Efficient stochastic simulation on discrete spaces

Using balancing functions to incorporate local target density information into Markov Chain Monte Carlo sampling schemes

Bjarne Herben

A thesis presented to obtain the degree of
Bachelor of Science

Student Number: 5163005
Supervisor: Frank van der Meulen
Project Duration: April, 2022 - July, 2022
To be defended publicly on July 12, 2022 at 10 : 00 AM
Faculty of Electrical Engineering, Mathematics and Computer Science

Efficient stochastic simulation on discrete spaces

Using balancing functions to incorporate local target density information into Markov Chain Monte Carlo sampling schemes

Bjarne Herben

Impact statement

Experiments are performed to find out how things work. Similarly, by simulating a random process many times we can learn a lot about its properties. Essentially, simulating a stochastic process is like throwing a very complicated dice. The fact that we perform these simulations on a discrete space merely means that we can count the number of possible outcomes of the process using the natural numbers $\{1, 2, 3, \dots\}$. However, suppose now that, unlike throwing a dice, we are unable to directly simulate the process that we want to study. Fortunately, we can often apply Markov Chain Monte Carlo sampling schemes to still be able to simulate the process of interest. The only downside is now that the simulations take a lot longer to perform, since we randomly propose an outcome for the "experiment" that we want perform; we are only rarely able to obtain a valid experimental result. In this work, we consider two ways to use a piece of mathematical machinery to make the "experiments" that we want to do significantly faster as we are using the knowledge that we have about the process more efficiently. We check both approaches to see how it can help us to merge two databases containing records, where we want to avoid duplicate recorded in the merged database.

Efficient stochastic simulation on discrete spaces

Using balancing functions to incorporate local target density information into Markov Chain Monte Carlo sampling schemes

Bjarne Herben

Abstract

The breadth of theoretical results on efficient Markov Chain Monte Carlo (MCMC) sampling schemes on discrete spaces is slim when compared to the available theory for MCMC sampling schemes on continuous spaces. Nonetheless, in [Zan17] a simple framework to design Metropolis-Hastings (MH) proposal kernels that incorporate local information about the target is presented. The class of functions for which the resulting MH kernels are Peskun optimal in high-dimensional regimes is characterized. We will refer to these functions as *balancing functions* and to the class of resulting MH proposal kernels as *pointwise informed proposals*. In [PG19], the class of balancing functions is used to construct Markov Jump Processes (MJP) on discrete state spaces. As a result, the Zanella process is constructed. In the absence of a theoretical result on the optimal balancing function to choose from the class of balancing functions, a heuristic approach is proposed using the Zanella process. To further encourage the mixing behaviour of the simulated chain, the algebraic structure of the state space is exploited to achieve non-reversible Markov chains on short to medium timescales. Simulations are performed for all the considered MCMC sampling schemes by studying the Bayesian record linkage problem.

Contents

Chapter 1

Introduction

The efficient simulation of stochastic processes in continuous state spaces has been an active area of research. However, on discrete state spaces there are not nearly as much results on increasing the efficiency of Markov Chain Monte Carlo (MCMC) sampling schemes. This work explores the proposed framework in [Zan17] to incorporate local information about the target distribution into Markov Chain Monte Carlo sampling in the setting of discrete-valued high-dimensional parameters. A direct discrete time application is to create locally-informed proposal kernels that can be used in the Metropolis-Hastings algorithm.

In continuous time, this framework can also be utilized to a design a Markov Jump Process (MJP) that leaves the target distribution invariant as detailed in [PG19]. The jumping rates for this process are calculated by using local information about the target density. This is done in the Zanella sampler. Moreover, the algebraic structure of the state-space can be exploited to stimulate non-reversibility of the simulated Markov chain. This is essentially an extensions to the Zanella process and results in the Tabu sampler. These two samplers are compared in performance to the regular and locally-informed variation of the Metropolis-Hastings algorithm.

The aim is to compare the incorporation of local information about the target density for the four discrete and continuous time samplers. The comparison of these four samplers is conducted on three examples. Two of these examples are relatively simple, independent binary components and weighted permutations, and can be used to reflect on the performances for different properties of the target distribution. A more interesting example is the Bayesian record linkage problem, where the aim is two merge two databases containing duplicate records. MCMC sampling is then used to explore the posterior distribution on the possible ways in which these two databases can be matched, where we want to avoid the occurrence of duplicate records in the merged database. Note that we assume that duplicate records, i.e. two records that contain information on the same unique entity, only occur between the two databases and not within a database.

1.1 Motivation: Bayesian inference

The difficulty in finding a matching for these two databases is that noise is a part of every database record. However, when merging these two databases we wish to separate the true signal from the noise; uncover the entity to which the stored records refers. Hence, if two records refer to the same entity, then these should be entered as one record in the merged database. The difficulty in doing this is that we do not know exactly where noise has entered the database record. On the other hand, we are able to make reliable estimates about the rates at which noise occurs in the database. If we want to try and merge the two databases in the best possible way, then we need to use all the reliable information that we have. Fortunately, we also have full access to the two databases that we wish to merge. All of this information can be used by incorporating it into a statistical model. Typically, this statistical model describes the following connection, where with merged we mean the merged database and with observed the two databases that we want to merge,

$$\text{Merged} \xrightarrow{\text{Statistical model}} \text{Observed}. \tag{1.1}$$

However, when we want to merge the two databases we are actually interested in going the opposite direction. This is where Bayesian statistics could be a solution, since Bayesian statistics allows us

to relate the probability of the observed given the unobserved to the probability of the unobserved given the observed [Lam18].

$$P(\text{observed} \mid \text{unobserved}) \xrightarrow{\text{Bayesian statistics}} P(\text{unobserved} \mid \text{observed}) \tag{1.2}$$

In case we want to merge the two databases, we want to use Bayesian statistics in the following way

$$P(\text{observed} \mid \text{merged}) \xrightarrow{\text{Bayesian statistics}} P(\text{merged} \mid \text{observed}). \tag{1.3}$$

We can interpret $P(\text{merged database} \mid \text{observed databases})$ as how certain we are that a specific matching is correct if we are trying to match the observed databases. Naturally, we can then choose the merged database which is most likely to be correct. Therefore, Bayesian statistics in this setting is certainly useful, by connecting the two probabilities of interest in the following way

$$P(\text{merged} \mid \text{observed}) = \frac{P(\text{observed} \mid \text{merged})P(\text{merged})}{P(\text{observed})}, \tag{1.4}$$

where we call $P(\text{merged})$ the prior, since this term allows for incorporation of the prior knowledge that we have about the merged database. This prior is usually a probability distribution on the possible values of the parameter with most probability mass placed in a neighbourhood of the values that we consider plausible a priori. In addition, notice that given the two databases that we want to merge, we can consider $P(\textit{observed})$ a constant in this context. Therefore, we obtain the following distribution to observe,

$$P(\text{merged} \mid \text{observed}) \propto P(\text{observed} \mid \text{merged})P(\text{observed}). \tag{1.5}$$

Notice that the term $P(\text{observed} \mid \text{merged})P(\text{observed})$, also known as the joint distribution, is entirely determined by the specified statistical model, the observed information, and the prior knowledge that we have. Thus, we end up with a distribution that we know up to proportionality. In this case, MCMC is the most common sampling method used to obtain information about the posterior $P(\text{merged} \mid \text{observed})$; make Bayesian inferences about the merged database. This is precisely how we wish to apply MCMC to the Bayesian record linkage problem, for more details the reader is referred to chapter 6.

