is absorbing if :

$$\sum_{i \neq j} p_{ij} = 0$$

States that are not absorbing are called *transient states*. *Absorbing Markov chain* (AMC) is a Markov chain in which every state can reach an absorbing state. A discrete random variable denoting the number of steps taken before reaching one of the absorbing states is called *the absorption time*.

We are now interested in the mean value and the variance of the absorption time for a given absorbing Markov chain. Those values can be computed efficiently using the notion of a *fundamental matrix* of an absorbing Markov chain.

Consider an absorbing Markov chain with r absorbing and t transient states, transition matrix P and initial state distribution $\overline{\pi}$.

We say that an absorbing Markov chain is in *normal form*, if the rows of transition matrix P corresponding to the transient states appear sooner that the absorbing states.

Note that any absorbing Markov chain can be *normalized* by permuting the rows and columns of the transition matrix and the initial state distribution vector. Transition matrix P of an AMC in normal form can be written as the following block matrix:

$$P = \begin{bmatrix} Q & R \\ \mathbf{0}_{r\times t} & I_r \end{bmatrix},$$

where $Q \in \mathbf{R}^{t\times t}$ is the matrix of transition probabilities between transient states, $R \in \mathbf{R}^{t\times r}$ is the matrix representing transitions from the transient states to the absorbing states, $\mathbf{0}_{r\times t}$ is a zero matrix of size $r \times t$, and I_r is an identity matrix of size $r \times t$. Using the matrix Q, the *fundamental matrix* N of an absorbing Markov chain in normal form is defined as the following geometric series:

$$N := \sum_{k=0}^{\infty} Q^k \overset{*}{=} (I - Q)^{-1},$$

where I is an identity matrix of corresponding size. It can be shown that the equality marked with an asterisk $(*)$ is always true, i.e. that geometric series always has a limit. Element $N_{i,j}$ of fundamental matrix N corresponds to the *expected number of times* the process started at state i *visits state* j before the absorption.

Let's denote the absorption time when starting in state i as Y_i. Using the fundamental matrix, we can compute $E[Y_i]$ as a sum of the i-th row of the fundamental matrix (since each step is a *visit* of a transient state). Let's denote the column vector of mean absorption times as $\overline{e} := \begin{pmatrix} E[Y_1] & E_[Y_2] & \dots & E[Y_t] \end{pmatrix}^T$. We can then compute it in matrix form as follows:

$$\overline{e} = N \cdot \overline{\mathbf{1}}_t,$$

where $\overline{\mathbf{1}}_t$ is a column vector of size t with values 1 in each position.

The similarly defined vector $\overline{v} := \begin{pmatrix} Var[Y_1] & Var[Y_2] & \dots & Var[Y_t] \end{pmatrix}^T$ of the variance of the absorption time starting in state i can be computed as follows:

$$\overline{v} = (2N - I) \cdot \overline{e} - \overline{e} \circ \overline{e},$$

where $\overline{e} \circ \overline{e} = \begin{pmatrix} e_1^2 & e_2^2 & \dots & e_n^2 \end{pmatrix}^T$ is the Hadamard product of vector $\overline{e}$ with itself [10, 19].

1.4 Absorption time as a data distribution in MCDP model

The class of distributions arising from the absorption times in finite discrete absorbing Markov chains are called *discrete phase-type distributions* (or *DPT* for short). The simplest examples of distributions belonging to that class are *geometric* and *negative*

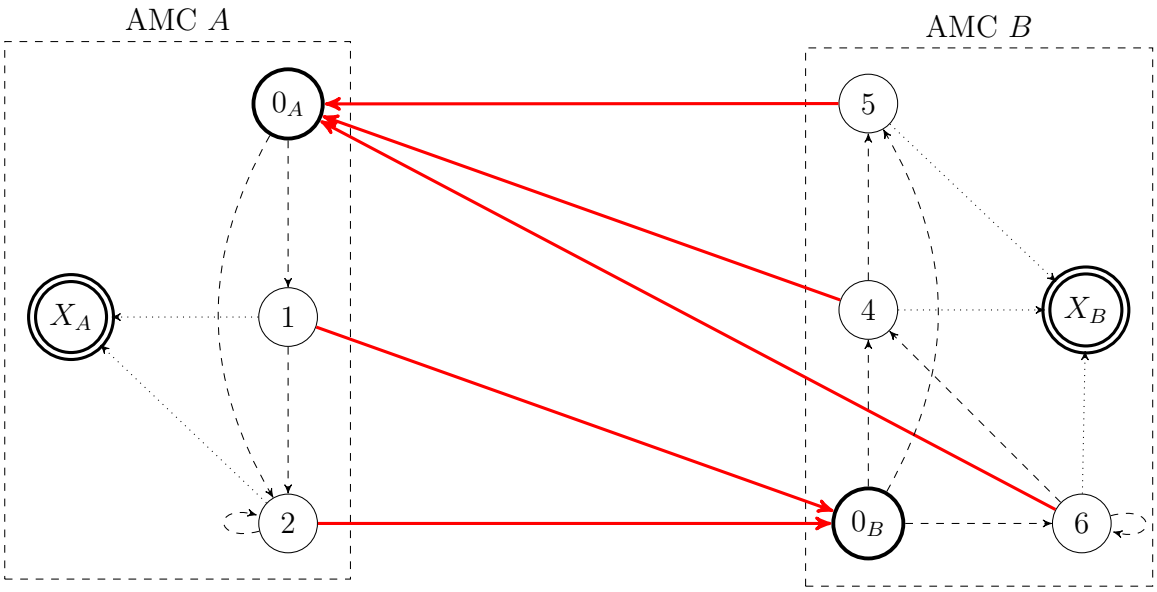

Figure 1.4: Visualization of the concatenation of two AMCs. States 0_A and 0_B, being initial states, and X_A and X_B being absorbing states, in AMC A and AMC B, respectively. The absorbing transitions from states 1 and 2 have been redirected to 0_B, similarly for states 4, 5 and 6 in AMC B.

binomial distributions. However, with enough states in the underlying Markov chain, a discrete phase-type distribution can approximate any discrete distribution with arbitrary precision [1]. This allows us to essentially "encode" any probability distribution as a Markov chain with an absorbing state.

Recall that in the Markov chain null hypothesis, the authors have modelled the genomic annotations using a two-state Markov chain (see subsection 1.2.1), where state 0 produces "gap" positions, and state 1 produces "interval" positions. One can view the state 0 as a two-state absorbing Markov chain with state 1 being the absorbing state, and vice versa. Essentially, the gap lengths and the interval lengths in an annotation are generated using two discrete phase-type distributions, each operating with just one transient state, thus limiting the respective DPT distributions to be geometric. Our goal is to use DPT distribution with more states, thus fitting the input annotation more closely.

The scheme works as follows. We replace the two states with two absorbing Markov chains (AMCs): one representing the "inside" of the interval and the other representing the "outside". We connect the AMCs by redirecting the transition probabilities of one AMC to the initial state of the other, and vice versa (see Figure 1.4). We generate the annotation similarly to the previous case: states from the "inside" AMC generate 1, and states from the "outside" AMC generate 0.

For example sequence of states $(0_A, 1, 2, 2, 0_B, 5, 0_A, 2)$ form Figure 1.4, would generate annotation $(0, 0, 0, 0, 1, 1, 0, 0)$.

The main challenge of this approach comes from the necessity to fit two discrete phase-type distributions to the input interval and gap lengths distribution, respectively. Whereas in the original model, the fitting procedure consists of simply computing the average length, now we have to choose all transition weights in the corresponding absorbing Markov chains. There is no unified analytical algorithm for that, thus we have to rely on general optimisation algorithms.

We have found one existing tool called *PhFit* [15] design to fit DPT distributions to arbitrary input data. However, we were not able to find any implementation of that software available for downloading. In [17], the authors conducted a series of training on different standard distributions. They chose to optimize also the initial state distribution, whereas in our case, it is fixed. Both methods [17, 15] use a similar approach to fit the DPT distribution by defining a loss function and optimizing parameters based on that function. The main differences lie in the choice of loss function, the optimization method, and the parameters selected for optimization —some approaches restrict to a specific AMC structure, while others also optimize the initial state vector.

Given these constraints and the lack of available tools, we concluded that we need to implement the fitting of DPT distributions ourselves.

1.4.1 Model

Architecture of Markov chain refers to the structure of its finite state automata, specifically the transitions that are allowed to be non-zero. To absorbing Markov chain with fixed architecture and number of states we will refer to as *model*. We will refer to the transition probabilities as *model parameters* and we will denote them θ. Our model is represented as a transition matrix P, it is clear that the character of the transition matrix P also implies specific architecture. We are provided a dataset of annotation lengths, we will refer to this dataset simply as *data*. It is useful to note that *data probability distribution* is discrete, since data are integer lengths.

We aim to create a model and calculate parameters, such that the absorption time distribution of our model provides the best approximation to the *data probability distribution*.

1.4.2 Goal - minimizing cross entropy

Our goal is to maximize the probability that our model will generate specific data given parameters vector θ.

$$P(data|\theta)$$

We are searching for optimal parameter values θ^*, that maximizes this probability.

$$\theta^* = \arg\max_{\theta} P(\text{data}|\theta)$$

Using the product rule we can rewrite this as :

$$\theta^* = \arg\max_{\theta} \prod_{d \in data} P(d|\theta)$$

We will shift this to logarithmic space, a commonly used technique to simplify the computations. This approach offers faster addition compared to multiplication and helps maintain numerical precision, particularly when dealing with very small probabilities. Since logarithmic function in monotonous this will not affect values of θ^*.

$$\theta^* = \arg\max_{\theta} \sum_{d \in data} \ln P(d|\theta)$$

We will refer to the probability $\sum_{d \in data} \ln P(d|\theta)$ as *log-likelihood*. Now for computation sake, we reformulate the problem form maximizing the *log-likelihood* to minimizing the *negative log-likelihood*: $-\sum_{d \in data} \ln P(d|\theta)$. We can clearly see that these formulations are interchangeable.

Minimizing the *negative log-likelihood* is equivalent to minimizing the *cross-entropy* between *phase-type distribution* of our model and *data distribution*. The *cross-entropy* $H(p, q)$ of two discrete distributions p, q is defined as :

$$H(p, q) = -\sum_{x \in X} p(x) \log q(x)$$

So in the end our goal is to find optimal parameter values θ^*, such that they minimize the *cross-entropy* of *phase-type distribution* of our model and *data distribution*.

Shannon entropy

In [6] *Shannon entropy* of finite probability distribution p is defined as:

$$H(p) = -\sum_{i} p_i \log p_i$$

In our case p is the *data* distribution. If our modelled *phase-type distribution* would model *data* 100% accurately, the resulting *cross-entropy* would be exactly *Shannon entropy*. Therefore we will use *Shannon entropy* as a comparison to our achieved *cross-entropy* to determine how accurate our fit is, as *Shannon entropy* is a lower bound of possible achieved cross-entropy.

1.4.3 Softmax function

To ensure that *model parameters* "going out" from the transient state, sum up to 1, we use *Softmax function*. This is a common approach used also in Neural networks

training. Given a k dimensional vector parameter $\gamma = (\gamma_1, \gamma^2, \ldots, \gamma_k)$, Softmax function returns k dimensional vector $\overline{\gamma}$ normalized to a probability distribution, in other words such that vector elements sum up to 1.

$$\overline{\gamma_i} = \frac{exp(\gamma_i)}{\sum_j^k exp(\gamma_j)}$$

1.4.4 Model training

In code transition matrix P and initial probabilities vector $\overline{\gamma}$ are represented as objects from the numpy library [12]. We define a function *Markov chain to likelihood*, that for given parameters, number of states, data and architecture returns two vectors. *likelihoods* and *data likelihoods*. These likelihoods are simply commuted by multiplying the initial vector and transition matrix up to the power maximal data; we use multiplication by square to speed up the matrix multiplication. We define *objective function* which computes *negative logarithmic likelihood* for given parameters, data and number of states, normalized by the number of data, this is done by calling the *Markov chain to likelihood function* for given input and then summing the *data likelihood vector* in logarithmic space and then dividing it by length of data vector.

Then we use minimize function from scipy.optimize to find the optimal parameters θ^* that minimizes the *objective function*. We will refer to this process as *model training* and to transition probabilities we want to optimize as parameters for optimization. Later we compare the phase-type distribution of our model and the data distribution our model was trained on.

1.4.5 L-BFGS-B

For optimization we used the L-BFGS-B algorithm. L-BFGS-B is a limited memory, quasi-Newton algorithm, used for large nonlinear optimization with simple bound constraints on variables. This algorithm does not require knowledge of the objective function, or its Hessian.

At the beginning of each iteration, an approximation of inverse Hessian is updated, function values from previous iterations are used for this. Hessian is then used as an approximate quadratic model of the objective function. This model is then minimized and the vector direction, from current iterate to the approximate minimum, is used to determine the step direction. Then linear search alongside the step direction is performed, to determine the step size [33, 4].

The Optimizer Parameters and Stopping Conditions

One of the user-defined parameters in the L-BFGS-B optimization algorithm is maxcor. This parameter sets the maximum number of variable metric corrections, for construct-

ing the inverse Hessian matrix. In all of our experimental setups, the value of maxcor was set to 10.

Other user defined parameters, occur in optimization stopping conditions:

$$\frac{f^k - f^{k+1}}{\max\{|f^k|, |f^{k+1}|, 1\}} \leq \text{factr} \times \text{eps}$$

Where eps is the machine precision, and factr is a parameter controlled by the user. This is designed to terminate the optimization when change of the function value between iteration is sufficiently small.

$$\frac{f^k - f^{k+1}}{\max\{|f^k|, |f^{k+1}|, 1\}} \leq \text{ftol}$$

This is designed to terminate the optimization when change of the function value between iteration is smaller than ftol, where ftol is parameter controlled by user [33, 4].

$$\max\{|\text{proj}\, g_i| \, i = 1, \ldots, n\} \leq \text{gtol}$$

Projected gradient is the projection of the gradient vector onto the space defined by the variable bounds. It must be equal to 0 at the local minimizer. This test is designed to terminate the optimization run, where the norm of projected gradient is smaller than gtol, where gtol is a parameter controlled by the user.

Additionally, optimization stops after reaching a specified number of iterations – maxiter, also a parameter controlled by a user.

In all our experiments, we set these constants as follows: ftol : 1×10^{-6}, gtol : 1×10^{-6} and maxiter : 15000.

Chapter 2

Basic Markov chains architectures

In this chapter we will describe the early stages of the development of the Markov chain architectures. We will introduce three architecture models: *self-loop architecture*, *early-escape architecture* and *combined architecture*. We will evaluate each architecture on a set of synthetic datasets, and discuss advantages and disadvantages of them.

We aim to present the development process in chronological order, thus we start from the most simple architecture and continue to the more complex ones. Along the way, we address their weak spots and training problems as they arise, and we introduce solutions and developed tools in respect to the order we developed them.

We begin by outlining the architecture criteria and subsequently introduce synthetic challenges. Following, we define the self-loop architecture, discussing its properties and demonstrating its performance on challenges. Then, we introduce the early-escape architecture, discussing its properties and showcasing its performance on challenges. Afterwards, we define the combined architecture and illustrate its performance on synthetic challenges. Finally, we introduce a technique called *mean shifting* aimed at improving the architecture performance.

2.1 Architecture selection criteria

There are two general choices to make when fitting a Markov chain to a given distribution: the number of states and the selection of (non-zero probability) transitions between them. We will refer to that selection of transitions as *the architecture* of a Markov chain, regardless of a specific number of states. We will refer to the proportion of non-zero transitions in a Markov chain as its *density*.

Those choices affect several key factors. The most fundamental of them is the *theoretical expressive power* of the resulting Markov chain, i.e. its ability to match different input distributions with a right set of transition probabilities. The denser the Markov chain, the higher expressive power it has. An increase in the number of states

Choice	Expressive power	Training speed	Ability to train	Usage speed
Number of states ↑	↑	↓	?	↓
Architecture density ↑	↑	↓	↓	-

Table 2.1: The summary of the impacts of parameter selection. An up arrow (↑) (down arrow (↓)) means that this factor improves (worsens) with an increase in the according criterion. A dash (-) means there is no effect. A question mark (?) means that the effect is unclear.

also generally leads to the higher expressive power and therefore lower cross-entropy (we will show some exceptions later in this chapter).

The second key factor is the *running time of the training process*, i.e. the numerical minimisation of the cross-entropy between the input data and the absorption time distribution. The running time increases with both the number of states and the architecture density. The computational time or phase-type distribution grows cubic from the number of states.

Another factor to consider is the *ability to train*, i.e. optimizes capability to find the optimal solution. Complex architectures — with high architecture density, would overparametrize the model and therefore the optimizer might not be able to find optimal weights.

Consequently, the practical quality of the resulting fit is influenced by the number of states and architecture density, as well as numerical precision, including the range of parameters in optimisation. The effects are summarized in Table 2.1.

In conclusion we aim to find the *simplest architecture* with the *least possible number of states* that is still capable of modelling given a challenge, and only when it fails, we should search for a more complicated model.

2.2 Synthetic data challenges: Architecture testing

In order to assess the strengths and weaknesses of the proposed architectures, we selected four synthetic datasets. We will refer to them as *challenges*. These challenges are visualized in Figure 2.1.

The simplest challenge is to fit a single "peak" of small data with no gaps. We use a Gaussian distribution (rounded up) with a mean of 20 and a standard deviation of 5, comprising 500 samples.

Next, we increase the data magnitude while maintaining the simplicity of fitting a single "peak". We utilize a Gaussian distribution with a mean of 350 and a standard deviation of 7, also with 500 samples.

Subsequently, we examine the model's capability of modeling bimodal data, which

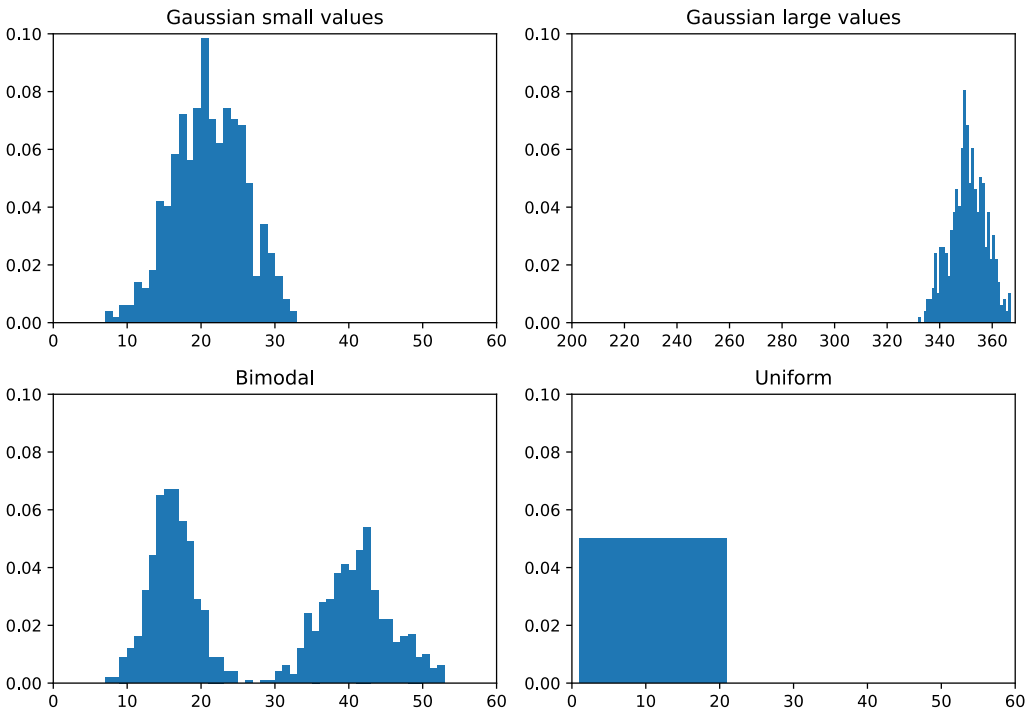

Figure 2.1: Synthetic datasets (challenges): Gaussian small values (top left), Gaussian large values (top right), bimodal (bottom left) and uniform (bottom right).

involves data with a "gap", not departing from the simplicity of small values. We employ two Gaussian distributions with means of 15 and 40 respectively, and standard deviations of 3 and 5 respectively, each consisting of 500 samples.

Lastly, we explore the accuracy of the model in fitting specific probabilities. For this purpose, we employ a uniform distribution, comprising 20 samples ranging from 1 to 20.

2.3 Self-loop architecture

Consider a Markov chain where each state has two (non-zero probability) outgoing arrows: one to itself (creating a *self-loop*) and another to the next state (see Figure 2.2). We will call such configuration a *self-loop architecture*. This model also appears in [17].

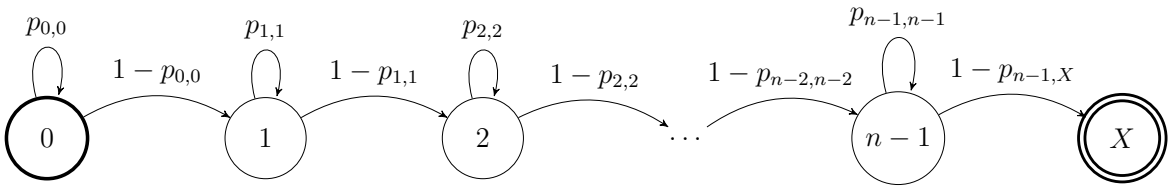

Figure 2.2: The self-loop architecture. State 0 is the initial state, and state X is the absorption state.

This idea is formalized as follows. Consider an absorbing Markov chain with finite state space I, one absorbing state $X \in I$ and transition matrix P where $p_{i,i} + p_{i,i+1} = 1$ for all $i \neq X$. As a convention, the last row of P will contain probabilities for absorbing state X.

$$P = \begin{bmatrix} p_{1,1} & 1-p_{1,1} & 0 & \dots & 0 \\ 0 & p_{2,2} & 1-p_{2,2} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & p_{n-1,n-1} & 1-p_{n-1,n-1} \\ 0 & \dots & \dots & 0 & 0 \end{bmatrix}$$

An interesting property of the self-loop architecture is that we can change the order of the states, i.e. swap any[1] two pairs of cells $(p_{i,i}, p_{i,i+1})$ and $(p_{j,j}, p_{j,j+1})$ in the transition matrix, without changing the resulting absorption time distribution. Intuitively, it follows from the fact that we need to go through all of the states on our route from the starting state to the absorbing state. Formally, it is possible to transform the probability of getting into the absorbing state after exactly K steps into a clearly symmetric function w.r.t. transition weights $p_{i,i}$:

$$\begin{aligned} \Pr[\text{absorbing after } K \text{ steps}] &= \sum_{\substack{k_0+\dots+k_{n-1}=K \\ k_i \geq 1}} \prod_{i=0}^{n-1} p_{i,i}^{k_i-1}(1-p_{i,i}) = \\ &= \left(\prod_{i=0}^{n-1}(1-p_{i,i})\right) \cdot \sum_{\substack{k_0+\dots+k_{n-1}=K-n \\ k_i \geq 0}} \prod_{i=0}^{n-1} p_{i,i}^{k_i} = \\ &= \left(\prod_{i=0}^{n-1}(1-p_{i,i})\right) \cdot \left(\sum_{i=0}^{n-1} p_{i,i}\right)^{K-n} \end{aligned}$$

Another property of this architecture is that the minimum absorption time is equal to the number of states of the Markov chain.

2.3.1 Performance of the self-loop architecture

We have fitted the self-loop architecture to the individual challenge datasets and plotted the resulting cross-entropy against the number of states (see Figure 2.3). We will refer to this curve as *Performance curve*, as it depicts the model success in fitting. We ran each experiment 10 times and selected the best cross-entropy out of them for each number of states. Additionally, we displayed the best overall distribution for each challenge in Figure 2.4. The technical details of the experiments are located in Chapter 6.

We observe an unintuitive increase in the cross-entropy in three out of four challenges. The problem is that the minimal absorption time is n for a self-loop Markov

[1] except the last row, of course

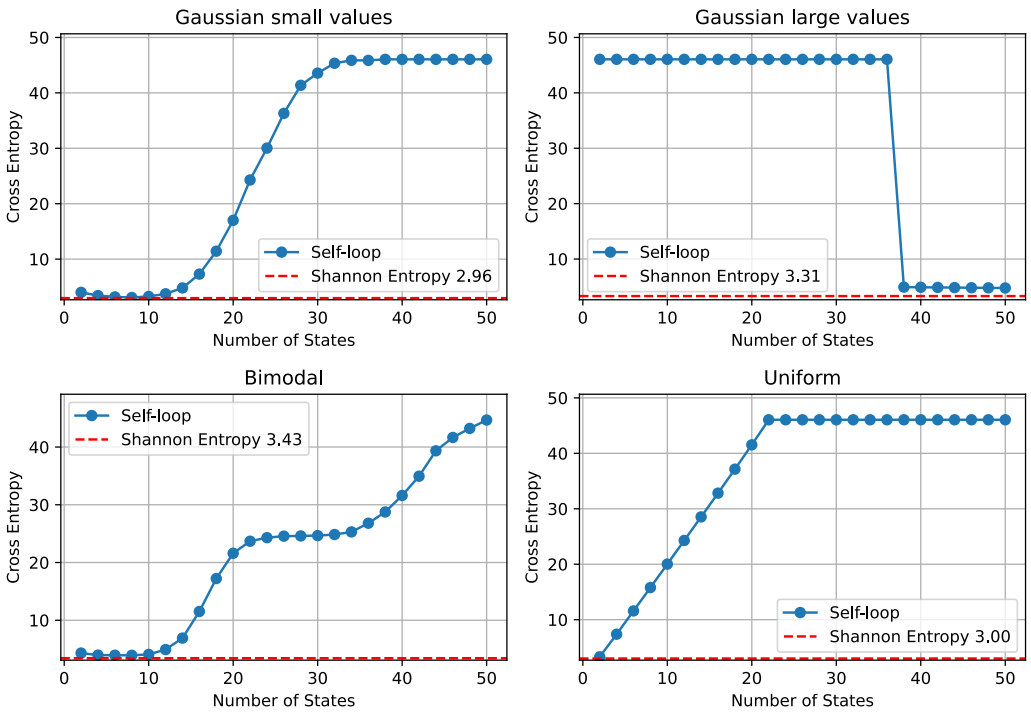

Figure 2.3: Comparison of cross-entropy versus the number of states for each challenge for the self-loop architecture. Cross-entropy in blue, Shannon entropy red.

chain with n states. Therefore, a large Markov chain with such architecture is unable to fit a distribution with small values, thus increasing the cross-entropy. This effect can be clearly observed with the Uniform challenge, where the smallest Markov chain with two states provides the best fit.

Additionally, we can see that in Gaussian large values challenge adding states till 38 does not affect cross-entropy at all. After 38 states we observe a dramatic drop in cross-entropy. This tells us that 38 states for self-loop architecture are sufficient to model a data with mean around 350. However, adding states is not a practical long-term strategy, therefore we will be searching for other techniques to model data with larger values.

Now let's inspect the best overall fits for each synthetic challenge (see Figure 2.4). As we can see, the self-loop model of 8 states is perfectly capable of modeling the Gaussian distribution with small values, achieving almost optimal cross-entropy. However, we need up to 50 states to model the large values Gaussian, still not getting close to the optimal cross-entropy and failing to mimic the data shape. In the Bimodal challenge we achieved reasonably good cross-entropy, however we can observe that it fails to model the "gap" in data and instead interpolates data together. In a similar fashion, the self-loop architecture achieves a reasonably good cross-entropy, but fails to learn the shape of the uniform distribution in the last challenge.

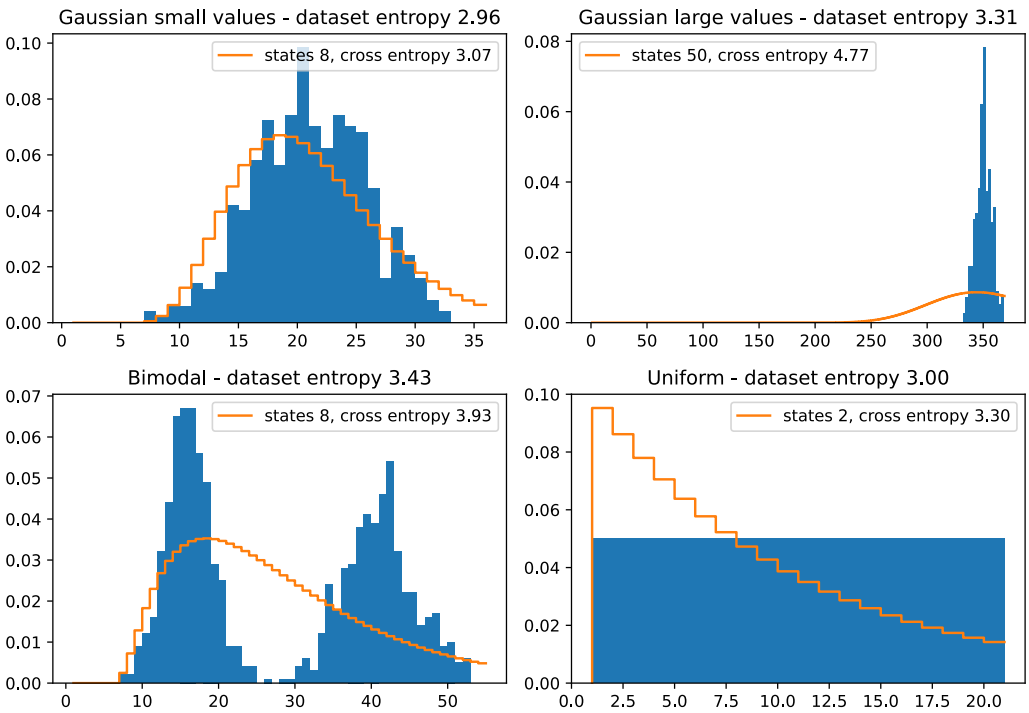

Figure 2.4: Best training for self-loop architecture, each figure shows density histogram of the challenge, in blue and learned phase-type distribution in orange.

2.4 Early-escape architecture

Consider a Markov chain where each state has two (non-zero probability) outgoing arrows: one to the next state, and another straight to the absorbing state (creating an "early escape" from the chain). We will call such configuration an *early-escape architecture* (see Figure 2.5). This model is again inspired by [17].

This idea is formalized as follows. Consider an absorbing Markov chain with finite state space I, one absorbing state $X \in I$ and transition matrix P, where $p_{i,i+1}+p_{i,X} = 1$ for all $i \neq X$. As a convention, the last row of P will contain the probabilities for absorbing state X.

$$P = \begin{bmatrix} 0 & 1-p_{1,X} & 0 & \dots & p_{1,X} \\ 0 & 0 & 1-p_{2,X} & \dots & p_{2,X} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & p_{n-1,X} \\ 0 & \dots & \dots & 0 & 0 \end{bmatrix}$$

Note that in this architecture, as opposed to the previous self-loop architecture, the order of the transition probabilities does affect the resulting absorption time distribution. It is easy to see that swapping a small escape weight at the beginning of the chain with a large escape weight further in the chain will decrease the mean absorption time.

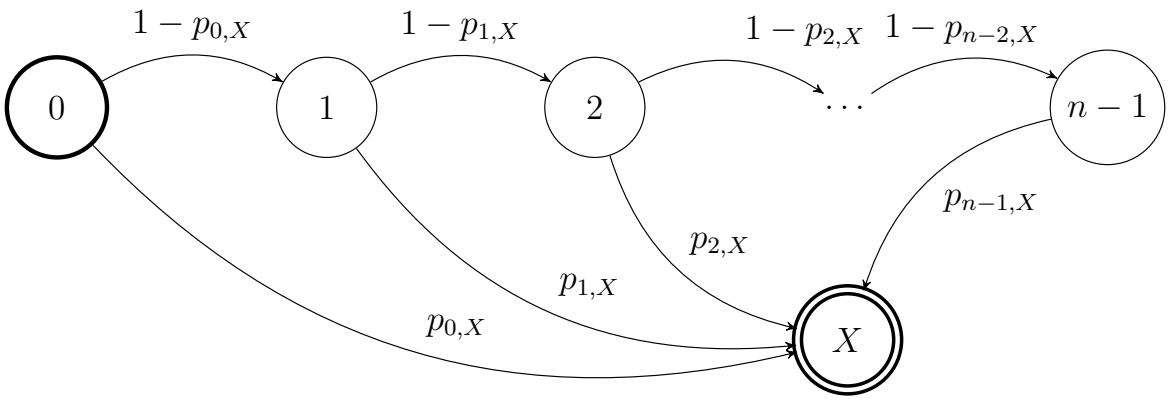

Figure 2.5: The early-escape architecture. State 0 is the initial state, and state X is the absorption state.

It is important to note that this architecture is acyclic. Therefore it has no means of prolonging the absorption process, therefore its maximal time to absorption is n for architecture of n states.

2.4.1 Performance of the early-escape architecture

We have performed the same experiments as for the previous architecture (see subsection 2.3.1). The performance curves are shown in Figure 2.6 and the overall best fits are shown in Figure 2.7.

The performance curve of the early-escape architecture is *non-increasing* in general. Additionally, given enough states, the early-escape architecture can fit any input dataset perfectly reaching the minimum cross-entropy. This was reached in Gaussian small values, Bimodal and Uniform challenge.

The early-escape architecture is unsuited for modeling larger values, as shown in Gaussian large values (see Figure 2.6). Since this architecture is acyclic, its maximal time to absorption is n for architecture of n states. Therefore, one would require around 400 states in order to adequately fit the Gaussian large values challenge, which is way outside of the scope of our project.

This model complements the weakness of the self-loop architecture, which was not able to fit small values and large values at the same time. This is particularly evident, when comparing early-escape cross-entropy and previous best cross entropy in Figure 2.6.