1.2 Outline of the thesis

In chapter 2 we give a brief theoretical overview of Monte Carlo methods, Markov chains, and Markov Chain Monte Carlo Methods. In addition, some employed diagnostics will be covered. The theory on using local-information in the Metropolis-Hastings algorithm as covered in [Zan17] is given in chapter 3. An extension of this framework to MJPs as presented in [PG19] is included in chapter 4, where also the Zanella and Tabu samplers will be covered. The independent binary components and weighted permutations examples are succinctly detailed in chapter 5. The Bayesian record linkage problem is extensively presented in chapter 6. The performance of the considered samplers on the covered examples is presented in chapter 7, where numerical simulations are performed.

Chapter 2

Markov Chain Monte Carlo

In this chapter we present a brief theoretical overview of Monte Carlo methods, Markov processes, and Markov chain Monte Carlo methods. We will denote the target distribution as π, and the corresponding probability measure as Π. Furthermore, we will assume the existence of a probability density (or probability mass function) unless specified otherwise, the probability density will be denoted by π. The aim with the notation is to keep it as general as possible, however, the theory for Markov chains on discrete state spaces is not overcomplicated. Therefore, we will on some occasions state results that are only applicable to Markov chains on discrete spaces.

2.1 Monte Carlo Methods

Monte Carlo methods are used to evaluate integrals of the following form [RC04],

$$I = \mathbb{E}_\pi[h(X)] = \int_\chi h(x)\pi(x)\,dx. \tag{2.1}$$

In this case, assume that $X \sim \pi$ is a random variable on $\mathcal{X}$. A Monte Carlo estimate $\hat{I}$ for Equation 2.1 can be obtained if we are able to generate iid replicas $X^{(i)} \sim \pi$. The Monte Carlo estimator is then given by

$$\hat{I} = \frac{1}{N}\sum_{i=1}^{N} h(X^{(i)}). \tag{2.2}$$

The Strong Law of Large Numbers [JP04] then ensures that

$$\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} h(X^{(i)}) = \int_\chi h(x)\pi(x)\,dx \text{ a.s. and in } L^2(\pi). \tag{2.3}$$

In addition, if $\int_\chi h^2(x)\pi(x)\,dx < \infty$, then we can apply the Central Limit Theorem (CLT) to the estimate to obtain asymptotic normality for the Monte Carlo estimator:

$$\frac{\sqrt{N}(\hat{I} - I)}{\sigma} \xrightarrow{d} \mathcal{N}(0,1), \tag{2.4}$$

with $\sigma^2 = \int_\chi h^2(x)\pi(x)\,dx - I^2$. This asymptotic normality can be used to derive confidence intervals for the estimate.

$$\|I - \hat{I}\| \leq c_{1-\alpha/2}$$

However, the caveat is that this confidence interval is only valid asymptotically.

Contrary to many numerical quadrature rules used to calculate integrals, the convergence rate of Monte Carlo methods is always $O(N^{-1/2})$, irregardless of the dimension of the space $\mathcal{X}$, merely under the assumptions that $\int_\mathcal{X} h(x)^2\pi(x)\,dx < \infty$ and that σ^2 remains bounded when the dimension of the space increases. This is indeed a good thing as the convergence of regular numerical quadrature schemes decrease as the dimension of the integral increases. Therefore, Monte Carlo methods are especially popular when it comes to the calculation of high-dimensional integrals.

In the context of Bayesian statistics, Monte Carlo methods can be used to estimate probabilities by considering expectations of the following form

$$\mathbb{E}_\pi[\mathbb{1}_{X\in A}] = P_\pi(X \in A). \tag{2.5}$$

2.2 Markov Processes

In this section, a brief overview on the theory of discrete time Markov chains on discrete spaces will be given.

2.2.1 Discrete time Markov chains

Consider a stochastic process $\{X_n : n \in \mathbb{N}_0\}$ defined on $\mathbb{N}_0 = \{0, 1, \dots\}$ and taking values on a discrete state space $\mathcal{X}$, which means that X is a finite or countable set. Denote by $\mathcal{B}(\mathcal{X})$ the Borel σ-algebra such that $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ is a measurable space.

Definition 2.1. *A stochastic process $\{X_n \in \mathcal{X} : n \in \mathbb{N}_0\}$ is a Markov chain if it satisfies the Markov property*

$$P(X_{n+1} = y_{n+1} \mid X_n = y_n, X_{n-1} = y_{n-1}, \dots, X_0 = y_0) = P(X_{n+1} \mid X_n = y_n) \tag{2.6}$$

for $y_0, \dots, y_{n+1} \in \mathcal{X}$.

Thus, a Markov process is a stochastic process for which the distribution of the next state only depends on the current position. A Markov process is even memoryless when we condition on a time that is a random variable, this property is called the Strong Markov property. However, before we can establish this we need the definition of a stopping time.

Definition 2.2. *(Stopping time). A random variable τ is called a stopping time if the event $\{\tau \leq n\}$ depends only on $X_0, \dots, X_n$, which means that the event $\{\tau \leq n\}$ is measurable with respect to the σ-algebra $\sigma(X_0, \dots, X_n)$ generated by $X_0, \dots, X_n$.*

Definition 2.3. *(Strong Markov property). Let τ be a stopping time of $\{X_n\}$. Conditional on $\tau < +\infty$ and $X_\tau = y_i \in \mathcal{X}$,*

$$P(X_{\tau+1} \mid X_\tau = y_i, X_{\tau-1} = y_{\tau-1}, \dots, X_0 = y_0) = P(X_{\tau+1} \mid X_\tau = y_i). \tag{2.7}$$

The Markov property implies that the probability distribution for the next state X_{n+1} depends only on the current state X_n, and possibly on the discrete time of the process n. If there is no dependence on n, then we say that a Markov chain is homogeneous. This means that for the transition probabilities from y_i to y_j there exists a constant $P_{i,j}$ such that

$$P_{i,j} = P(X_{n+1} = y_j \mid X_n = y_j),$$

for all $n \in \mathbb{N}_0$. Therefore, the transition probabilities only depend on the current state. From now on we will study homogeneous Markov chains, unless specified otherwise.

All the transition probabilities are contained in the Markov transition kernel P, for which it holds that $P_{i,j} = P(y_j, y_i)$. We formalize the properties of a Markov transition kernel below.

Definition 2.4. *A Markov transition kernel on $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ is a function $P : \mathcal{X} \times \mathcal{B}(\mathcal{X}) \to [0, 1]$ such that*

- *for all $y \in \mathcal{X}$, $P(y, \cdot)$ is a probability measure on $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$;*
- *for all $B \in \mathcal{B}(\mathcal{X})$, $P(\cdot, A)$ is a measurable function on $\mathcal{X}$.*

Usually when we speak about a Markov chain we denote it by (λ, P), where $X_0 \sim \lambda$ is a probability distribution of the starting state and P the Markov transition kernel. We do this as all the properties of a Markov chain are entirely determined by λ and P.