In the Bimodal challenge, the cross-entropy has three "plateaux" which correspond to the ranges with little or no data, while the sharp plunges correspond to the ranges with the most "mass" of the dataset. Figure 2.8 shows the resulting distributions after the first and the second plunges. We can see that given 30 states, the model fits the first peak perfectly (while scaling the individual probabilities to preserve the total

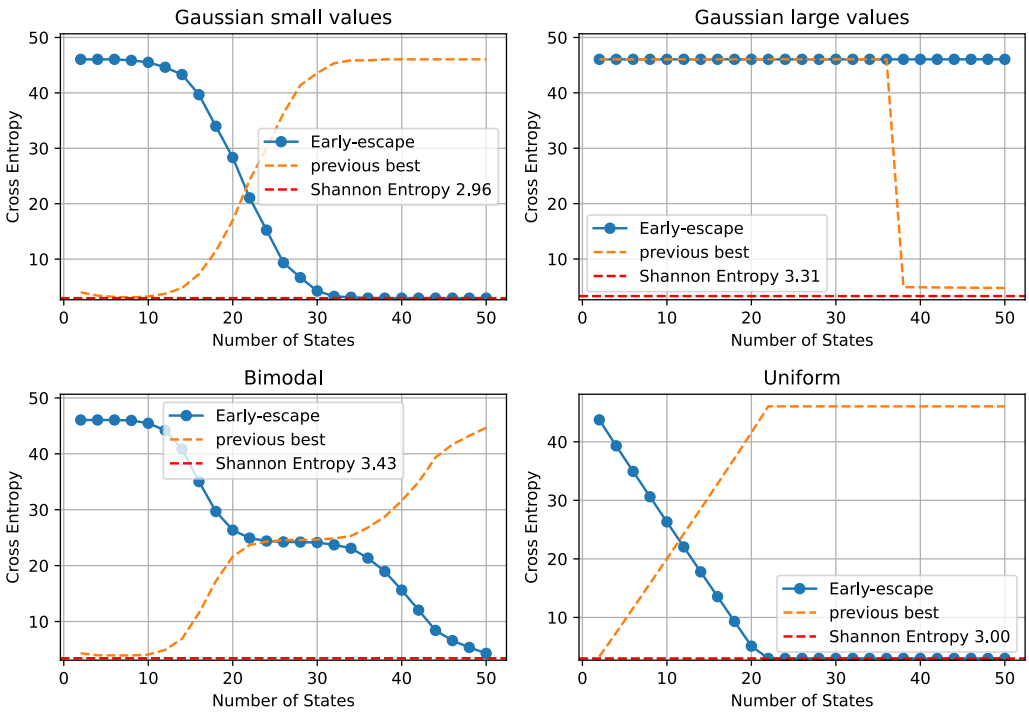

Figure 2.6: Performance curve for each challenge for the early-escape architecture. Cross-entropy in blue, Shannon entropy red, previous best achieved cross-entropy in orange.

probability mass).

In the Uniform challenge, we observe a steady decrease of the cross-entropy.

Now let's inspect the best overall fits for each synthetic challenge (see Figure 2.7). In both Gaussian small values and Bimodal challenge we observe that the early-escape architecture with enough states can fit data more accurately, mimicking also the shape of data precisely, thus reaching the optimal cross-entropy. When looking at the problem from the machine learning point of view, this might seem as an overfitting, since the model also learns the "noise" in the data. However, we are not trying to learn a trend in data, but to preserve as much information as possible, therefore this is a desired model behavior.

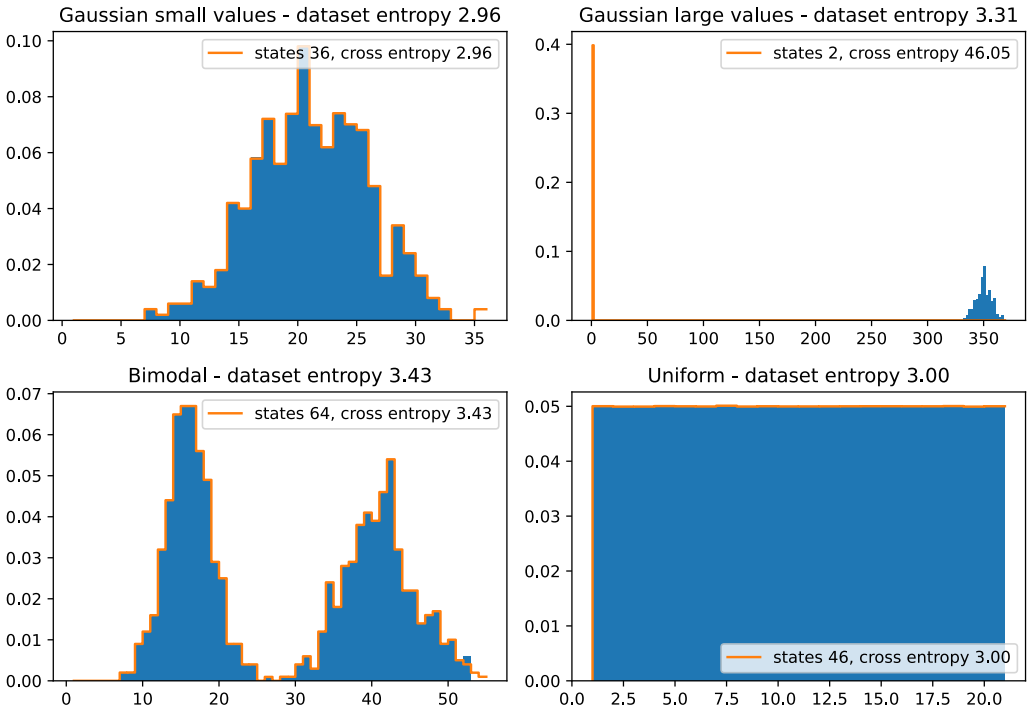

Figure 2.7: Best training for Early-escape architecture, each figure shows density histogram of the challenge, in blue and learned phase-type distribution in orange.

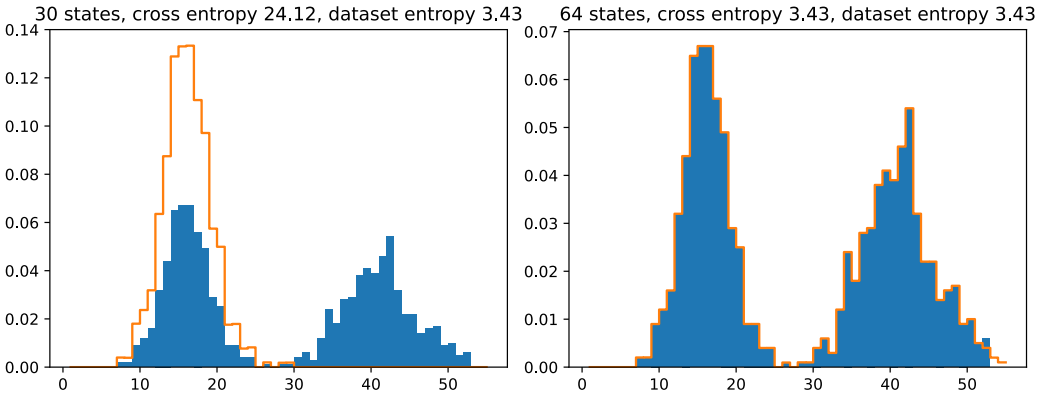

Figure 2.8: Comparison of two learned phase-type distributions for the Bimodal challenge with models having 30 and 50 states. Density histogram of Bimodal challenge in blue, learned phase-type distribution in orange

2.5 Combined architecture

Now, we introduce architecture that combines the advantages of both examined architectures; the ability of the self-loop architecture to model the datasets with large values (larger than the number of states) and the ability of the early-escape architecture to model the datasets with small values (smaller than the number of states). This corresponds to a state automaton, with each state having a *self-loop*, a *transition to the absorbing state* and a *transition to a next state*. We call this model the *combined architecture* (see Figure 2.9).

This idea is formalized as follows. Consider an absorbing Markov chain with finite state space I, one absorbing state $X \in I$ and transition matrix P, where $p_{i,i} + p_{i,i+1} + p_{i,X} = 1$ for all $i \neq X$. As a convention, the last row of P will contain probabilities for absorbing state X.

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & 0 & \dots & p_{1,X} \\ 0 & p_{2,2} & p_{2,3} & \dots & p_{1,X} \\ 0 & 0 & p_{3,3} & \dots & p_{2,X} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & p_{n-1,n-1} & p_{n-1,X} \\ 0 & \dots & \dots & 0 & 0 \end{bmatrix}$$

We expect less overfitting, a trend we have encountered with the early-escape architecture, because the self-loops should have some "smoothing" effect on the resulting phase-type distribution.

Both the early-escape and self-loop architectures are special cases of the combined architecture. The early-escape architecture sets $p_{i,i}$ to 0, while the self-loop architecture sets $p_{i,X}$ to 0. Hence, the combined architecture should model data with a comparable accuracy level by simply learning 0 in the relevant transition probabilities.

2.5.1 Performance of the combined architecture

We have performed the same experiments as for the previous architectures (see subsections 2.3.1 and 2.4.1). The performance curves are shown in Figure 2.10 and the overall best fits are shown in Figure 2.11.

The combined architecture clearly outperforms the previous ones in the challenges with small values, achieving a low cross-entropy with considerably less states required. We can see that combined architecture performance curve copies or outperforms previously achieved cross-entropy. In Figure 2.11 we can see that for small values Gaussian challenge combined architecture reaches almost optimal cross-entropy, outperforming self-loop architecture. In Bimodal challenge outperforms both previous architectures,

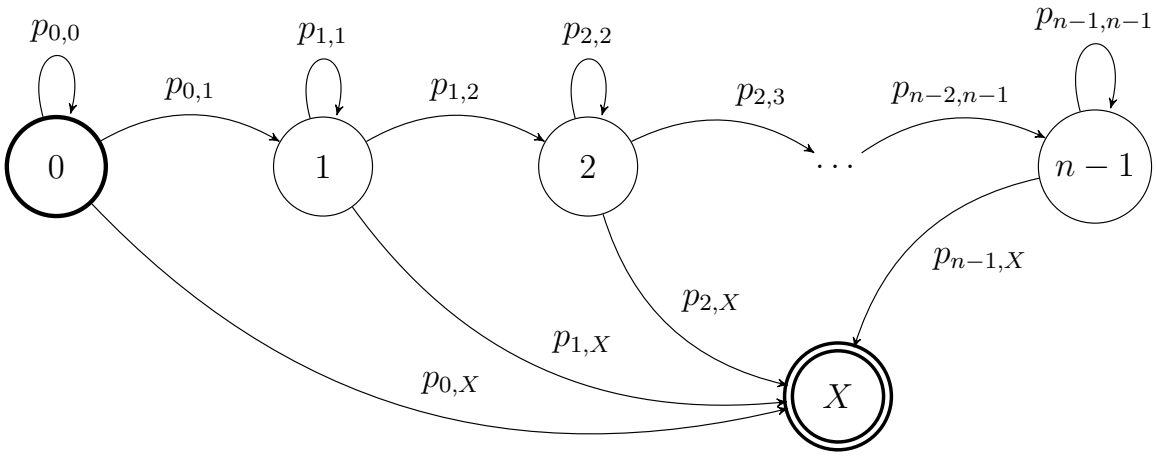

Figure 2.9: The combined architecture. State 0 is the initial state, and state X is the absorption state.

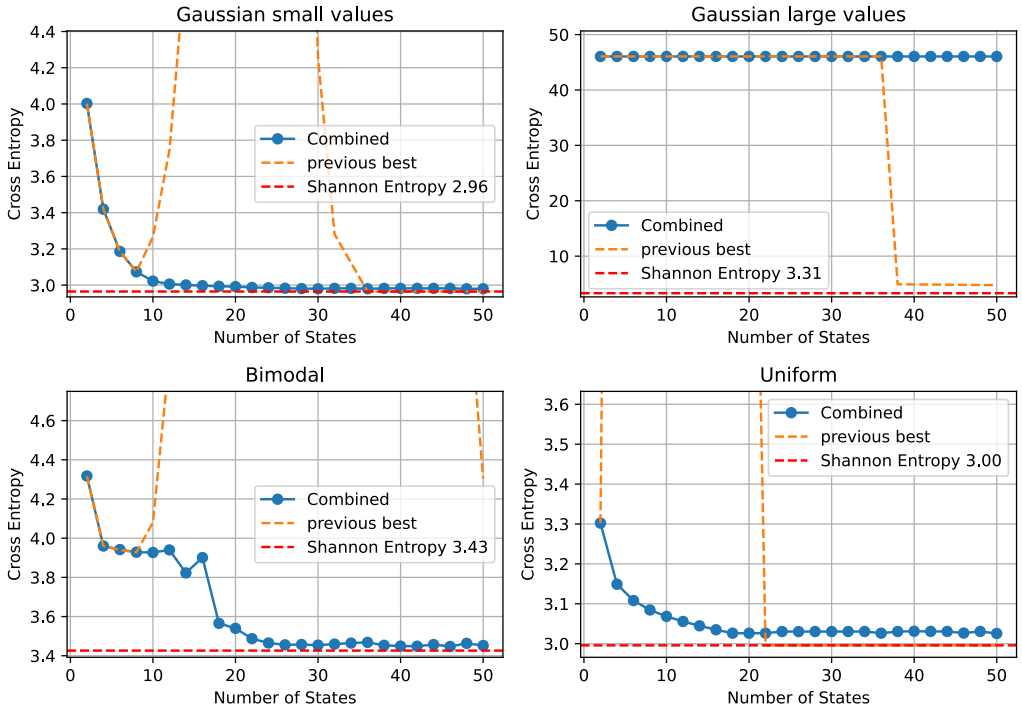

Figure 2.10: Performance curve for each challenge for the combined architecture. Cross-entropy in blue, Shannon entropy red, previous best achieved cross-entropy in orange.

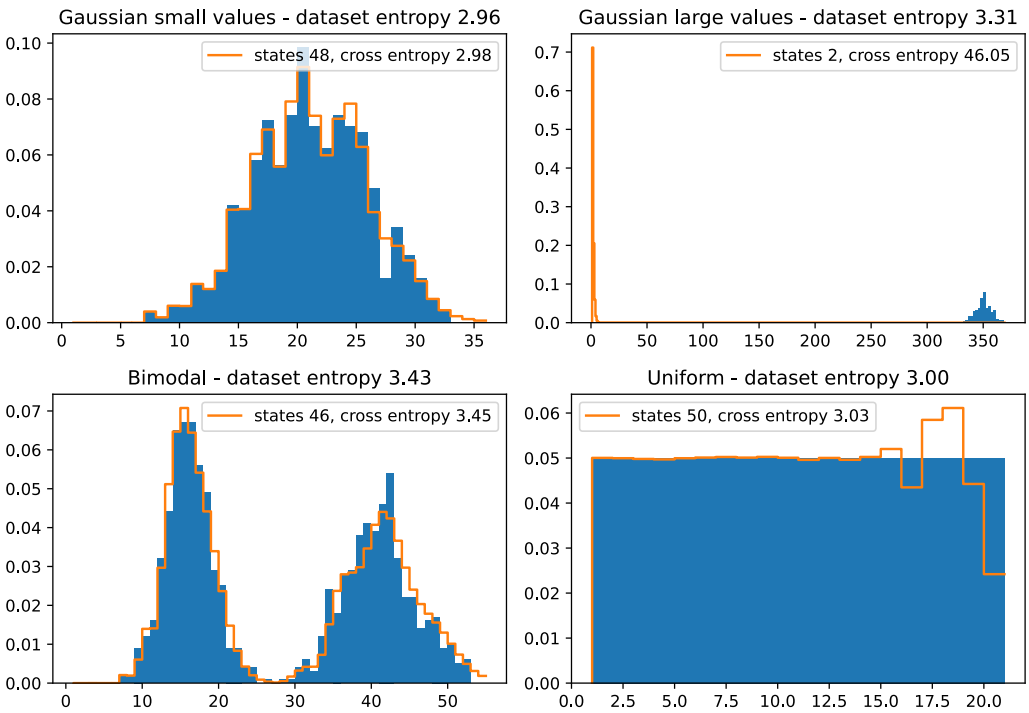

Figure 2.11: Best training for combined architecture, each figure shows density histogram of the challenge, in blue and learned phase-type distribution in orange.

managing to capture "gap" in data at 46 states. The model also trains itself reasonably well at Uniform challenge.

The combined architecture does not fit the data distribution perfectly, retaining some smoothing effect. This can be observed in Figure 2.11 in the Bimodal challenge, where the model does not learn the precise data shape but results in a smoother distribution. Consequently, although Figure 2.10 shows a desired drop in the performance curve, it does not entirely reach the optimal Shannon entropy.

The model struggles with the large values as seen in Gaussian large values challenge, where it fails to train. We will address this issue next.

Optimizer failure

As discussed earlier, both previous architectures are special cases of the combined architecture. However, the experiments have shown the inability of the optimizer to find satisfactory transition probabilities, even when they clearly exist. For instance, in the small values Gaussian challenge, the combined architecture fails to achieve optimal cross-entropy, unlike the early-escape architecture. Moreover, while the self-loop architecture with 50 states can model the large values Gaussian challenge, the combined architecture fails to do so (see Figure 2.11).

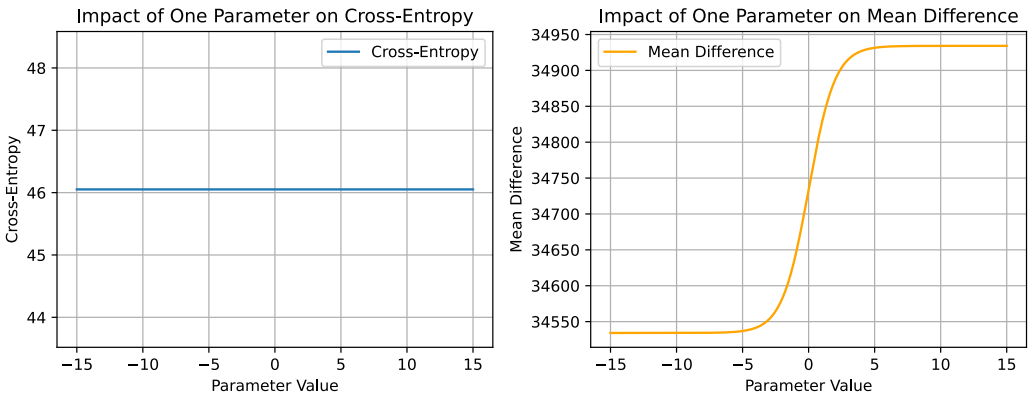

Figure 2.12: Comparative Impact of a Single Parameter on Cross-Entropy and Mean Difference for combined architecture of 30 states on on large values Gaussian challenge.