Consider now the Markov chain (λ, P) on the discrete state space $\mathcal{X}$. Suppose that we are interested in $P(X_n \in A \mid X_0 \sim \lambda)$, denoted by $P_\lambda(X_n \in A)$ for $A \in \mathcal{B}(\mathcal{X})$. If we want to calculate $P_\lambda(X_n \in A)$, then it is convenient to associate with the Markov transition kernel P the transition

operator $\mathcal{P} : \mathcal{M}_1(\mathcal{X}) \to \mathcal{M}_1(\mathcal{X})$, where $\mathcal{M}_1(\mathcal{X})$ is the set of all probability measures on $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ [Nob21]. Define the operator $\mathcal{P}$, working on the left of probability measures, in the following way

$$\mu = \lambda\mathcal{P} \Longrightarrow \mu(A) = \int_{\mathcal{X}} P(y, A)\, \lambda(dy). \qquad \forall A \in \mathcal{B}(\mathcal{X})$$

By applying this operator twice, the following distribution is obtained

$$\lambda\mathcal{P}^2 = (\lambda\mathcal{P})\mathcal{P} = \int_{\mathcal{X}} \int_{\mathcal{X}} P(x, \cdot)\mathcal{P}(y, dx)\, \lambda(dy) = \int_{\mathcal{X}} P^{(2)}(y, \cdot)\, \lambda(dy).$$

Hence, $\mathcal{P}^2$ refers to the operator associated to P^2 and, in the general case, $\mathcal{P}^n$ is the operator associated to $P^{(n)}$. If we now utilize this operator consideration, then we can denote the measure of X_n by $\pi^{n,\lambda}$, where $\pi^{n,\lambda}(A) = P_\lambda(X_n \in A)$. It follows that

$$\pi^{n,\lambda} = \lambda\mathcal{P}^n = \int_{\mathcal{X}} P^{(n)}(y, \cdot)\, \lambda(dy).$$

2.2.2 Convergence of Markov chains

The idea behind the convergence of Markov chains is that $\lim_{n\to\infty} \pi^{n,\lambda} = \pi$, for π a probability distribution on the discrete state space $\mathcal{X}$. A possible way to interpret this is by considering the operator $\mathcal{P}$ associated to the Markov transition kernel P. For a discrete time Markov chain, every time step essentially corresponds to applying the operator $\mathcal{P}$ to the probability distribution $\pi^{n,\lambda}$ on the possible states for the chain in $\mathcal{X}$ at time step n. In case of convergence, the rate with which the distribution $\pi^{n,\lambda}$ changes decreases and the distribution becomes more and similar to the limiting distribution π. However, if we want to speak about similarities between distributions, then we need a distance metric to do so. A logical choice would be the total variation metric used to measure the distance between probability distribution.

Definition 2.5. *The total variation norm $\|\cdot\|_{TV}$ for two measures μ_1 and μ_2 on the measurable space $(\mathcal{X}, \mathcal{B}(X))$ is defined in the following way:*

$$\|\mu_1 - \mu_2\|_{TV} = \sup_{A\in\mathcal{B}(\mathcal{X})} \|\mu_1(A) - \mu_2(A)\|. \tag{2.8}$$

Thus, if a Markov chain (λ, P) converges, then we can approximate the probability distribution for the states of the chain at time n, $\pi^{n,\lambda}$, by the limiting distribution π for n sufficiently large. Now, we still need to determine when a Markov chain converges, and if it does to what distribution. We start by considering a necessary condition for a limiting distribution π.

Definition 2.6. *A probability distribution π on $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ is called invariant with respect to a Markov transition kernel P if*

$$\pi = \pi\mathcal{P} = \int_{\mathcal{X}} P(y, \cdot)\, \Pi(dy). \tag{2.9}$$

We will now define detailed balance, which turns out to be a sufficient condition for π being invariant with respect to the Markov transition kernel P.

Definition 2.7. *We say that a distribution π and a Markov transition kernel P are in detailed balance if for all $A, B \in \mathcal{B}(\mathcal{X})$ and $\Pi(A), \Pi(B) > 0$, the following holds:*

$$\int_A P(x, B)\, \Pi(dx) = \int_B P(y, A)\, \Pi(dy). \tag{2.10}$$

Note that as $\mathcal{X}$ is a discrete state space, detailed balanced can also be defined as

$$\Pi(dx)P(x, dy) = \Pi(dy)P(y, dx) \text{ for all } x, y \in \mathcal{X}.$$

Although, a less general definition for detailed balance, this is the definition that we will use later when we introduce the Metropolis-Hasting Markov Chain Monte Carlo sampling scheme. Using the detailed balance, we can now obtain a sufficiency condition for the distribution π being invariant with respect to the Markov transition kernel P.

Proposition 2.1. *If the Markov transition kernel P and the distribution π on the state space $\mathcal{X}$ are in detailed balance, then π is an invariant distribution for $\mathcal{P}$.*

Proof. Start by writing, with $B \in \mathcal{B}(\mathcal{X})$,

$$\pi\mathcal{P}(B) = \int_{\mathcal{X}} P(x, B)\, \Pi(dx) = \int_B \underbrace{P(y, \mathcal{X})}_{=1}\, \Pi(dy) = \Pi(B).$$

From which the result immediately follows. Detailed balance was used to interchange B and $\mathcal{X}$ in the integrals and the last equality followed from the definition of the Markov transition kernel P, see definition 2.4. □

Now, we have seen candidate distribution to which the Markov chain (λ, P) could converge. However, we need more conditions to establish converge than just a distribution π that is invariant with respect to the Markov transition kernel P. We will define some extra notions that are useful in establishing convergence [GWW86].

Definition 2.8. *(Irreducibility). Let $\mathcal{X}$ be a discrete state space, the Markov chain (λ, P) is irreducible if all states communicate, which means that*

$$P(\tau_y < \infty \mid X_0 = x) > 0, \qquad \forall x, y \in \mathcal{X},$$

where τ_y being the first time the state y is visited by the chain, see definition 2.3.

Definition 2.9. *(Aperiodicity). An irreducible Markov chain (λ, P) is aperiodic if for all $x \in \mathcal{X}$,*

$$gcd\{n \in \mathbb{N}_0 : \pi^{n,\delta_x} > 0\} = 1.$$

Definition 2.10. *(Positive recurrence). For a Markov chain with transition kernel P on a discrete state space $\mathcal{X}$. Define $\tau_{ii} = inf\{n \in \mathbb{N} : X_n = y_i \mid X_0 = y_i\}$, the stopping time for the chain revisiting its starting state. If for all $y_i \in \mathcal{X}$ it hold that*

$$\mathbb{E}[\tau_{ii}] < \infty,$$

then we say that the Markov chain is positive recurrent.

The following result gives sufficient conditions for the convergence of a Markov chain, in the total variation norm, on a discrete state space.

Theorem 2.1. *([RC04]). For a positive recurrent and aperiodic Markov chain with transition kernel P on a countable state space, for every probability measure μ on $\mathcal{X}$, with π the unique invariant distribution of the chain,*

$$\lim_{n\to\infty} \left\| \int_{\mathcal{X}} P^{(n)}(x, \cdot)\mu(dx) - \Pi \right\|_{TV} = 0.$$

The following result will be useful as we consider Markov chain Monte Carlo (MCMC) methods in the next section. The idea behind MCMC is to simulate a Markov chain that converges to the target distribution π: the distribution from which we want to generate samples to obtain Monte Carlo estimates. This will be done by ensuring that the simulated Markov chain is in detailed balance with π, then if we ensure positive recurrence and aperiodicity the Markov chain will eventually converge to the target distribution.

2.3 Markov Chain Monte Carlo

Here we introduce the main idea behind Markov Chain Monte Carlo sampling schemes on discrete spaces by considering the Metropolis-Hastings algorithm.

2.3.1 Metropolis-Hastings

The idea behind MCMC is to simulate a Markov chain that has π (the target distribution) as its invariant distribution. One of the most popular algorithms, in a discrete time and state space setting, used to construct and simulate such a Markov chain is the Metropolis-Hastings (MH) algorithm. The idea behind this algorithm is to simulate a proposal Markov chain, where this proposed next state is accepted or rejected with a certain probability. By choosing this acceptance probability in a clever way we can force the simulated Markov chain to be in detailed balance with π [Nob21].

Start by choosing $Q(x, dy)$ a Markov transition kernel for which it holds that $Q(x, dy) = 0 \iff Q(y, dx) = 0$. A natural choose for Q would be the symmetric random walk kernel, where the Markov chain has a uniform probability to walk to a next state defined on a neighbourhood of the current state x.

Next, we define the acceptance probability for any $x, y \in \mathcal{X}$ by

$$\alpha(x,y) = \min\left\{1, \frac{\Pi(y)Q(y,dx)}{\Pi(x)Q(x,dy)}\right\} \qquad \text{if } Q(x,dy) > 0$$
$$\alpha(x,y) = 0 \qquad \text{if } Q(x,dy) = 0.$$

Sampling from this constructed Markov chain can then be done by taking the steps described in Algorithm 1.

Algorithm 1 Metropolis-Hastings algorithm

Given: λ (initial distribution), Q (proposal), π (target)
Generate $X_0 \sim \lambda$
  for $n = 1, \ldots, N$ **do**
    Propose a new state $\hat{X}_{n+1} \sim Q(X_n, \cdot)$
    Generate $U \sim \mathcal{U}([0,1])$
    if $\alpha(X_n, \hat{X}_{n+1}) \geq U$ **then**
      $X_{n+1} = \hat{X}_{n+1}$ proposal accepted
    else
      $X_{n+1} = X_n$ proposal rejected
    end if
  end for

Lemma 2.1. *The transition kernel of the chain produced by the Metropolis-Hastings algorithm is given by [Zan17]*

$$P(x, dy) = \alpha(x,y)Q(x,dy) + \delta_x(dy)(1 - \int_\chi \alpha(x,z)Q(x,dz)). \tag{2.11}$$

Proof. Let $(\mathcal{X}, \mathcal{B}(\mathcal{X}))$ be a measurable space. Suppose that $X_n = x$, and sample $U \sim \text{Unif}([0,1])$ and $\tilde{X}_{n+1} \sim Q(x, \cdot)$. Then,

$$\begin{aligned} P(x,B) &= P(X_{n+1} \in B \mid X_n = x) \\ &= P(\tilde{X}_{n+1} \in B, U \leq \alpha(X_n, \tilde{X}_{n+1}) \mid X_n = x) + P(X_n \in B, U > \alpha(X_n, \tilde{X}_{n+1}) \mid X_n = x). \end{aligned}$$

We split the previous expression into two parts, and start by considering the first one,

$$\begin{aligned} P(\tilde{X}_{n+1} \in B, U \leq \alpha(X_n, \tilde{X}_{n+1}) \mid X_n = x) &= \int_B \int_0^1 \mathbb{1}_{\{u \leq \alpha(x,y)\}}\, du\, Q(x,dy) \\ &= \int_B \alpha(x,y)\, Q(x,dy). \end{aligned}$$

For the second part of the expression we obtain the following:

$$\begin{aligned} P(X_n \in B, U > \alpha(X_n, \tilde{X}_{n+1}) \mid X_n = x) &= \mathbb{1}_B(x) \int_{\mathcal{X}} \int_0^1 \mathbb{1}_{\{u > \alpha(x,y)\}}\, du\, Q(x,dy) \\ &= \mathbb{1}_B(x) \int_{\mathcal{X}} (1 - \alpha(x,y))\, Q(x,dy). \end{aligned}$$

If we combine these two parts, and set $B = \{y\}$ then we obtain the desired result. □

Recall from the previous section, proposition 2.1, that when P and π are in detailed balance, the Markov chain with transition kernel P has π as its invariant distribution. The following theorem confirms that this is in fact the case.

Theorem 2.2. *Let P be the transition kernel from the Markov process that is produced by the Metropolis-Hastings algorithm, then P and π are in detailed balance.*

Proof. We need to show that $\Pi(dx)P(x,dy) = \Pi(dy)P(y,dx)$ for all $x, y \in \chi$. It is trivial that this holds for $x = y$. Hence, suppose that $x \neq y$. If we then have that $\Pi(x)P(x,dy) = 0$, it follows that $P(x,dy) = 0$ as we assumed that the Markov chain is irreducible. This means that $Q(x,dy) = 0 \Rightarrow Q(y,dx) = 0$. Thus, we can conclude that $P(y,dx) = 0 \Rightarrow \Pi(dx)P(x,dy) = \Pi(dy)P(y,dx)$. In case $\Pi(dx)P(x,dy) \neq 0$ we obtain the following by using Lemma 2.1

$$\begin{aligned}\Pi(dx)P(x,dy) &= \Pi(dx)\alpha(x,y)Q(x,dy) = \Pi(dx)Q(x,dy)\min\left\{1, \frac{\Pi(dy)Q(y,dx)}{\Pi(dx)Q(x,dy)}\right\} \\ &= \min\{\Pi(dx)Q(x,dy), \Pi(dy)Q(y,dx)\} = \Pi(dy)Q(y,dx)\left\{\frac{\Pi(dx)Q(x,dy)}{\Pi(dy)Q(y,dx)}, 1\right\} \\ &= \Pi(dy)P(y,dx).\end{aligned}$$

Therefore, we have shown that P and π are in detailed balance. □

Recall, however, from subsection 2.2.2 that we need more than just detailed balance to obtain the required convergence. An extra condition that we need for the convergence of the Metropolis-Hastings Markov chain is *pi*-irreducible with respect to the target distribution.

Definition 2.11. *Given a probability distribution π, the Markov chain (X_n) with Markov transition kernel $P(x,dy)$ is π-irreducible if, for every $A \in \mathcal{B}(\mathcal{X})$ with $\Pi(A) > 0$, there exists $n \in \mathbb{N}$ such that $P^(n)(x,A) > 0$ for all $x \in \mathcal{X}$.*

Note that π-irreducibility is in fact equivalent to stating that $P_(\tau_A \mid X_0 = x) > 0$ for all $x \in \mathcal{X}$ and all $A \in \mathbb{B}(\mathcal{X})$. Defined in this way, π-irreducibility is equivalent to irreducibility, see definition 2.8, on the discrete state space $\mathcal{X}$.

Theorem 2.3. *([RC04]). Suppose that the Metropolis-Hastings Markov chain (X_n) with transition kernel K is π-irreducible.*

(i) If $h \in L^1(\pi)$, then

$$\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} h(X_i) = \int_{\mathcal{X}} h(x)\pi(x)\,dx \quad a.e.\ \pi.$$

(ii) If, in addition, (X_n) is aperiodic, then

$$\lim_{n\to\infty} \left\| \int_{\mathcal{X}} K^{(n)}(x,\cdot)\mu(dx) - \Pi \right\|_{TV}$$

for every probability measure μ on $\mathcal{X}$, where $K^{(n)}(x,\cdot)$ denotes the kernel for n transitions.

Hence, for Metropolis-Hastings to work properly as an MCMC scheme, we need to check irreducibility and aperiodicity for the simulated Markov chain (X_n) with transition kernel K. If we choose for the proposal kernel Q the symmetric random walk kernel, i.e. a uniform distribution on all neighbouring states, then it is not hard to see that these conditions are met.