2.5.2 Improvement of the optimisation by fitting the mean first

The experiments have shown the inability of the optimizer to find a satisfactory solution for the Gaussian big values challenge despite the fact that it clearly exists. The issue is that the stopping criterion *projected gradient norm threshold* (see subsection 1.4.5) is fired on the very first iteration. In other words, a change in the individual weights right after the random initialization does little to improve the overall cross-entropy. We assume that this effect arises because the majority of the "mass" of the initial phase-type distribution is located *too far* from the target distribution (see Figure 2.13).

We came up with two possible fixes for this problem. The first is to decrease the stopping threshold *PGTOL*. However, this solution is not scalable for inputs with larger data points. Instead, we've decided to move the mass of the phase-type distribution closer to the target distribution first, and only then run the full optimisation of the weights. We call this simplified pre-training procedure the *mean shifting*.

Specifically, we change the optimisation target to the difference between the mean of the input distribution and the mean of the phase-type distribution. We compute the mean of the phase-type distribution using the matrix formula described in subsection 1.3.1. This procedure allows the individual weights to have much greater impact on the optimisation target (see Figure 2.12), thus not firing the stopping criteria too early. Once the pre-training is completed, we take the resulting weights, apply a small noise and use them as an initial point of the full optimisation.

Performance evaluation of Combined architecture with mean shifting

We have performed the same experiments as for the previous architectures (see subsections 2.3.1 and 2.4.1 and 2.5.1). The performance curves are shown in Figure 2.14 and the overall best fits are shown in Figure 2.15.

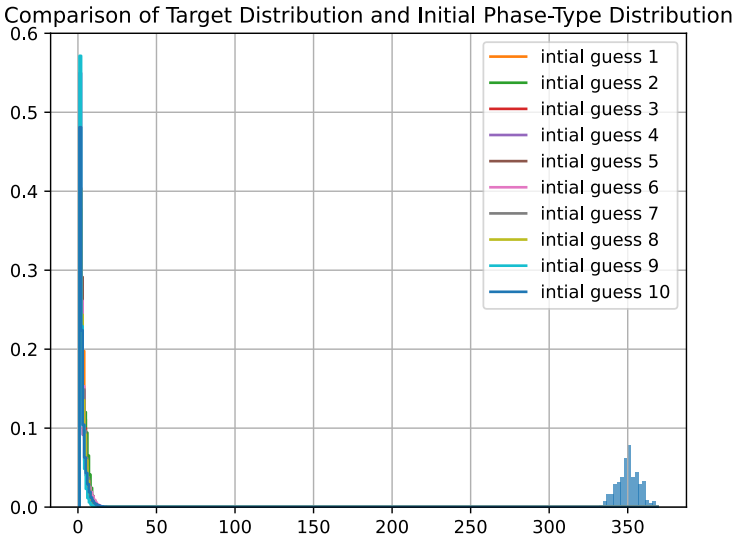

Figure 2.13: Comparison of target distribution and initial phase-type distributions for combined architecture of 30 states on large values Gaussian challenge.

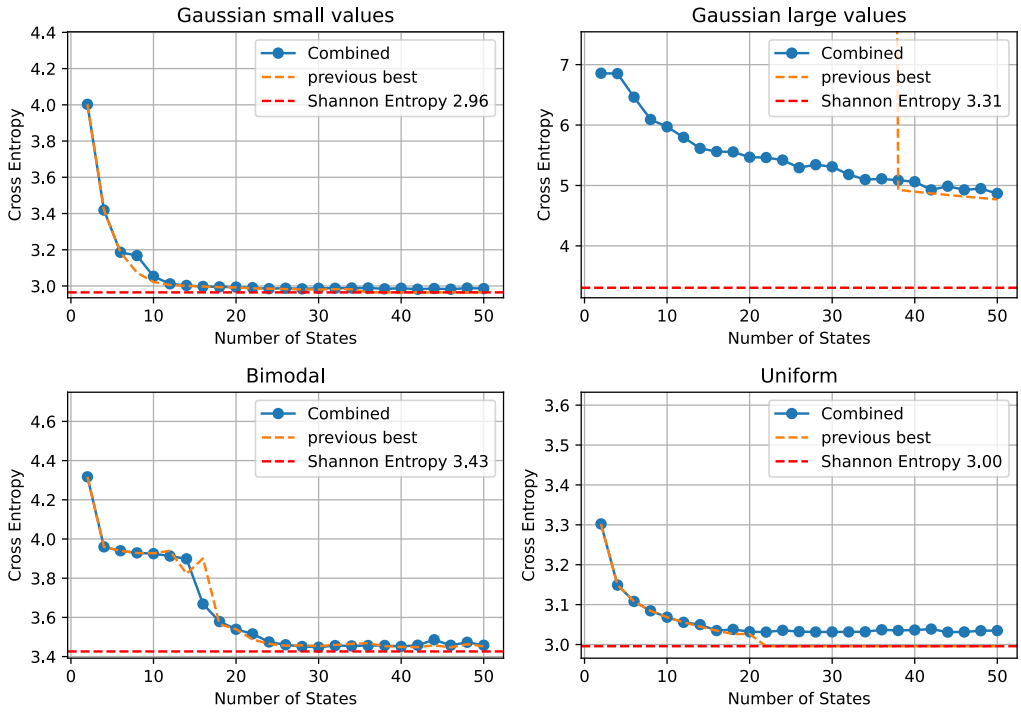

Figure 2.14: Performance curve for each challenge for the combined architecture with mean shifting. Cross-entropy in blue, Shannon entropy red, previous best achieved cross-entropy in orange.

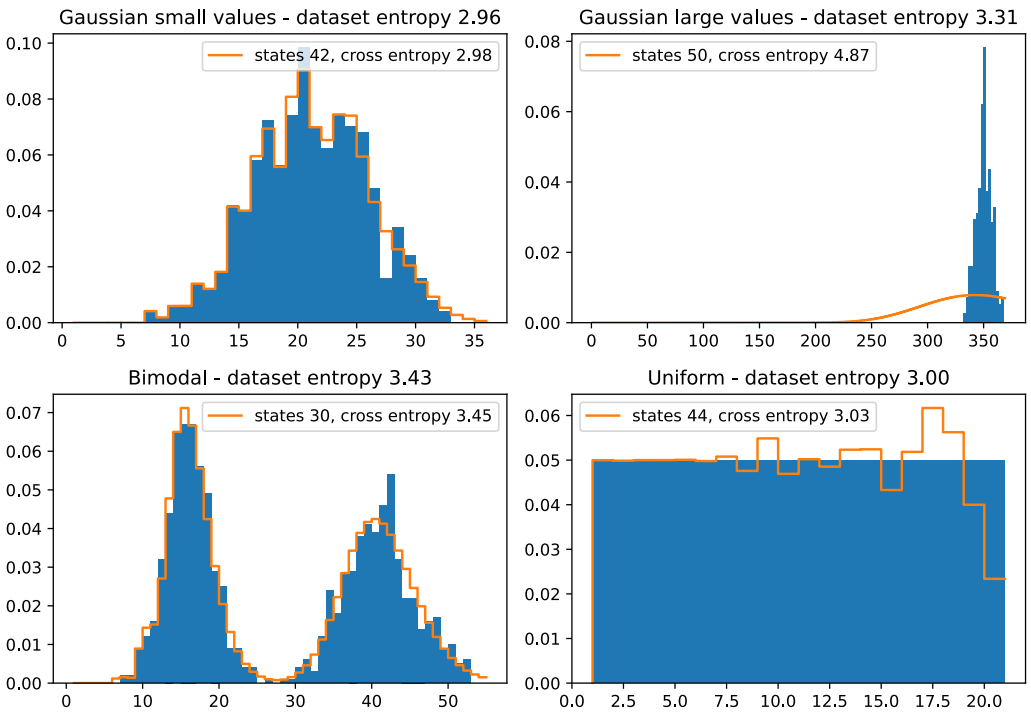

Figure 2.15: Best training for combined architecture with mean shifting, each figure shows density histogram of the challenge, in blue and learned phase-type distribution in orange.

The mean shifting significantly improves the combined architecture performance on the Gaussian big values challenge. We can also observe a steady drop in cross-entropy with adding states (see Figure 2.14). When looking at the best trained model in Figure 2.15, we can clearly see that the mean of the resulting phase-type distribution moved to the mean of data distribution, however the resulting phase-type distribution does not mimic the shape of the data.

We can see that this technique did not compromise the performance elsewhere much. This can be seen in Figure 2.15, where the small values Gaussian, Bimodal and Uniform challenges are fit quite well. In both Bimodal and Uniform challenge it worsened the best achieved cross-entropy slightly, compared to training without mean shifting (see Figure 2.14).

Chapter 3

Advanced Markov chains architectures

In this chapter we will introduce dense architectures *fully connected architecture*, *pruned architecture* and *k-jumps architecture*. We will again test them against a set of challenges and describe their weaknesses and properties.

In section 1 we present Fully connected architecture, we proceed to show its performance on synthetic challenges. In section 2 we present a new concept on pruning and compare the results with previous architecture. In section 3 we present Cyclic architecture and again evaluate its performance. Lastly in section 4 we present k-jumps architecture.

3.1 Fully connected architecture

We will present an architecture with each state having *self-loop*, *transition to absorbing state* and *transition all other transient states*, to this model we will refer to as *fully-connected architecture* an its 4 state version is depicted in Figure 3.1.

This idea is formalized as follows. Consider absorbing Markov chain with finite state space I, one absorbing state $X \in I$ and transition matrix P where $\sum_{j=0}^{i} p_{i,j} = 1$ for all $i \neq X$. As a convention, the last row of P will contain probabilities for absorbing state X.

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,X} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{1,X} \\ p_{3,1} & p_{3,2} & p_{3,3} & \cdots & p_{2,X} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ pn-1,1 & \dots & p_{n-1,n-2} & p_{n-1,n-1} & p_{n-1,X} \\ 0 & \dots & \dots & 0 & 0 \end{bmatrix}$$

The fully connected architecture is the most dense architecture for a given amount of states, and therefore it has all the effects we have discussed in section 2.1 about the model selection criteria. To summarize, its theoretical expressive power is the best

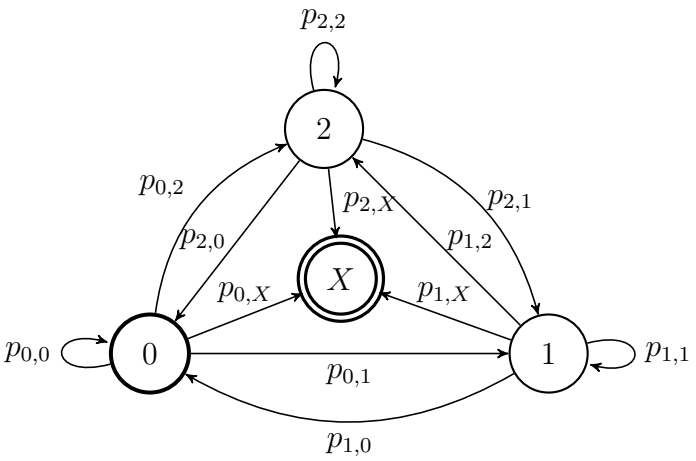

Figure 3.1: The fully connected architecture of 4 states. State 0 is the initial state, and state X is the absorption state.

among all other architectures, while its training speed and the ability to reach a good fit in practice should be the worst.

3.1.1 Performance evaluation of fully connected architecture

Fully connected architecture is never significantly better than the previous sparse architectures, i.e., it cannot achieve significantly better cross-entropy with the same amount of states. In all challenges, the fully connected architecture managed to mimic the data shape to some extent, but not as effectively as previous architectures. In Gaussian small values and Bimodal challenge, the fully connected architecture can only partially mimic the data shape (see Figure 3.3), lacking the precision of either combined or early-escape architectures. In Uniform challenge, fully connected architecture manages to jump around the data distribution rather precisely given its complexity, but once again fails to take on the structure of early-escape architecture that is most suitable for this task.

Its performance gets progressively worse with more states added (see Figure 3.2). We can observe this in Gaussian small values challenge, where the performance curve shows increase after 30 states. Similar trends of increasing cross-entropy after a certain number of states were observed in Uniform and Bimodal challenges.

It may be partially attributed to the fact that the L-BFGS-B algorithm needs to approximate the inverse Hessian matrix of all parameters using the constant memory. The Hessian matrix is the matrix of second partial derivatives, i.e., it has $O(n^4)$ cells, where n is the number of states of the Markov chain. It might be that such a steep increase in the number of values makes the performance of the inverse Hessian matrix approximation degrade quickly as well. This could be potentially fixed by changing the optimisation algorithm from a second-order method to a first-order (e.g., gradient

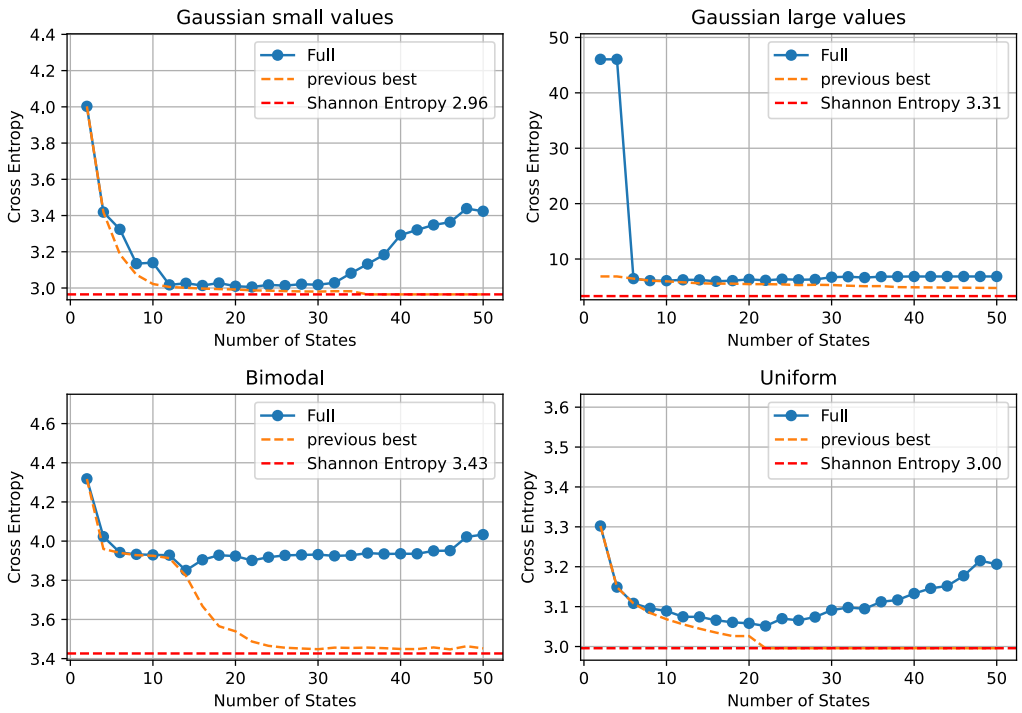

Figure 3.2: Performance curve for each challenge for the Fully connected architecture.

descent) or a derivation-free (e.g., Nelder-Mead [27] or simulated annealing [2]) one.

One of the most significant achievements of this architecture is managing to model Gaussian large values challenge at 6 states already without the need of mean shifting, outperforming by far all the previous architectures. This is due to the structure of fully connected architecture. With each state being connected to each other, there is a higher likelihood of remaining transient states. This interconnectedness enhances cyclic properties similarly to self-loop architectures but with significantly improved outcomes. The process has a tendency to stay in transient states and thus prolong the absorption time. In the Gaussian large values challenge even though we achieved acceptable cross-entropy at 16 states, we failed to learn the shape of the data.

The training time for fully connected architecture is growing consistently with the number of states and is among the highest (see Figure 3.4). The peak memory consumption during the training of the previous architectures stays around 168 MB, whereas for the fully connected architecture it consistently rises up to 172 MB for 50 states.

In conclusion, our experimental results confirm our suspicions regarding the performance of the fully connected architecture. Despite its ability to partially mimic the data shape across various challenges, it falls short compared to previous, sparse architectures. Additionally, its performance worsens with the addition of more states, suggesting that using this architecture is not recommended.

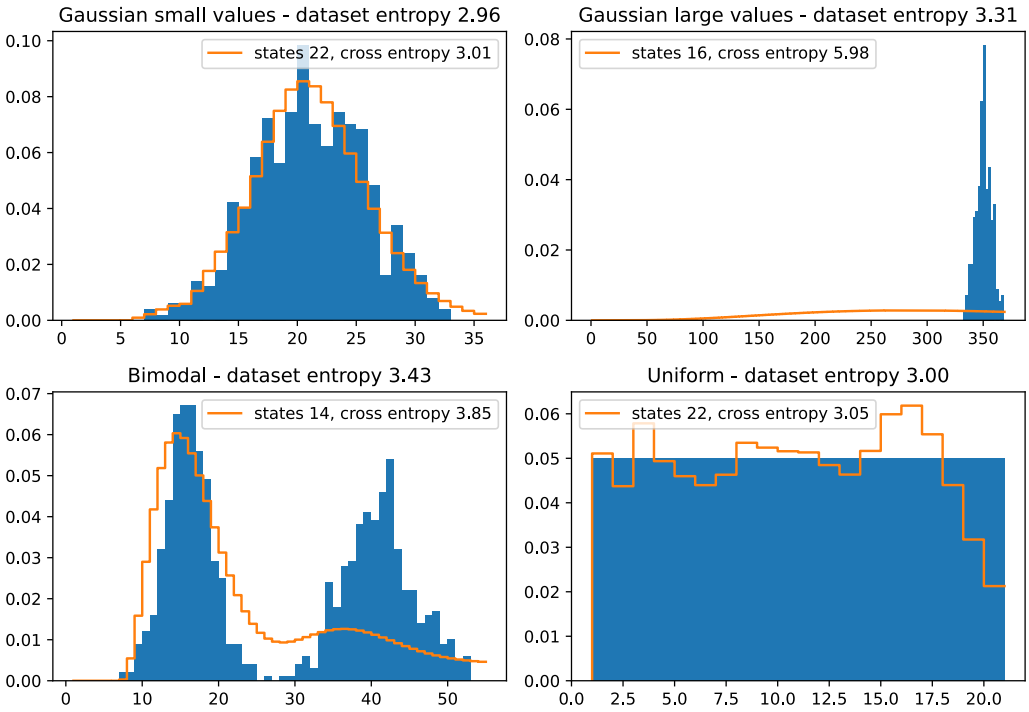

Figure 3.3: Best training for fully connected architecture, each figure shows density histogram of the challenge, in blue and learned phase-type distribution in orange.