Metropolis-Hastings efficiency

The efficiency of the Metropolis-Hastings sampling algorithm depends on the speed of convergence of the chain and the computational resources needed to generate samples. One of the major factors in determining the efficiency of the MCMC sampling scheme is the speed with which the simulated Markov chain converges. A chain that takes a long time to converge has a long burn-in time: the number of simulated Markov chain iterations that need to be deleted as the chain is still far from convergence and has not reached a stationary state. Note that when we speak about convergence,

we mean that the chain has reached stationarity: when the traceplots show regular or repeated patterns. This is inspected visually by considering a trace plot.

A proposal that is often rejected is also inefficient as it takes it increases the time needed to reach stationarity, since the chain needs to move in order to achieve this. Furthermore, as our goal is to generate independent samples from π, a state that is the result of a rejection is correlated with the previous state and is therefore not an independent realization of π.

Suppose now that we choose $Q(x, dy)$ to be an uniformed symmetric kernel. This means that Q will propose a random walk with a uniform distribution over the neighbouring states of x. Note that in this case the acceptance probability becomes

$$\alpha(x, y) = \min\left\{1, \frac{\pi(y)}{\pi(x)}\right\}.$$

Which means that the proposed state is always accepted whenever $\pi(y) \geq \pi(x)$. However, the proposal kernel does not take this into account when proposing a new state. The ratio $\frac{\pi(y)}{\pi(x)}$ can usually be calculated reasonably quickly. Indeed, if we know the target distribution up to a constant $\pi = c\tilde{\pi}$, where we only know $\tilde{\pi}$, then $\frac{\tilde{\pi(y)}}{\pi(x)} = \frac{\pi(y)}{\pi(x)}$. In the next chapter we will see that we can incorporate this knowledge into the proposal kernel to increase the efficiency of the Metropolis-Hastings algorithm. The rationale behind this is that the acceptation-rejection step corrects for the fact that the proposed states are not distributed according to π, where sampling new states proportional to the probability ratio $\pi(y)/\pi(x)$ leaves less correcting to do and results in fewer rejected proposal states.

2.3.2 Gibbs sampler

Suppose that the state space $\mathcal{X}$ has several components, $x = (x_1, \ldots, x_d)$, with $x_i \in \mathcal{X}_i$. The Gibbs sampler is a single component MH, where we update each component separately. The advantage of using this approach is that it converts a d-dimensional problem into d separate 1-dimensional problems [Pen22]. In order to do this we do need a proposal density $q(y_i \mid x_{-i})$ for each conditional density $\pi(x_i \mid x_{-i})$, with $x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d)$, as we cycle trough sampling from $\pi(x_i \mid x_{-i})$ for each component i.

Algorithm 2 Gibbs sampler

Given: λ (initial distribution), π (target distribution)
Generate $X_0 \sim \lambda$
for $n = 1, \ldots, N$ **do**
  Set $Y_{n+1} = X_n$
  for i = 1, …, d **do**
    Sample $\hat{Y}_{n+1,i} \sim q(Y_{n+1,i} \mid Y_{n+1,-i})$
    Generate $U \sim \mathcal{U}([0,1])$
    if $\alpha(\hat{Y}_i \mid Y) \geq U$ **then**
      Set $Y_{n+1,i} = \hat{Y}_{n+1,i}$ proposal accepted
    else
      $Y_{n+1,i} = Y_{n+1,i}$ proposal rejected
    end if
  end for
  Set $X_{n+1} = Y_{n+1}$
end for

The procedure for applying a Gibbs sampling scheme is given in algorithm 2. The accept-reject probability is defined by the following:

$$\alpha(\hat{y}_i \mid y) = \min\left\{\frac{\pi(\hat{y}_i \mid y_{-i})\pi(y_i \mid y)}{\pi(y_i \mid y_{-i})\pi(\hat{y}_i \mid y)}, 1\right\}.$$

2.4 Diagnostics

The simulated Markov chain will, if converged, generate independent and identical realisations of the target distribution. However, this only occurs asymptotically, which means that this will

happen if we were to generate the Markov chain for an infinite amount of time. When we want to make concrete estimations, this is indeed not possible and we will need to make estimations with only a finite number of samples. In this case, diagnostics can be used to assess the quality of the finite chain against the properties of the converged chain.

2.4.1 Effective sample size

Effective sample size is a diagnostic used to assess the quality of the obtained samples from the simulated Markov chain. This is done by considering the autocorrelation within the simulated Markov chain (X_n). Ideally, we want to sample identical and independent realizations from the target π, where the simulated Markov chain is by design created to converge to the target distribution. As a proxy to measure this convergence, we are correcting for the autocorrelation within the chain to obtain an estimate of the number of independent samples. Indeed, independent samples hint at the Markov chain state distribution being relatively unchanged for each time iteration, i.e. the chain is close to converging to the target.

Assume that the state space $\mathcal{X}$ is 1-dimensional. If this is not the case, then a summary statistic of the simulated chain should be used. The following formula is used to calculate effective sample size [Rip87]:

$$ESS = \frac{n}{1 + 2\sum_{k=1}^{\infty} \rho(X_k)}, \tag{2.12}$$

where n is the number of samples, and $\rho(X_k)$ is the autocorrelation at lag k.

2.4.2 Gelman-Rubin diagnostic

The Gelman-Rubin diagnostic can be used to assess how well the used the used samplers converge, not how close a certain samples are to being independent realizations of the target distribution. The method will be used as presented in [GR92]. The main idea is to generate multiple sequences from with different starting states, the chains are then studied by consideration of a scalar summary statistic. In the beginning, when the chains have not yet converged, the individual sequences will be quite different as they are still dependent on their starting state. When converged, all the chains should be identical, and so the sequences of the summary statistic should be indistinguishable. As a proxy to measure this, the variability within a sequence is compared to the total variability, where indeed in the beginning the sequence variability will be smaller as the entire state space is yet to be fully explored.

The Gelman-Rubin diagnostic is especially relevant when it comes to applied inference for Bayesian posterior distributions as the diagnostic is derived as a normal-theory approximation to exact Bayesian inference, conditional on the observed simulations.

The method consists of two major steps. First, the starting states for the simulated sequences are generated from an overdispersed distribution, i.e. a distribution that is more variable than the target distribution. Second, the generated starting states are used to run the iterative simulations. Each iterative sequence is then analyzed to obtain a distributional estimate of what is known about the target distribution, given the realised simulations thus far. In the end, this results in an estimate on how close the simulation process is to convergence. This is eloquently summarized in the resulting statistic R: the factor by which the scale of the current distribution for the summary statistic might be reduced if the simulations were to be continued asymptotically.

The overdispersed starting distribution

In the case of a discrete state space, the overdispersed starting distribution is empirically constructed by generating a totally random state, which is used as the starting state for the Metropolis-Hasting algorithm with a symmetric random walk kernel. After a relatively small number of steps, the final state of this simulation is added to the empirical overdispersed distribution. The rationale behind this procedure is that the totally random state is indeed overdispersed, where a small amount of information about the target is incorporated by considering a fixed number of proposed states. Repeat this procedure N times (typically $N = 1000$) and consider these samples taken from the overdispersed distribution.

A way to improve the realizations from this overdispersed distribution is by using a procedure called *importance resampling*. Draw without replacement m samples from the previously realized

N, where the probability of drawing a state x is proportional to the log energy of the target density. A typical value for m would be somewhere in the ballpark of $m = 10$, since these realizations will be the starting point for each of the simulated sequences.

Analyzing the iterative simulations from multiple sequences

For each of the m sampled starting states simulate a sequence with length $2n$, where we will consider the first n as burn-in and focus on the last n iterations. Denote by x the considered summary statistic and by π the target distribution.

For the summary statistic, calculate the following,

$$B = \text{the variance between the m sequence means } \overline{x}_i \text{ for n values of } x;$$

$$B = \frac{1}{n}\sum_{i=1}^{m}\frac{(\overline{x}_i - \overline{x})^2}{(m-1)}; \tag{2.13}$$

$$W = \text{the average of the } m \text{ within sequence variances } s_i^2;$$

$$W = \sum_{i=1}^{m}\frac{s_i^2}{m}. \tag{2.14}$$

With $\overline{x}$ being the average of the averages of the m simulated sequences. Use $\overline{x}$ to estimate the target mean, $\mu = \int x\pi(x)\,dx$. The target variance, $\sigma^2 = \int(x-\mu)^2\pi(x)dx$, will be estimated by a weighted average of W and B,

$$\hat{\sigma}^2 = \frac{n-1}{n}W + \frac{1}{n}B. \tag{2.15}$$

Under the assumption that the starting distribution is overdispersed, $\hat{\sigma}^2$ is an overestimation of σ^2. However, if the distribution from which the starting states were sampled would be the target distribution, then $\hat{\sigma}^2$ is in fact an unbiased estimator of σ^2.

Now, we are ready to estimate what is known about x. This will be done by using an approximate Student's t distribution, since we will also account for the sampling variability of the estimates $\hat{\mu}$ and $\hat{\sigma}^2$. The resulting distribution will have the following properties:

$$\text{center: } \hat{\mu};$$

$$\text{scale: } \sqrt{\hat{V}} = \sqrt{\hat{\sigma}^2 + \frac{B}{mn}};$$

$$\text{degrees of freedom: } df = 2\frac{\hat{V}^2}{\hat{var}(\hat{V})}.$$

The $\hat{var}(\hat{V})$, is defined by the following expression,

$$\hat{var}(\hat{V}) = \left(\frac{n-1}{n}\right)^2\frac{1}{m}\hat{var}(s_i^2) + \left(\frac{m+1}{mn}\right)^2\frac{2}{m-1}B^2 + 2\frac{(m+1)(n-1)}{mn^2}\cdot\frac{n}{m}\left[\hat{cov}(s_i^2,\overline{x}_i^2) - 2\overline{x}\;\hat{cov}(s_i^2,\overline{x}_i)\right]. \tag{2.16}$$

Notice that all estimated variances and covariances are applied to the m sample values of $\overline{x}_i$ and s_i^2.

Lastly, we can monitor the convergence of the iterative simulations by estimating the factor by which the scale of the current distribution could be reduced if the simulations were to be continued indefinitely. This is estimated by the quantity $\hat{R}$, given by the following expression

$$\sqrt{\hat{R}} = \sqrt{\left(\frac{\hat{V}}{W}\right)\frac{df}{df-2}}. \tag{2.17}$$

The factor $\hat{R}$ declines to 1 as $n \to \infty$. This factor is composed of the ratio of the current variance estimate, $\hat{R}$, to the within-sequence variance, W, with the factor $df/(df-2)$ accounting for the extra variance of the t distribution when compared to the Normal distribution. Therefore, if this factor is high, typically that means $\hat{R} > 1.1$, then a lot of scale reduction can still be achieved by continuing the iterative simulations; the samplers have not yet converged.

Chapter 3

Informed proposals for local MCMC

In this chapter we will be concerned exclusively with sampling from a target probability distribution Π on a discrete state space $\mathcal{X}$. We will also assume that Π admits a bounded density π with respect to the counting measure dx. Furthermore, when we speak about the proposal kernel of the Metropolis-Hastings algorithm, then we will denote this Markov kernel by Q. On the other hand, if we are considering the resulting Markov kernel of the process, with incorporation of the acceptation-rejection step, we will denote this by P.

The corresponding theory, and the Peskun optimality of pointwise informed proposals that are locally balanced with respect to the target measure Π, as introduced in [Zan17] will be explored.

3.1 Locally-balanced proposals

Let $\mathcal{X}$ be a discrete state space equipped with a distance metric d, consider $K_\sigma(x, dy)$ the uniform distribution on the states:

$$y \in \mathcal{X} : d(x, y) \leq \sigma.$$

For $\sigma \downarrow 0$ $K_\sigma(x, dy)$ converges weakly to the delta measure in x, whereas for $\sigma \uparrow \infty$ it converges weakly to the base measure dy, i.e. the uniform distribution on all states in $\mathcal{X}$. The proposal kernels that we will consider combine the kernel K_σ with a multiplicative bias $g(\pi(y)/\pi(x))$. Therefore, we will consider proposals with the following structure, as introduced by [Zan17],

$$Q_{g,\sigma}(x, dy) = \frac{g\left(\frac{\pi(y)}{\pi(x)}\right) K_\sigma(x, dy)}{Z_g(x)}. \tag{3.1}$$

Where $g : (0, \infty) \to (0, \infty)$ is a continuous function, and where $Z_g(x)$ is the normalizing constant:

$$Z_g(x) = \int_{\mathcal{X}} g\left(\frac{\pi(z)}{\pi(x)}\right) K_\sigma(x, dz). \tag{3.2}$$

By the continuity of g and the fact that π is a probability density, it is not hard to see that this integral is finite. Furthermore, as was done in [Zan17], we will assume that $g(t) \leq at + b$ for some $a, b \in \mathbb{R}$. Now, we can introduce the notion of locally-balanced kernels.

Definition 3.1 (*Locally balanced kernels* [Zan17])**.** *A family of Markov transition kernels* $\{Q_{g,\sigma}\}_{\sigma>0}$ *is locally-balanced with respect to a distribution* Π *if each* $Q_{g,\sigma}$ *is reversible with respect to some distribution* Π_σ *such that* Π_σ *converges weakly to* Π *as* $\sigma \downarrow 0$*.*

The motivation behind using locally-balanced proposals is that these will be approximately Π-reversible for local moves: moves proposed from a kernel $Q_{g,\sigma}$ with σ small. Therefore, the acceptation-rejection step will need to do less correcting and reject less moves. As explained earlier, this increases the efficiency of the MH-algorithm. The reason for choosing small moves is that as the dimensionality of $\mathcal{X}$ increases, the MH moves decrease relative to the size of the state

space $\mathcal{X}$. This is especially relevant since for many distributions defined on $\mathcal{X}$, the reason for using MCMC is the computational cost needed to calculate the normalizing constant for the distribution Π. The following theorem characterizes the pointwise informed proposals that are locally-balanced.

Theorem 3.1 (*[Zan17]*)**.** *A pointwise informed proposal* $\{Q_{g,\sigma}\}_{\sigma>0}$ *is locally-balanced with respect to a general* Π *if and only if*

$$g(t) = tg(1/t) \qquad \forall t > 0. \tag{3.3}$$

From now on we will refer to these functions g as *balancing functions*. Later we will see that balancing functions are Peskun optimal when used in the pointwise informed proposals in Equation 3.1. However, note that there are not yet theoretical results on what function is optimal within the class of balancing functions.