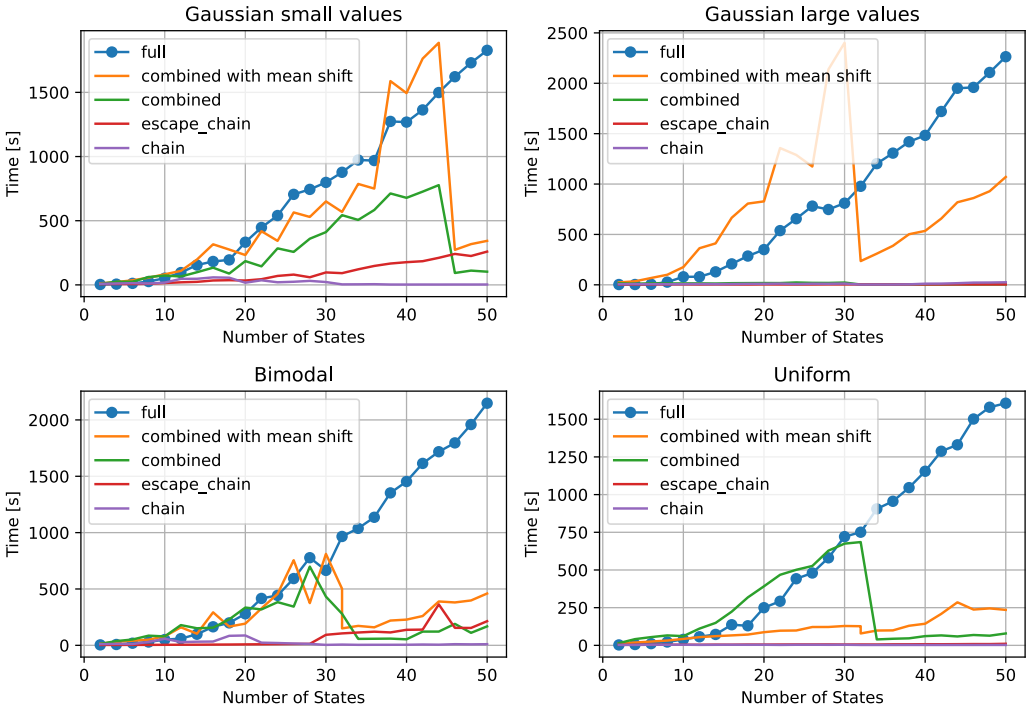

Figure 3.4: Time comparison between training of fully connected and previous architectures across all four challenges.

3.2 Pruned architecture

When examining the learnt transition probabilities for fully connected architecture (see Figure 3.5), it was observed that certain transitions were, with notably low probabilities. We came up with the concept of pruning, which involves setting probabilities lower than a certain threshold to 0. After this, we normalize the resulting probabilities so that the transition probabilities from a state sum up to 1.

The pruning may lead to a reduction in matrix density (by removing the edges) and a reduction in the number of states (by removing the inaccessible states). The reduction in the number of states has a direct positive effect on the training and usage time. The reduction in the matrix density (see Figure 3.6) may also have a positive effect on the downstream usage by allowing to use algorithms specific for the sparse matrices. The pruned architectures may also serve as an inspiration for new architecture designs.

3.2.1 Performance evaluation of the pruned architecture

Pruning is less effective for large values datasets. Notably, in the Gaussian large values challenge, pruning with a threshold of 0.013 resulted in an untrained phase-type distribution (see Figure 3.7). This could be caused by the removal of the majority of escape probabilities, which were initially low to achieve a far mean. Pruning also affected the small values Gaussian, where it eliminated short absorption times (see Figure 3.8), likely by removing small escaping probabilities from the first states.

Pruning can occasionally lead to unexpectedly positive results. For example, in the Gaussian small svalues (38 and 40 states), pruning resulted in better cross-entropy than the original architecture (see Figure 3.8). However, this is best understood as a fortunate accident. Generally, we expect cross-entropy to worsen with pruning and aim to find the largest pruning threshold that keeps cross-entropy acceptable.

Pruning makes the phase-type sharper. This effect is evident in the Uniform challenge, where pruning with a threshold of 0.05 resulted in a sharper distribution with a similar shape (see Figure 3.8). This observation is further supported by an extreme case seen in the large value Gaussian, where pruning likely removed transitions between transient states. Consequently, this shortened the possible absorption time and led to poor cross-entropy (see Figure 3.7).

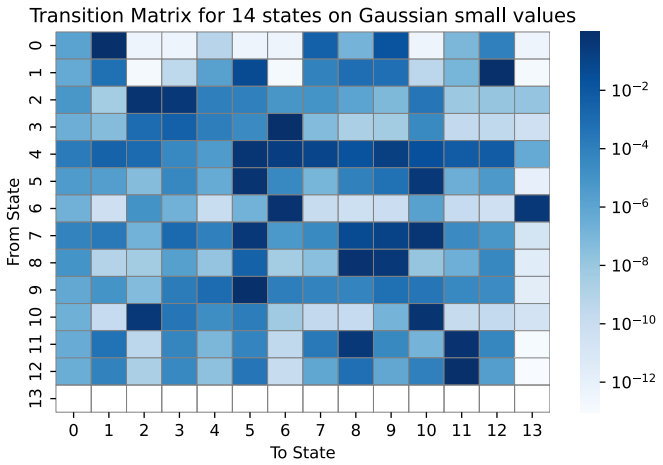

Figure 3.5: Visualization of learned transition probabilities. The heatmap shows the learned transition probabilities for a fully connected architecture with 14 states on the Gaussian small values challenge. The colors represent the probabilities on a logarithmic scale.

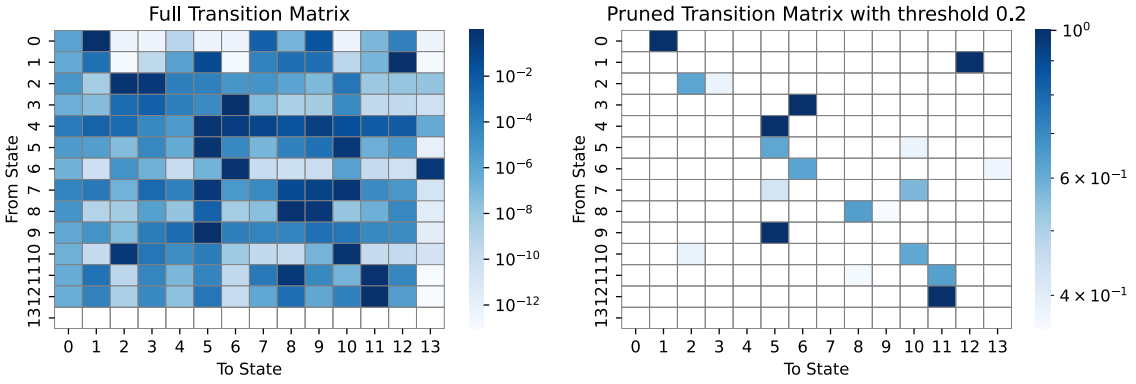

Figure 3.6: Density reduction in transition matrix of 14 states trained no small values Gaussian, with threshold 0.2

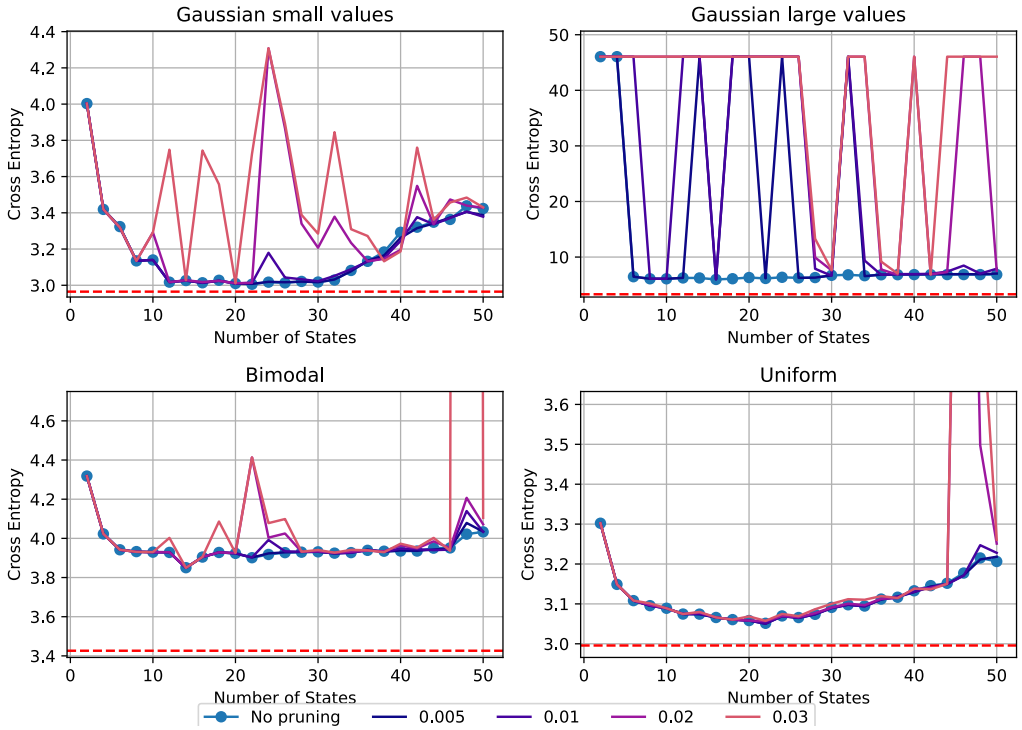

Figure 3.7: Effect of pruning on cross-entropy for all four challenges for different pruning thresholds.

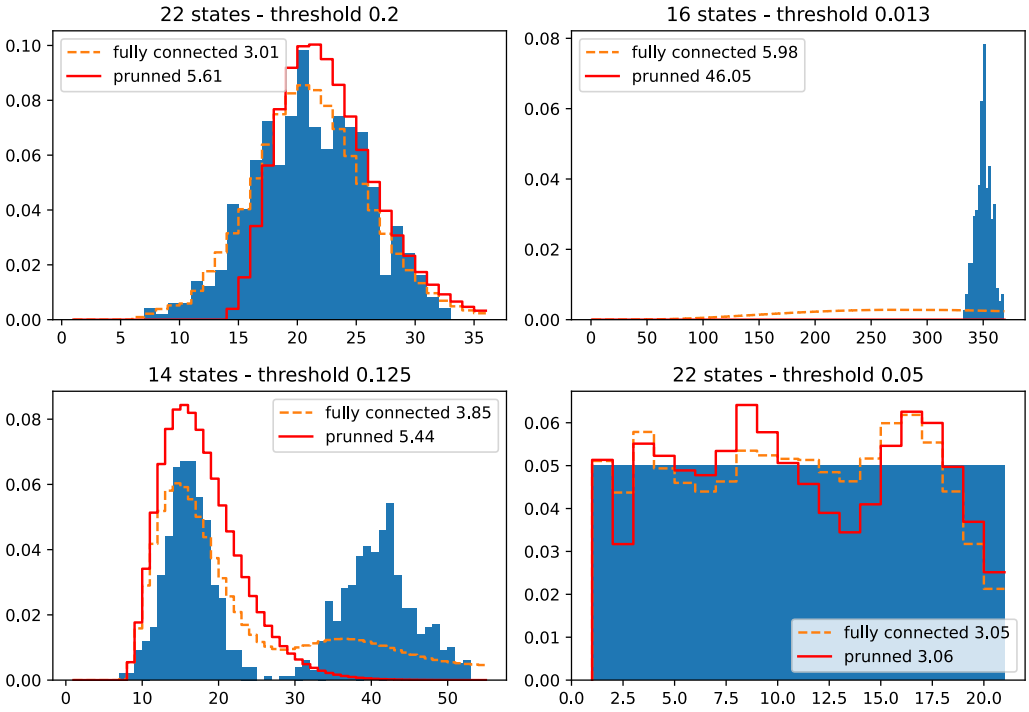

Figure 3.8: Effect of pruning on phase-type distribution for best trained fully connected model on each challenge. Thresholds were chosen based on their ability to show deviations from the original fit while still providing a reasonable fit. For the large values Gaussian challenge, the threshold of 0.013 is the smallest that results in an untrained model; smaller thresholds yield the same fit as the original.

3.3 k-jumps architecture

We have observed that the sparse architectures struggle with datasets with large values, since they cannot prolong the absorption time enough. Therefore, we decided to add a "backlink" to each non-absorbing state in the combined architecture, allowing to go k states back. We will refer to this architecture to as *k-jumps architecture*.

This idea is formalized as follows. Consider absorbing Markov chain with finite state space I and constant k, one absorbing state $X \in I$, and transition matrix P where $p_{i,i+1} + p_{i,i} + p_{i,X} = 1$ if $i < k$ and for all $i > k$, $p_{i,i-k} + p_{i,i+1} + p_{i,i} + p_{i,X} = 1$. As a convention last row of P will contain probabilities for absorbing state X.

$$P = \begin{bmatrix} p_{0,0} & p_{0,1} & 0 & \dots & 0 & \dots & p_{0,X} \\ 0 & p_{1,1} & p_{1,2} & \dots & 0 & \dots & p_{1,X} \\ 0 & 0 & p_{2,2} & \dots & 0 & \dots & p_{2,X} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ \dots & p_{i,i-k} & \dots & p_{i,i} & p_{i,i+1} & \dots & p_{i,X} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \\ 0 & \dots & \dots & 0 & 0 & 0 & 0 \end{bmatrix}$$

K-jumps architecture has a cyclic structure, similar to fully connected architecture, therefore we expect this architecture to be able to model distributions with large mean value.

The constant k is a hyperparameter, has to be chosen independently from the optimisation process, which complicates the model selection.

3.3.1 Performance evaluation of the k-jumps architecture

We have performed the same experiments as for the previous architectures. The performance curves are shown in Figure 3.10 and the overall best fits are shown in Figure 3.11.

When comparing the performance of the k-jumps architecture with respect to the hyperparameter k, it appears that k = 7 performs the best. This observation is supported by Figure 3.10, where the performance curve for k = 7 outperforms the others. Additionally, in Figure 3.11, during the bimodal challenge, k = 7 manages to mimic the shape of the data most accurately.

Contrary to our intuition, the additional back-loops intended to enhance cyclicity did not improve the performance on the Gaussian large values challenge. We failed to learn the data distribution for all values of k in this scenario (see Figure 3.10).

This architecture does not outperform the combined architecture. The additional back-loops only overparametrize the model, complicating the training. Consequently, we deem this approach a failure because it neither improves performance nor simplifies training; instead, it adds unnecessary complexity.

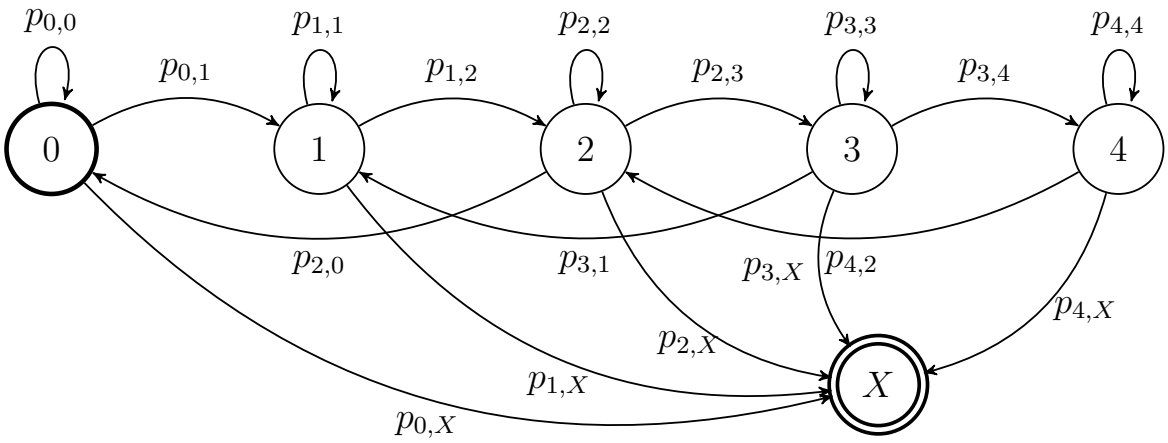

Figure 3.9: K-jumps architecture of 6 states, with $k = 2$ is the initial state and X is absorbing state.

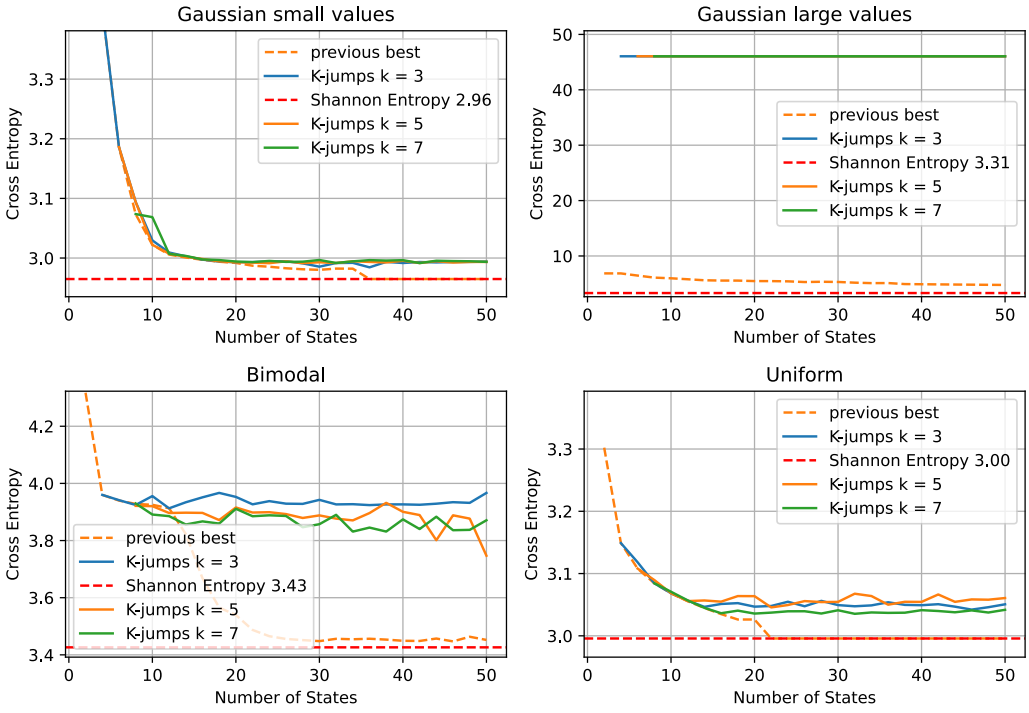

Figure 3.10: Performance curve for k-jumps architecture for different values of k.

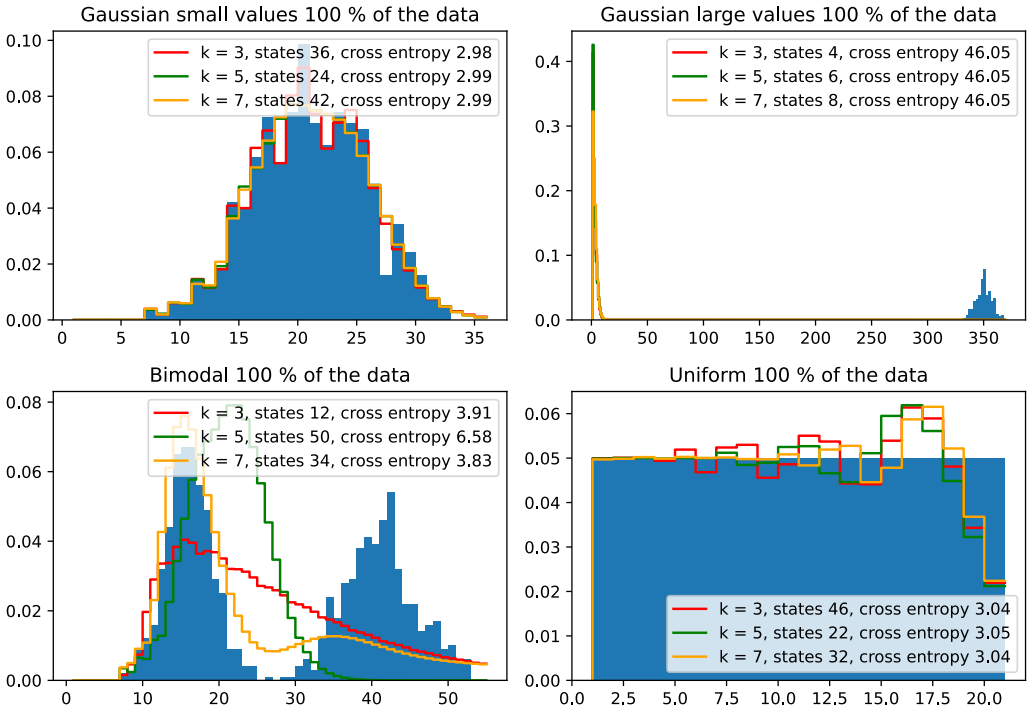

Figure 3.11: Best training for k-jumps architecture. Data density histogram in blue, resulting phase-type distributions in red, green, and orange respectively.

Chapter 4

Real data modeling

In this chapter we will show how our model preformed, when trained on real data from public genomics databases. We choose *combined architecture* with *mean shifting* for this task.

Firstly, we present a new sampling technique, needed when modeling bigger datasets. Then, we present genomic annotations we chose to model and give a brief overview of the used terms as well as some biological background. Then we proceed to intervals modelling, firstly introducing the datasets, then with cross-entropy analysis and in the end we show best trained model for each challenge. Then, we do respectively the same for annotations gaps.

Note that the original MDCP uses 2 states, so any improvement in cross-entropy we achieve is a significant advancement.

4.1 Sampling

So far all of our challenges were ranging from 500 to 1000 samples. However, real-world datasets commonly contain multiple thousands of samples. Due to this model training would be slow, since each evaluation needs to compute cross-entropy, which is calculated from the whole dataset. This can be avoided by reducing the dataset we work with. We will employ a sampling technique, that samples at random form the original dataset, sample of smaller size and the we proceed to train model on this sub-sample of data. This results in faster training processes; however, there is a trade-off between speed and model accuracy. We used samples of size 2000.

4.2 Data

We chose the following datasets due to the fact, that they were also used in original, MCDP article [7].

The **hirt** dataset annotates sequences enriched in extra-chromosomal DNA [30]. Extra-chromosomal DNA refers to DNA molecules existing outside of the main set of chromosomes found in a cell's nucleus. These molecules can be separate circular DNA pieces or linear fragments not part of the chromosomes.

The **Gains inc** dataset annotates regions with copy number gains, which are variations in the number of copies of specific DNA segments [32]. Sometimes, certain regions of DNA can be duplicated, leading to an increased number of copies, which is referred to as copy number gains. These variations can affect gene expression and contribute to genetic diversity.

The **H3K4me3** dataset annotates regions of DNA marked by a specific histone modification called H3K4me3 [11]. Histones are proteins that help package DNA into chromatin, the complex structure of DNA and proteins in the cell nucleus. Histone modifications, like H3K4me3, are chemical alterations to these proteins that can affect how tightly the DNA is packed around them. In this case, H3K4me3 is associated with regions of active gene expression, indicating that the genes in those regions are more likely to be turned on and transcribed into RNA.

Additionally, we included the **Drosophila melanogaster exons** dataset as an example from a public database [22]. Drosophila melanogaster is a species of fruit fly commonly used in biological research as a model organism due to its relatively simple genome. Exons are the coding regions of a gene, they contain the genetic information necessary for producing functional proteins.

4.3 Intervals modeling

Firstly we present a brief overview of the data in Figure 4.1.

Dataset	Mean	Range of interval lengths	Number of intervals
Drosophila exons	488	1–28074	188467
Gains inc	35581	73–3002725	3132
H3K4me3	1937	124–22590	19358
Hirt	1966196	6–24019400	247

Table 4.1: Characteristics Datasets of interval lengths.

Given this brief overview of the data, the rationale behind choosing a combined architecture with mean shifting becomes evident. When we look at the sample range of the data, it is clear why we dismissed early escape architecture, that would need around 28000 states just to model Drosophila melanogaster dataset. Instead we opted for more complex combined architecture over basic self-loop architecture.

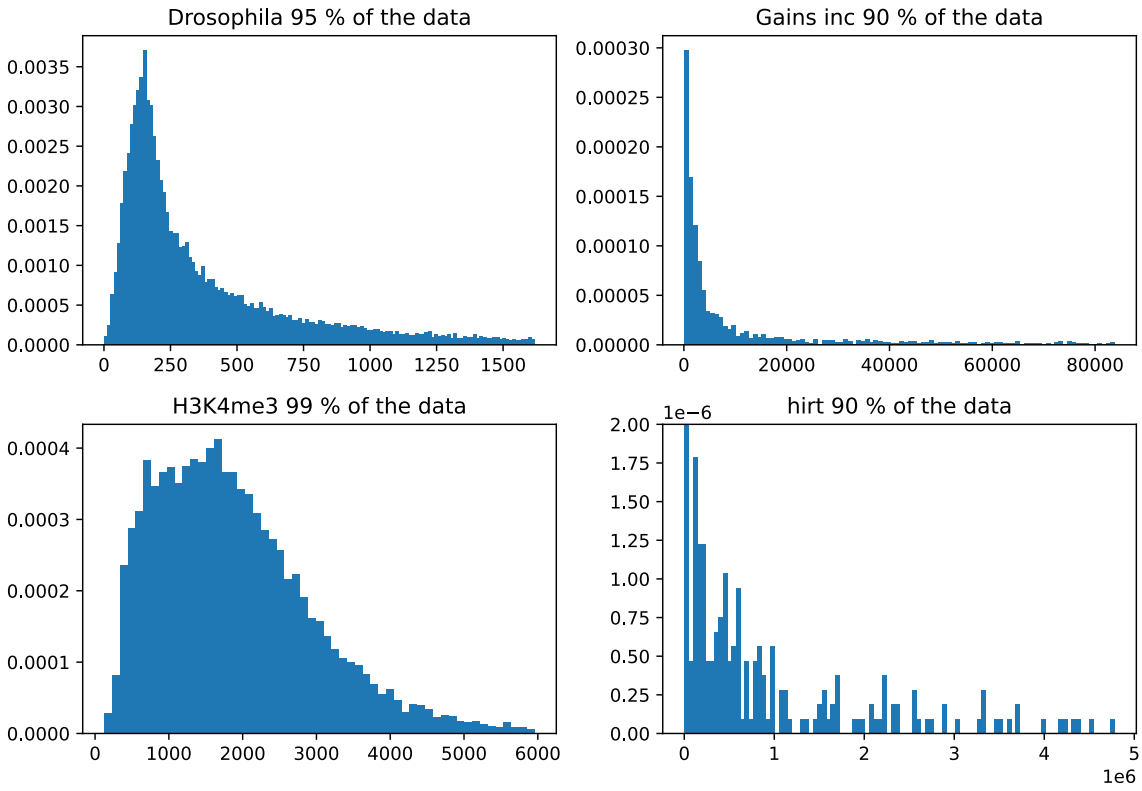

Figure 4.1: Histograms of interval lengths : Drosophila exons (top left), Gains inc (top right), H3K4me3 (bottom left), Hirt (bottom right). We had to opt for visualizing only some percentage of the data, so that the shape of data is visible.

In synthetic challenges datasets with a mean around 350 combined architecture required mean shifting. Here we work with significantly larger dataset means, therefore mean shifting is necessary.

In Figure 4.1 we show histograms of annotations lengths for each of the selected datasets.

4.3.1 Performance evaluation

In Figure 4.2 we can observe that we achieved very good cross-entropy in both Drosophila and H3K4me datasets. However contrary our intuition in Drosophila dataset the cross-entropy doesn't show a consistent decrease with the addition of states; instead, it fluctuates. This can be caused by the representative quality of sample chosen for specific training. This hypothesis is supported by the fact that the H3K4me3 dataset, which contains 19358 samples, shows a consistent decrease in cross-entropy compared to the Drosophila dataset of 188467 samples. Nonetheless, we can still observe some decreasing trend when adding states in Drosophila dataset.

In the hirt dataset, we observe difficulties in achieving a cross-entropy improvement

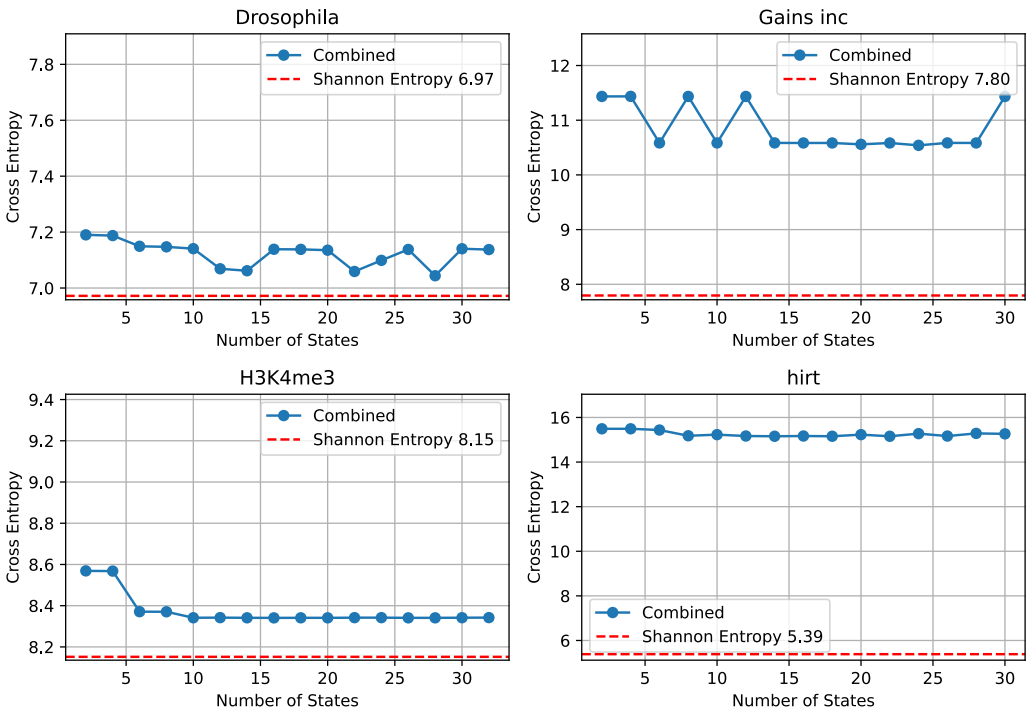

Figure 4.2: Performance curve for each dataset of annotation lengths trained on combined architecture with mean shifting. Cross-entropy in blue, Shannon entropy red.

below approximately 15, even with an increase in the number of states. This limitation may come from the dataset's characteristics: although the hirt dataset contains of a broad range of samples, it consists of only 247 samples in total. The ratio of samples to sample range and mean may block our ability to achieve better training outcomes.

Similar characteristic in ratio of samples to sample range can be observed also in Gains inc, where the cross-entropy does not significantly improve beyond 10.5.

In the H3K4me3 dataset an elbow-shaped pattern is noticeable in the cross-entropy line. Initially, adding states significantly improves cross-entropy, then it stops. This trend is especially clear at 6 states. This discovery is significant because the elbow point on the cross-entropy line indicates the number of states that first achieve acceptable cross-entropy with the fewest states, thus representing the least complex model with acceptable cross-entropy. This characteristic of the cross-entropy line can later guide model complexity selection.

In Figure 4.3, we present the best-trained models for each dataset. We observe that both the Drosophila and H3K4me3 datasets achieved very good cross-entropy with 28 and 16 states, respectively. This outcome is significant considering the scale of means in both datasets and the accurate replication of data shape.

In Gains inc dataset, we achieved acceptable cross-entropy, resulting phase-type distribution also adequately mimics the data shape.

We performed less effectively on the hirt dataset compared to the others, due to

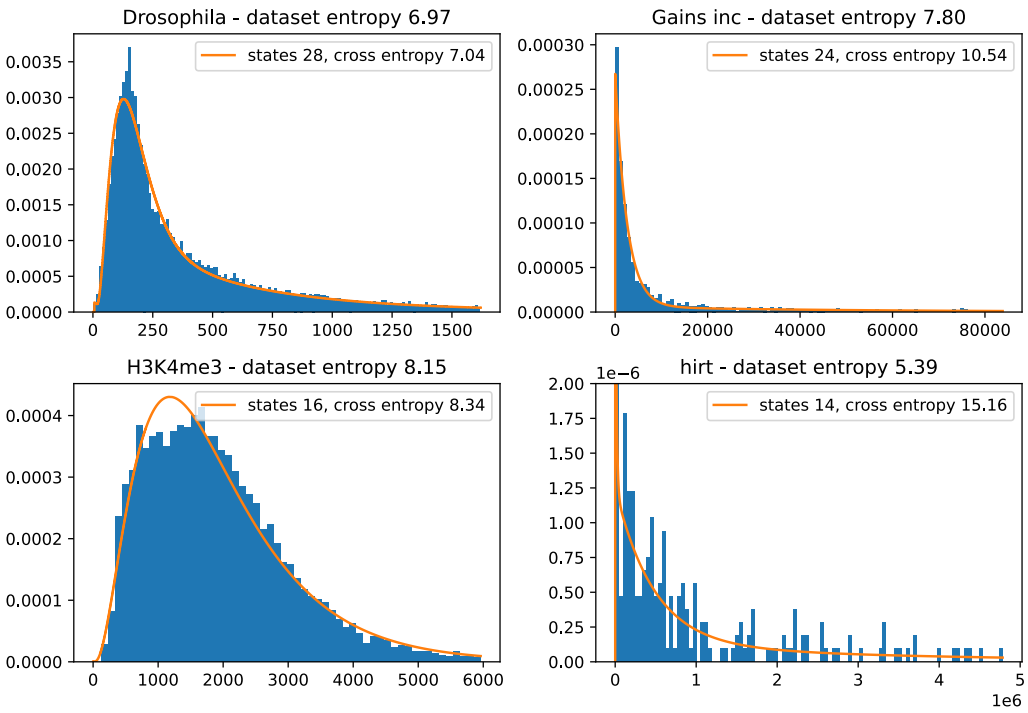

Figure 4.3: Best training for each dataset of interval lengths trained with combined architecture with mean shifting. Each figure shows density histogram of training data, in blue and learned phase-type distribution in orange.

the reasons mentioned before.

4.4 Gaps

Firstly we present a brief overview of the gaps datasets in Figure 4.2.