3.2 Peskun ordering

Usually when we perform Monte Carlo simulations we want to estimate the quantity

$$I = E_\pi[f(X)]. \tag{3.4}$$

Under the assumption that $E_\pi[|f|] < \infty$. For this we will use the estimator

$$\hat{I} = \frac{1}{N}\sum_{t=1}^{N} f\{X(t)\}, \tag{3.5}$$

where $X(t)$ are realizations of the simulated Markov chain in the MCMC simulations. The error made in estimation is approximately equal to the variance of the estimator; $\text{Error}(\hat{I}) \approx \text{var}(\hat{I})$. The expression for the asymptotic variance of $\hat{I}$, where we assume that the chain has reached stationarity, as stated by [KS$^+$60] is the following:

$$\text{var}(f, \pi, P) = \lim_{N\to\infty} N\text{var}\left[\sum_{t=1}^{N} f\{X(t)\}/N\right]. \tag{3.6}$$

Note that P refers to the resulting Markov transition kernel of the MH-algorithm, and with stationarity it is meant that we assume that $X_1 \sim \pi$. Therefore, reduction of the asymptotic variance would asymptotically result in better estimation of the target quantity I.

Now, Peskun ordering is a way to order MH-algorithms, in terms of their respective asymptotic variance, by studying their resulting Markov kernels. Let P_1 and P_2 be two be two such resulting Markov kernels, then we say that $P_1 \geq P_2$ if

$$P_1(x, dy) \geq P_2(x, dy) \text{ for all } x, y \in \chi \text{ such that } x \neq y. \tag{3.7}$$

This means that P_1 has a uniformly higher probability of moving to a different state; more probability mass on the off-diagonal elements. Equivalently, this could be reformulated as

$$P_{P_1}(X_{n+1} \neq x \mid X_n = x) \geq P_{P_2}(X_{n+1} \neq x \mid X_n = x) \text{ for all } x \in \chi. \tag{3.8}$$

The significance of this ordering follows from the following theorem, which was stated first in [Pes73].

Theorem 3.2. *Suppose that both of the irreducible transition matrices* P_1 *and* P_2 *are in detailed balance with the same probability distribution* π*. If* $P_1 \geq P_2$*, then for the estimate* $\hat{I} = \sum^N)_{t=1} f\{X(t)\}/N$*,*

$$var(f, \pi, P_1) \leq var(f, \pi, P_2).$$

Thus, being optimal in terms of Peskun ordering is equivalent to having the lowest asymptotic variance of the resulting estimator $\hat{I}$.

3.3 Optimality of balancing functions

In this section we will take a fixed σ and only study the optimal class of g, the balancing functions, in terms of the Peskun ordering. Therefore, we will also index the pointwise informed proposals as stated in Equation 3.1 only by g and write Q_g. Before we can state the main result we need to define the following constant,

$$c_g = \sup_{(x,y)\in R} \frac{Z_g(y)}{Z_g(x)}, \tag{3.9}$$

where $R = \{(x, y) \in \mathcal{X} \times \mathcal{X} : \pi(x)K_\sigma(x, dy) > 0\}$ and $Z_g(x)$ as defined in Equation 3.2.

Theorem 3.3. *Consider the state space $\mathcal{X}^{(n)}$ with target distribution $\pi^{(n)}$, where n refers to the dimension of the state space. Let $g : (0, \infty) \to (0, \infty)$ be continuous and locally-bounded. Define $\tilde{g}(t) = \min\{g(t), tg(1/t)\}$. If we then apply the Metropolis-Hastings algorithm with the pointwise informed proposals Q_g and $Q_{\tilde{g}}$, we get the resulting kernels P_g and $P_{\tilde{g}}$. If it holds that*

$$c_g^{(n)} \to 1 \qquad as \quad n \to \infty, \tag{3.10}$$

for every g that is continuous and bounded on compact subsets of $(0, \infty)$. Then we obtain that

$$P_{\tilde{g}}(x, y) \geq P_g(x, y) \quad \forall x \neq y. \tag{3.11}$$

Where Equation 3.11 refers to the Peskun optimality, since this causes the asymptotic variance of the estimate to be smaller, see Theorem 3.2. Note that Theorem 3.3 states that locally-balanced proposals are asymptotically optimal in terms of Peskun ordering, since any pointwise informed proposal as stated in Equation 3.1 can be characterized by the choice of g. By construction $\tilde{g}$ is a balancing function, which means that the proposal kernel becomes locally-balanced as is stated in Theorem 3.1. Hence, this implies Peskun optimality since for every pointwise informed proposal we can construct a locally-balanced one that has less asymptotic variance.

Now, we will focus on proving this result.

Proof. Suppose that $g : (0, \infty) \to (o, \infty)$ is continuous and locally bounded, and let $x, y \in \mathcal{X}^{(n)}$ such that $x \neq y$, with n the dimension of the state space. Remember that we have chosen σ fixed in this section. Therefore, we have only indexed the kernel with the function g. In order to simplify notation, define $t_{xy} = \pi(y)/\pi(x)$ In this case, the resulting Markov kernel from the MH-algorithm, for $x \neq y$, is the following,

$$P_g(x, dy) = Q_g(x, dy)\alpha(x, y) = \frac{g(t_{xy})K(x, dy)}{Z_g(x)} \min\left\{1, t_{xy}\frac{g(t_{yx})}{Z_g(y)}\frac{Z_g(x)}{g(t_{xy})}\right\}, \tag{3.12}$$

where we have taken the proposal kernel expression from Equation 3.1. Then, we can use this expression to derive a different expression for the proposal kernel divided by the symmetric random walk kernel,

$$\frac{P_g(x, dy)}{K(x, dy)} = \frac{g(t_{xy})}{Z_g(x)} \min\left\{1, t_{xy}\frac{g(t_{yx})}{Z_g(y)}\frac{Z_g(x)}{g(t_{xy})}\right\} = \min\left\{\frac{g(t_{xy})}{Z_g(x)}, t_{xy}\frac{g(t_{yx})}{Z_g(y)}\right\}$$

Using Equation 3.12 we can derive the following inequality:

$$\frac{P_g(x, dy)}{K(x, dy)} = \frac{Z_g(x)}{Z_g(y)}\frac{Z_g(y)}{Z_g(x)} \min\left\{\frac{g(t_{xy})}{Z_g(x)}, t_{xy}\frac{g(t_{yx})}{Z_g(y)}\right\} \geq \frac{1}{c_g^{(n)}} \frac{\min\{c_g^{(n)} g(t_{xy}), t_{xy}g(t_{yx})\}}{Z_g(x)} \tag{3.13}$$

$$\geq \frac{1}{c_g^{(n)}} \frac{\min\{g(t_{xy}), t_{xy}g(t_{yx})\}}{Z_g(x)}. \tag{3.14}$$