Dataset	Mean Gap Size	Range of Gap Lengths	Total Samples
Drosophila exons	1587	1–710734	63948
Gains inc	952046	4–30821993	3132
H3K4me3	156902	134–30542789	19333
Hirt	16296782	92740–87181600	116

Table 4.2: Characteristic of Datasets of interval gaps.

From the overview it is again clear why we opted for combined architecture with mean shifting. In Figure 4.4 we present histograms of annotation gaps form each dataset. We again had to opt for visualizing only some percentage of the data, so that the shape of data is visible.

4.4.1 Performance evaluation

In Figure 4.5 we see that overall, we were less successful in modeling gaps than interval lengths.

We were most successful in the Drosophila gaps dataset, where we also identify an elbow-shaped pattern in the cross-entropy line.

In the hirt gaps dataset, significant jumps from 20 to 45 in cross-entropy occur, notice that the cross-entropy remains unchanged at 45. This observation again suggests our failure to train this model. Instances where cross-entropy reaches approximately 20 can be attributed to fortuitous initial guesses rather than robust training. Here, we are approaching the threshold of what remains within the scope of model-ability, since the finding any solution (which still may be far from optimal one), depends heavily on initialization.

In both Gains inc gaps dataset and H3K4me3 gaps dataset, cross-entropy does not significantly improve when adding states. We can see that we somehow managed to train on these datasets, however probably reached the model capability.

In Figure 4.6 we present best trained model for each dataset. In Drosophila gaps dataset we can see that while achieving acceptable cross-entropy at 22 states, the phase-type distribution does not mimic the shape of data well.

In both Gains inc dataset and H3K4me3 dataset the resulting phase-type distribution mimics the shape of data well, however the achieved cross-entropy is not as good. In Gains inc dataset we achieved the best cross-entropy an 12 states which is surprisingly low given the sample range and mean size.

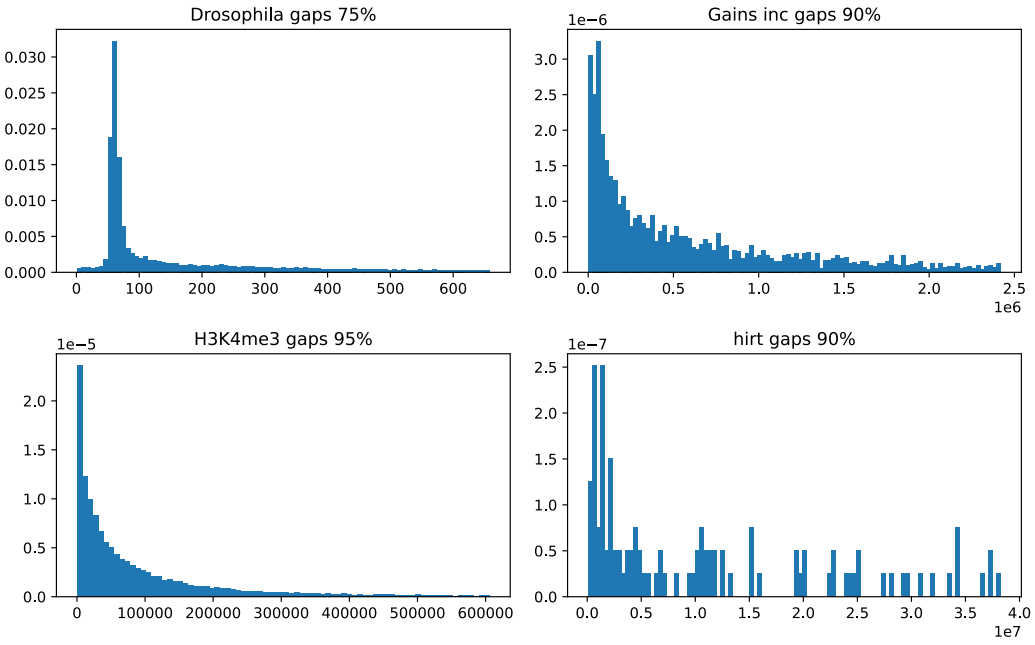

Figure 4.4: Histograms of annotation gaps : Drosophila exons (top left), Gains inc (top right), H3K4me3 (bottom left), Hirt (bottom right)

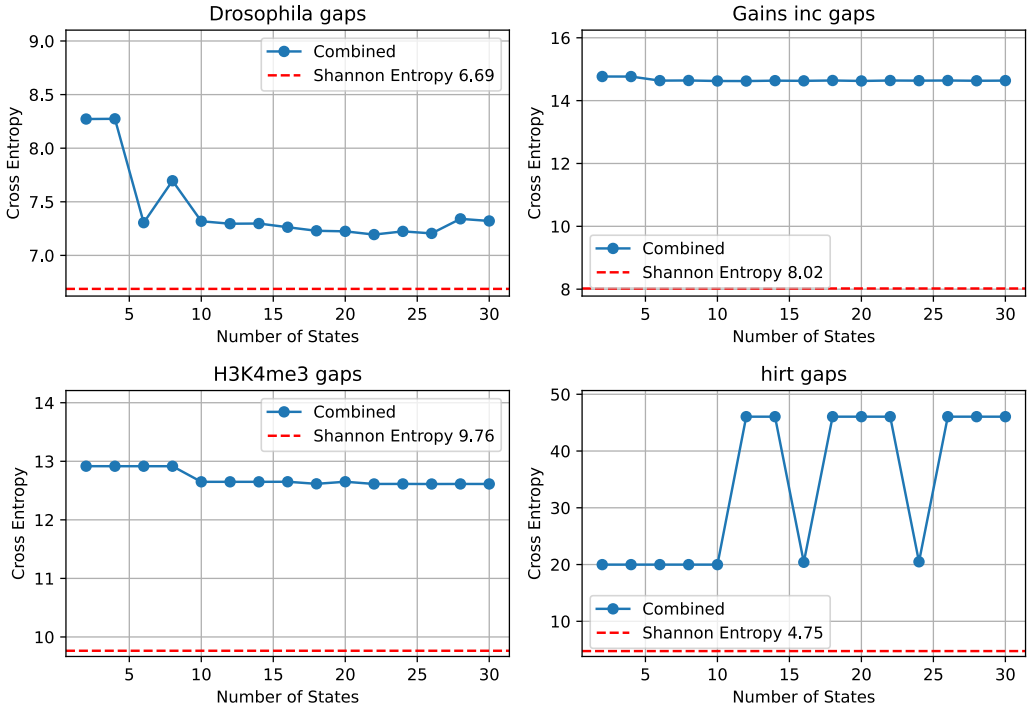

Figure 4.5: Performance curve for each dataset of annotation gaps trained on combined architecture with mean shifting. Cross-entropy in blue, Shannon entropy red, previous best achieved cross-entropy in orange.

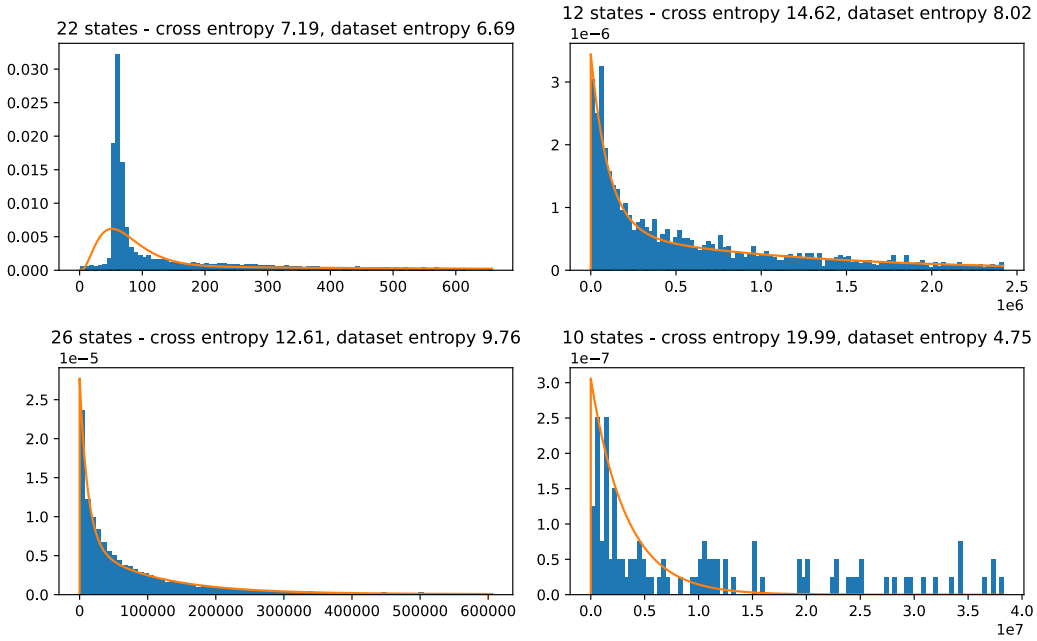

Figure 4.6: Best training for each dataset of annotations gaps trained with combined architecture with mean shifting. Each figure shows density histogram of training data, in blue and learned phase-type distribution in orange.

In hirt gaps we achieved the worst cross-entropy 19.99 to 4.75 of Shannon entropy of the dataset. We can observe that the resulting phase-type distribution only approximates the shape of the data, not fitting it correctly.

4.5 Conclusion

Overall, we were more successful modeling annotation lengths than gaps. We can say, that the characteristic of the dataset has huge impact on the resulting model accuracy. First of all the number of samples in dataset influence accuracy, secondly the sample range and mean of the data are significant. This can be observed especially in the hirt dataset, that performed worst in both interval lengths and gap lengths, having the biggest mean and the least amount of samples.

We can also conclude that the sampling technique sometimes decreases the precision of model fit especially in big datasets where the chosen sample may not be that representative of whole model, since it is chosen at random.

Chapter 5

Extending MCDP algorithm

In this chapter we will present updated MCDP algorithm, modified to work with our trained models, and we will show computed results. Firstly we present modified algorithm and then we show our results.

5.1 Description of the algorithm and its extension

In our task, the input consists of two annotations: one called reference R and the other called query Q. For clarity, let $R = \{[e_1, b_1), [e_2, b_2), \ldots, [e_m, b_m)\}$ and $|R| = m$, $|Q| = n$. The original MCDP algorithm, described in [7], aimed to approximate the *Gold standard null hypothesis*. According to this hypothesis, R is fixed, and Q is chosen among all possible rearrangements, taking into account the structure of R and the length of intervals in Q. However, computing the p-value under this hypothesis is NP-hard [7]. The p-value can be computed by sampling.

The MCDP proposes a Markov chain null hypothesis that Q is generated by a two-state Markov chain. The task is to determine the likelihood that an annotation generated from a Markov chain will intersect with a given reference annotation. A Markov chain is fitted to the query annotation. Let $R = \{[e_1, b_1), [e_2, b_2), \ldots, [e_m, b_m)\}$ the output is the probability that this Markov chain-generated annotation will hit at least k intervals in the reference, for each k from 0 to $|R|$.

Our extension lies in replacing two original states of Markov chain from MCDP with two Absorbing Markov Chain (AMC) (A, B), one trained on gap the other on interval distribution. This is achieved by connecting all transitions to absorbing state of one AMC to initial state of the other and vice versa (see Figure 5.1), forming a new transition matrix T (see section 1.4).

Transition matrices of both AMC P_A and P_B are in normal form (see subsec-

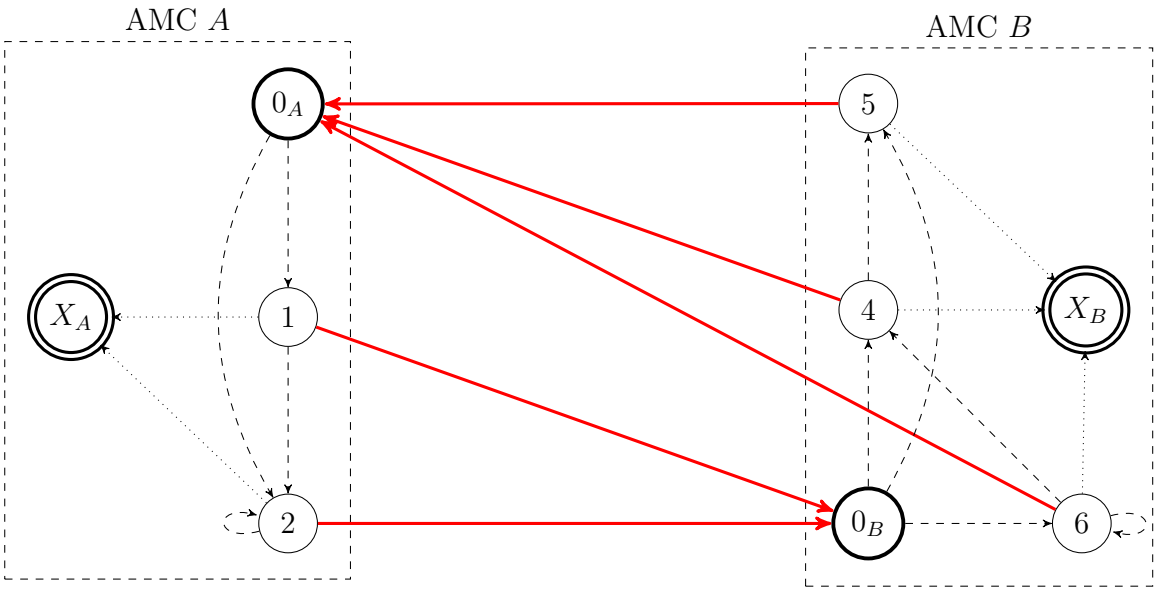

Figure 5.1: Visualization of the concatenation of two AMCs. States 0_A and 0_B, being initial states, and X_A and X_B being absorbing states, in AMC A and AMC B, respectively. The absorbing transitions from states 1 and 2 have been redirected to 0_B, similarly for states 4, 5 and 6 in AMC B.

tion 1.3.1 for definition). Then the transition matrix T is defined as:

$$T = \left(\begin{array}{c|c|c} Q_A & R_A & \mathbf{0} \\ \hline R_B & \mathbf{0} & Q_B \end{array}\right)$$

5.1.1 Dynamic programming

The subsequent subsection explains the extension of the MCDP algorithm as presented in [7], with the original notation preserved. MCDP computes $\Pr[\#\text{overlaps} = k]$.

The authors proposed an algorithm based on the dynamic programming paradigm. Let's define $P[j, k, s]$ as probability of hitting k reference intervals, after generating e_j states of Markov chain, with s being the last generated state. This probability admits a recursive relationship, based on the following idea: To hit k out of j intervals, we either hit k out of $j-1$ intervals and do not hit the j-th interval, or we hit $k-1$ out of $j-1$ intervals and hit the j-th interval. Formally, this recursive relationship looks as follows:

$$\begin{aligned} P[j,k,s] = &\sum_{\text{state} \in I_T} P[j, k-1, \text{state}] \cdot P_{\text{nohit}}(j, \text{state}, s) + \\ &+ \sum_{\text{state} \in I_T} P[j-1, k-1, \text{state}] \cdot P_{\text{hit}}(j, \text{state}, s) \end{aligned}$$

Ultimately, we need to compute the probabilities $P_{\text{hit}}(j, \text{state}, s)$ and $P_{\text{nohit}}(j, \text{state}, s)$.

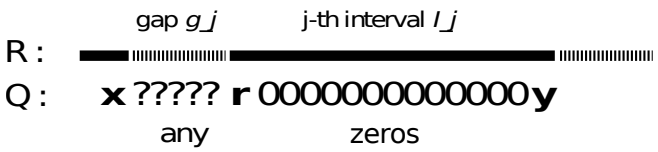

Figure 5.2: No hit for the j-th interval. Before the j-th interval a gap of length g_j precedes. At the end of the $(j-1)$-th interval, the Markov chain is in state x, transitioning to state r at the start of the j-th interval, and ultimately reaching state y.

Let's define $\Psi_{\text{any}}\left(x \xrightarrow{a} y\right)$ as probability of getting form sate x to state y in a steps, while generating anything, and $\Psi_{\text{zeros}}\left(x \xrightarrow{a} y\right)$ as probability of getting form sate x to state y in a steps, while generating only 0. This specifically allows for transitions that exclusively visit states in the AMC designated for gaps. We can express the probability of not hitting interval j as follows.

To avoid hitting the j-th interval, the Markov chain can generate any symbol during the gap, transitioning from state x to r for g_j steps. Then, it must only generate zeros during the interval, transitioning from state r to state y for I_j steps. Formally the probabilities are computed as follows.

$$P_{\text{nohit}}(j,x,y) = \sum_{\text{state}\in I_T} \Psi_{\text{any}}\left(x \xrightarrow{g_j} \text{state}\right) \cdot \Psi_{\text{zeros}}\left(\text{state} \xrightarrow{I_j} y\right)$$

$$P_{\text{hit}}(j,x,y) = \Psi_{\text{any}}\left(x \xrightarrow{g_j+I_j} y\right) - P_{\text{nohit}}(j,x,y)$$

We can compute function Ψ by exponentiating the transition matrix as follows (see section 1.3):

$$T^a = \begin{bmatrix} \Psi_{\text{any}}\left(0 \xrightarrow{a} 0\right) & \Psi_{\text{any}}\left(0 \xrightarrow{a} 1\right) & \dots & \Psi_{\text{any}}\left(0 \xrightarrow{a} n\right) \\ \Psi_{\text{any}}\left(1 \xrightarrow{a} 0\right) & \Psi_{\text{any}}\left(1 \xrightarrow{a} 1\right) & \dots & \Psi_{\text{any}}\left(1 \xrightarrow{a} n\right) \\ \vdots & \vdots & \ddots & \vdots \\ \Psi_{\text{any}}\left(n \xrightarrow{a} 0\right) & \Psi_{\text{any}}\left(n \xrightarrow{a} 1\right) & \dots & \Psi_{\text{any}}\left(n \xrightarrow{a} n\right) \end{bmatrix}$$

Now, we aim to model the same scenario while restricting to generating only 0. This can be effectively achieved by prohibiting all transitions to the Markov chain that represents intervals, essentially setting those transition probabilities to zero. We define modified transition matrix

$$D = \left(\begin{array}{c|c} Q_A & \mathbf{0} \\ \hline R_B & \mathbf{0} \end{array}\right)$$

Then, from Markov chain property follows that (x,y)-th entry of D^a specifies $\Psi_{\text{zeros}}\left(x \xrightarrow{a} y\right)$.

$$D^a = \begin{bmatrix} \Psi_{\text{zeros}}\left(0 \xrightarrow{a} 0\right) & \Psi_{\text{zeros}}\left(0 \xrightarrow{a} 1\right) & \dots & \Psi_{\text{zeros}}\left(0 \xrightarrow{a} n\right) \\ \Psi_{\text{zeros}}\left(1 \xrightarrow{a} 0\right) & \Psi_{\text{zeros}}\left(1 \xrightarrow{a} 1\right) & \dots & \Psi_{\text{zeros}}\left(1 \xrightarrow{a} n\right) \\ \vdots & \vdots & \ddots & \vdots \\ \Psi_{\text{zeros}}\left(n \xrightarrow{a} 0\right) & \Psi_{\text{zeros}}\left(n \xrightarrow{a} 1\right) & \dots & \Psi_{\text{zeros}}\left(n \xrightarrow{a} n\right) \end{bmatrix}$$

Computing the dynamic table P_{DP} breaks down to evaluating the $P_{\text{hit}}(j, x, y)$ and $P_{\text{nohit}}(j, x, y)$. Both can be effectively expressed as matrix multiplication:

$$P_{\text{nohit}}(j, x, y) = T^{g_j} D^{I_j}$$
$$P_{\text{hit}}(j, x, y) = T^{g_j + I_j} - T^{g_j} D^{I_j} = T^{g_j}(T^{I_j} - D^{I_j})$$

The original MCDP operates in $O(m^2 + n)$ time and requires $O(m)$ memory, where $m = |R|$ and $n = |Q|$. This time complexity breakdown consists of $O(m^2)$ for the dynamic table and $O(n)$ for computing the transition matrix. To effectively compute each row of the P_{DP} table in our modified program, we require evaluation of matrices T and D raised to the power of a. Using the fast matrix exponentiation algorithm (see section 1.3), we achieve this in $O(\log a)$ matrix multiplications. Our modified then program operates in $O(m^2 \log a)$ time, where $a = \max(g_j, I_j | 0 < j \leq m)$. Since we use precomputed transition matrices, we can omit the $O(n)$ complexity from the original MCDP. It runs in $O(m + s^2)$ memory, where s is the size of the transition matrix.

5.1.2 Normalization

During the implementation of our algorithm, we encountered overflow issues due to raising small transition probabilities to large powers, which resulted in a loss of numerical precision. Therefore, after each exponentiation of the matrix T, we applied a normalization step to ensure that the elements in each row sum up to 1.

5.2 Results

We run our model on two examples from the original MCDP article with the hirt and Gains datasets as queries. These datasets are chosen because they are the smallest ones, hence the simulation will be the fastest. We used our best trained AMC for both gaps and intervals (see Chapter 4). The results are shown in Figure 5.3.

The computation for hirt dataset ran in approximately a minute, the Gains dataset took around one hour to complete. This is due to the significant difference in the dataset sizes.

Surprisingly, despite our anticipation of more accurate results from our extended model due to its complexity and ability to capture query structure more precisely, we

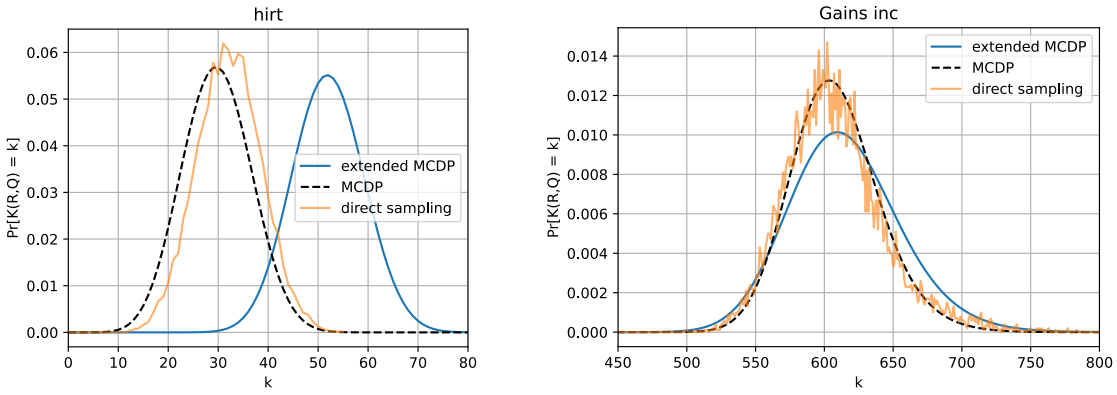

Figure 5.3: P-values comparison of MCDP, direct sampling and extended MCDP for hirt and Gains queries.

observed the opposite outcome. The simpler MCDP proved to be more accurate in modeling direct sampling from the gold null hypothesis, compared to our extended version.

The implementation of our extended model is available in the digital attachment.

Chapter 6

Implementation details

In this chapter, we will discuss implementation details and describe the tools used for optimisation, data visualization, and result analysis. We will cover the general tools employed, the setup for fitting and evaluation (including the optimisation algorithm and initialization), and some techniques used in modeling.

6.1 General used tools

We run our optimisation in *Python* [31]. For visualization we used *matplotlib* [16] library. We used *Snakemake* [21] for pipelineing the evaluation of the synthetic challenges. For experimental and early stages of development we used Jupyter notebooks [18]. We used the pandas library [20] for data processing. For the visualization of the final trained models we used *graphonline* [26]. The experiments were conducted on a server with Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz $\times$ 48 and 128 GB DDR RAM.

6.2 General setup of the fitting and evaluation

6.2.1 Objective function

As discussed in Section 1.4.2, our objective function is the cross-entropy, which is computed as the negative log-likelihood derived from the data likelihoods. In the process of calculating the log-likelihood of the data, we add a small constant, denoted as $\epsilon = 1 \times 10^{-20}$, to prevent division by zero in cases where the resulting data probabilities become extremely small. This adjustment ensures numerical stability during the computation.

6.2.2 Optimisation algorithm

We chose the L-BFGS-B optimisation algorithm, for multiple reasons. It is deterministic, therefore we can replicate the results with the same initialization, if needed. It is bounds constrained which is useful since we want to optimize probabilities which are of range 0 to 1. Also we ran a test on all bound constrained algorithms and L-BFGS-B performed the best for our specific task.

Bounds setting

Bounds are user defined input for the L-BFGS-B algorithm. All our experiments were run with bounds $(-15, 15)$ (this value goes into softmax to transform it to probability $(0, 1)$).

In our experiment we encountered cases, when changing the bounds for specific task, had an impact on optimisation. For example self-loop architecture on small value Gaussian challenge, trained better with bounds $(-2, 2)$. However, since we aimed at automatizing the optimisation, we decided to set the bound uniformly for all architectures and all challenges. We found that bounds $(-15, 15)$ provide a wide enough range to fit all our challenges.

Realize that computing the maximal mean of resulting phase-type distribution is deterministic for given bounds, architecture and number of states. This can be done through setting the back-loops and self-loops to maximum and transition probabilities to absorption state to minimum. Corresponding transition matrix is then easily constructed with knowledge of the architecture and number of states. Computing the mean is then done by matrix operations described in subsection 1.3.1.

6.2.3 Initialization

The L-BFGS-B algorithm requires an *initial guess* for parameter values. Right at the beginning of our tests, we found out that the correct initialization of parameters has a substantial impact on the resulting cross-entropy. We tried multiple approaches, when setting initial parameter weights.

Random initialization with multiple runs

Firstly, we decided to set each parameter at a random value from interval $(-1, 1)$, as our initial guess. However, this random initialization has an impact on optimisation results and the final model cross-entropy. To mitigate the impact of randomization on results, we run each optimisation 10 times, each time with different random initialization and then we proceed to pick the one with best cross-entropy.

Mean shifting

However, as described in subsection 2.5.2 when training with big samples, e.g. data mean around 300 or bigger, random initialization was not sufficient.

We developed a technique to pre-process the initial guess so that the optimisation continues, also on datasets consisting of large values. The technique described in subsection 2.5.2 is called *mean shifting*. Where we first shift the mass of the phase-type distribution closer to the mean of the data distribution, and then we proceed to train there.

The mean shifting technique consists of two phases. Firstly, we do a pre-processing optimisation, where we shift the mean of the phase-type distribution close to the data mean. This is done by optimizing objective function dependent on the difference of means of these distributions.

The pre-processing optimisation naturally also requires initial parameter values, we initialize these at random from interval $(-1, 1)$.

The result of this pre-processing is then used as initialization for objective function. User of our software tool can specify whether to train with mean shifting or without.

Adding noise

However, we would still like to keep the idea of running multiple optimisations with different initialization, and then proceed to pick the one that gives the best cross-entropy. We implemented this idea, by running the mean shifting optimisation, and then applying small noise to the final result.

The noise we applied in our experiments was Gaussian noise with mean 0 and standard deviation 0.1. We then proceed to optimize 10 times with output from mean shifting pre-processing and with 9 initialization with added Gaussian noise. Then we proceed to pick the most optimal one from these results.

However, this process is still heavily dependent on the initial guess, going to mean-shifting pre-processing. To address this, we run the entire process 10 times and select the best result. This approach helps mitigate the impact of random initialization on the final outcome.

Parallelization

Finally, since we run the same optimisation subroutine multiple times, just with different random initialization, we decided to parallelize the process to speed up the training. We used python multiprocessing library [31] to achieve this.

User of our software tool can specify how many threads will be used for training as well as number of iterations for each training.

6.2.4 Sub-sampling

In chapter 4, where we modeled annotations from real datasets, we described a sampling technique (see section 4.1). In chapter 4 we run all experiments with sample size 2000. To generate samples, we utilized the numpy library [12] random.choice function without replacement. For each optimisation run new random sample is generated. In chapters 3 and 2 were experiments run without sampling.

Sample size is a parameter user can define when running our software tool, to control the speed and training precision.

Conclusion

In this thesis, we developed a software tool for modeling genomic annotations using phase-type distributions. Our approach involved examining various architectures of absorbing Markov chains and evaluating their theoretical power and quality of fit. The *combined architecture with mean shifting* was selected as the most suitable model due to its flexibility and accuracy when tested on synthetic challenges.

We extended the MCDP model to incorporate absorbing Markov chains, which enabled more precise computation of p-values for observed overlaps in genomic annotations. This extension addressed the issue of the original tool modeling gap and interval lengths being geometric distributions.

Throughout the thesis, we addressed the necessity of accurately modeling gap and interval length distributions. Phase-type distributions proved to be an effective solution, and our fitting process, which minimized cross-entropy by optimizing transition probabilities, ensured high-quality fits for complex genomic data.

We explored various absorbing Markov chain architectures, discussing the trade-offs between density, theoretical power, and performance. We proposed four synthetic challenges to evaluate the architecture performance. Additionally, we introduced the performance curve as a model selection method, depicting the achieved cross-entropy vs. the number of states for each architecture.

We examined three basic architectures: *self-loop*, *early-escape*, and *combined* architectures. Additionally, we introduced the *mean shifting* technique to enhance the training of the combined architecture on datasets with large values. Then we introduced three dense architectures: *fully connected*, *pruned*, and *k-jumps* architectures and tested them against set of the synthetic challenges.

Based on the performance on synthetic challenges, we concluded that the combined architecture with mean shifting is most suitable for modeling real genomic annotations. We also introduced a sampling technique to speed up the training process on big datasets. We then trained the combined architecture with mean shifting on four genomic datasets. Our results indicated that interval lengths were modeled more effectively than gaps, and dataset characteristics such as mean and sample size significantly impacted the quality of fit.

Finally, we integrated our findings into the MCDP model and compared our results

with the original MCDP and permutation tests. This comprehensive evaluation underscored the advantages of our approach in genomic data analysis, providing a more accurate and robust method for modeling genomic annotations.

In conclusion, our work presents an approach to genomic annotation modeling that utilizes phase-type distribution. Future research could further enhance the fitting by implementing different loss functions, such as the earth mover's distance, to improve performance even further.

Bibliography

[1] Hansjörg Albrecher, Martin Bladt, Mogens Bladt, and Jorge Yslas. Continuous scaled phase-type distributions. *Stochastic Models*, 39(2):293–322, 2023.

[2] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.

[3] T Britannica. Editors of encyclopaedia. *Argon. Encyclopedia Britannica*, 2020.

[4] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.

[5] Wai-Ki Ching and Michael K Ng. Markov chains. *Models, algorithms and applications*, 2006.

[6] David Ellerman. *New foundations for information theory: logical entropy and Shannon entropy*. Springer Nature, 2021.

[7] Askar Gafurov, Broňa Brejová, and Paul Medvedev. Markov chains improve the significance computation of overlapping genome annotations. *Bioinformatics*, 38(Supplement_1):i203–i211, 2022.

[8] Askar Gafurov, Tomáš Vinař, Paul Medvedev, and Broňa Brejová. Efficient analysis of annotation colocalization accounting for genomic contexts. In *International Conference on Research in Computational Molecular Biology*, pages 38–53. Springer, 2024.

[9] Bernat Gel, Anna Díez-Villanueva, Eduard Serra, Marcus Buschbeck, Miguel A. Peinado, and Roberto Malinverni. regioneR: an R/Bioconductor package for the association analysis of genomic regions based on permutation tests. *Bioinformatics*, 32(2):289–291, 2015.

[10] Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 1997.

[11] Matthew G Guenther, Stuart S Levine, Laurie A Boyer, Rudolf Jaenisch, and Richard A Young. A chromatin landmark and transcription initiation at most promoters in human cells. *Cell*, 130(1):77–88, 2007.

[12] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

[13] Andreas Heger, Caleb Webber, Martin Goodson, Chris P. Ponting, and Gerton Lunter. GAT: a simulation framework for testing the association of genomic intervals. *Bioinformatics*, 29(16):2046–2048, 2013.

[14] Anne Marie Helmenstine. What is the null hypothesis? definition and examples, 2019.

[15] András Horváth and Miklós Telek. Phfit: A general phase-type fitting tool. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 82–91. Springer, 2002.

[16] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[17] C. Isensee and G. Horton. Approximation of discrete phase-type distributions. In *38th Annual Simulation Symposium*, pages 99–106, 2005.

[18] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[19] Alexander Koronen. Mean and variance of absorption time in a population genetics model: Comparison between Markov chain and diffusion process methods. *Master's Theses in Mathematica Sciences*, 2019.

[20] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[21] Felix Mölder, Kim Philipp Jablonski, Brice Letcher, Michael B Hall, Christopher H Tomkins-Tinch, Vanessa Sochat, Jan Forster, Soohyun Lee, Sven O Twardziok, Alexander Kanitz, et al. Sustainable data analysis with Snakemake. *F1000Research*, 10, 2021.

[22] Luis R Nassar, Galt P Barber, Anna Benet-Pagès, Jonathan Casper, Hiram Clawson, Mark Diekhans, Clay Fischer, Jairo Navarro Gonzalez, Angie S Hinrichs, Brian T Lee, et al. The ucsc genome browser database: 2023 update. *Nucleic acids research*, 51(D1):D1188–D1195, 2023.

[23] James R Norris. *Markov chains*. Cambridge university press, 1998.

[24] Athanasios Papoulis and S Unnikrishna Pillai. *Probability, random variables, and stochastic processes*. McGraw-Hill Europe: New York, NY, USA, 2002.

[25] Shahab Sarmashghi and Vineet Bafna. Computing the statistical significance of overlap between genome annotations with istat. *Cell systems*, 8(6):523–529, 2019.

[26] Oleg Shiakhatarov. graphonline. `https://graphonline.ru/en/`, 2019.

[27] Saša Singer and John Nelder. Nelder-mead algorithm. *Scholarpedia*, 4(7):2928, 2009.

[28] Joan Stephenson. Talking genetics glossary. *JAMA*, 281(7):600–600, 1999.

[29] Anders Tolver. An introduction to markov chains. *Department of Mathematical Sciences, University of Copenhagen*, 2016.

[30] Kristen M Turner, Viraj Deshpande, Doruk Beyter, Tomoyuki Koga, Jessica Rusert, Catherine Lee, Bin Li, Karen Arden, Bing Ren, David A Nathanson, et al. Extrachromosomal oncogene amplification drives tumour evolution and genetic heterogeneity. *Nature*, 543(7643):122–125, 2017.

[31] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[32] Mehdi Zarrei, Jeffrey R MacDonald, Daniele Merico, and Stephen W Scherer. A copy number variation map of the human genome. *Nature reviews genetics*, 16(3):172–183, 2015.

[33] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560, 1997.

Appendix A: content of the electronic appendix

In the electronic appendix accompanying the thesis, you will find the source code of the software tool for fitting genomic annotations, the source code for the extended MCDP program, and data files for reproducibility of the experiments. The source code for the genomic annotations tool is publicly available on the GitHub page `https://github.com/HanaDerkova/bakalarka`, and the source code for the extended MCDP program is publicly available on the GitHub page `https://github.com/HanaDerkova/extended-MCDP`. The specific usage of both tools is further described in their respective README.md files.