Similarly, we can also derive an upper bound for the same expression:

$$\frac{P_g(x, dy)}{K(x, dy)} = \frac{Z_g(x)}{Z_g(y)}\frac{Z_g(y)}{Z_g(x)} \min\left\{\frac{g(t_{xy})}{Z_g(x)}, t_{xy}\frac{g(t_{yx})}{Z_g(y)}\right\} \leq c_g^{(n)} \frac{\min\left\{\frac{1}{c_g^{(n)}} g(t_{xy}), t_{xy}g(t_{yx})\right\}}{Z_g(x)} \tag{3.15}$$

$$\leq c_g^{(n)} \frac{\min\{g(t_{xy}), t_{xy}g(t_{yx})\}}{Z_g(x)}. \tag{3.16}$$

For both inequalities we have used the definition of $c_g^{(n)}$, see Equation 3.9, and the fact that $c_g^{(n)} \geq 1$ since the expression is symmetric in x and y. Furthermore, it follows that $\tilde{g}(t) \leq g(t)$ by construction of $\tilde{g}$. This also implies that $Z_g(x) \geq Z_{\tilde{g}}(x)$. Now, we start by using Equation 3.13 to deduce the following,

$$P_{\tilde{g}}(x, dy) \geq \frac{1}{c_{\tilde{g}}^{(n)}} \frac{\min\{\tilde{g}(t_{xy}), t_{xy}\tilde{g}(t_{yx})\}}{Z_{\tilde{g}}(x)} K(x, dy) = \frac{1}{c_{\tilde{g}}^{(n)}} \frac{\tilde{g}(t_{xy})}{Z_{\tilde{g}(x)}} K(x, dy). \tag{3.17}$$

In similar fashion, this time by utilizing the upper bound in Equation 3.15, the following can be derived,

$$P_g(x, dy) \leq c_g^{(n)} \frac{\tilde{g}(t_{xy})}{Z_g(x)} K(x, dy) \leq \frac{1}{c_{\tilde{g}}^{(n)}} \frac{\tilde{g}(t_{xy})}{Z_{\tilde{g}(x)}} K(x, dy). \tag{3.18}$$

Thus, if we combine Equation 3.17 and Equation 3.18, then we obtain that

$$P_{\tilde{g}}(x, dy) \geq \frac{1}{c_g^{(n)} c_{\tilde{g}}^{(n)}} P_g(x, dy). \tag{3.19}$$

If we now use the condition stated in Equation 3.10, then as $n \to \infty$ we get that $c_g^{(n)} \to 1$ and $c_{\tilde{g}}^{(n)} \to 1$. It is important to note that if g is continuous and bounded on compact subsets of $(0, \infty)$, then, by construction, this trivially also holds for $\tilde{g}$. Hence, we end up with the desired result that

$$P_{\tilde{g}}(x, dy) \geq P_g(x, dy) \text{ for } n \to \infty \text{ and } x \neq y.$$

□

Chapter 4

Accelerated sampling on discrete spaces

In the previous chapter we have seen that by using locally-balanced proposals embedded in the Metropolis-Hastings framework, sampling on discrete spaces can be dramatically improved. It turns out that further improvement can be found when sampling with a continuous-time Markov process on a discrete state space: a Markov Jump Process (MJP). We will study the approach as described in [PG19], where the jumping rates are calculated by using locally-balanced functions (Theorem 3.1). Furthermore, by incorporating the algebraic structure of the state space into the sampling algorithm, the exploration of new modes is encouraged.

4.1 Markov Jump Process

The approach is to sample on a discrete state space with a continuous-time Markov process. There is only one class of Markov processes with these characteristics: Markov Jump Processes.

Definition 4.1. *[PG19] A Markov Jump Process is a continuous-time Markov process on a countable state space $\mathcal{X}$. Such a process is entirely characterised by its jump rates, denoted by $\lambda(x \to y)$, defined in the following way*

$$P(X_{t+h} = y \mid X_t = x) = h \cdot \lambda(x \to y) + o(h) \quad \text{for } x \neq y \tag{4.1}$$

$$P(X_{t+h} = x \mid X_t = x) = 1 - h \cdot \Lambda(x) + o(h). \tag{4.2}$$

Where $\Lambda(x) = \sum_y \lambda(x \to y)$. The generator of a MJP is given by

$$\mathcal{L}f(x) = \sum_y \lambda(x \to y)[f(y) - f(x)]. \tag{4.3}$$

Similar to how we did this for a discrete-time Markov process, we need a locally verifiable condition to ensure that the simulated Markov process has the target measure π as its invariant measure. We will use the reversibility condition with respect to the MJP generator as this is a sufficient condition for the MJP to have π as its invariant measure.

Definition 4.2. *[PG19] A Markov Jump Process with generator $\mathcal{L}$ is reversible with respect to the measure π if, for all $f, g \in L^2(\pi)$, it holds that*

$$E_\pi[(\mathcal{L}f)(x)g(x)] = E_\pi[f(x)(\mathcal{L}g)(x)]. \tag{4.4}$$

The following calculations show that this reduces to detailed balance in some cases.

Proposition 4.1. *By considering $f(x) = I[x = a], g(x) = I[x = b]$, the reversibility condition is equivalent to detailed balance:*

$$\pi(x)\lambda(x \to y) = \pi(y)\lambda(y \to x) \quad \text{for all } x, y \in \mathcal{X}. \tag{4.5}$$

Proof. Start by noting that

$$\mathcal{L}(I[x = a])(x) \;=\; \sum_y \lambda(x \to y)\{I[y = a] - I[x = a]\}.$$

Hence, for the expectation this means the following

$$\begin{aligned} E_\pi[(\mathcal{L}f)(x)g(x)] \;&=\; \sum_x \sum_y \lambda(x \to y)\{I[y = a] - I[x = a]\}I[x = b]\pi(x) \\ &=\; \sum_y \lambda(b \to y)I[y = a]\pi(b) \\ &=\; \lambda(b \to a)\pi(b). \end{aligned}$$

Where the second equality follows from the fact that only $x = b$ contributes to the summation over x. The other direction follows by a similar calculation. The proof is complete if we consider that we choose $a, b \in \mathcal{X}$ arbitrarily. □

Before we can specify the sampling algorithms, we need to make a few assumptions about the state space $\mathcal{X}$ and the MJP. For the state space $\mathcal{X}$ we assume that it has a graph-like structure. Essentially, this means that the notion of locality refers to the existence of an edge between two elements of $\mathcal{X}$. Thus, it is assumed that there exists some set of edges $\mathcal{E} \in \mathcal{X} \times \mathcal{X}$, where indeed x and y are neighbours if $(x, y) \in \mathcal{E}$. The neighbourhood of x is denoted by $\partial x = \{y \in \mathcal{X} : (x, y) \in \mathcal{E}\}$. The implications of these assumptions on the structure of the state space will be more clear when we consider the algebraic structure of the state space.

On the considered MJPs, $(X_t)_{t\geq 0}$, we make the following assumptions [PG19]:

1. The process can only jump from x to y if $y \in \partial x$.
2. The jump rate from x to y, $\lambda(x \to y)$, is a function of $\frac{\pi(y)}{\pi(x)}$.
3. X_t is in detailed balance with respect π

The first assumption is an indirect consequence of the graph structure assumption of the state space, since this ensures the MJP to behave in accordance with this structure. The second assumption is a simplifying one, however, by applying it we can also see the occurrence of balancing functions as we have already seen in [Zan17]. Thus, by applying the second assumption we obtain $\lambda(x \to y) = g(\frac{\pi(y)}{\pi(x)})$. By writing this into the detailed balance condition we obtain for all $(x, y) \in \mathcal{E}$

$$\begin{aligned} \pi(x)\lambda(x \to y) &= \pi(y)\lambda(y \to x) \\ \pi(x)g\left(\frac{\pi(y)}{\pi(x)}\right) &= \pi(y)g\left(\frac{\pi(x)}{\pi(y)}\right) \\ g\left(\frac{\pi(y)}{\pi(x)}\right) &= \frac{\pi(y)}{\pi(x)}g\left(\frac{\pi(x)}{\pi(y)}\right) \\ g(t) &= t \cdot g(1/t) \quad \text{for } t = \frac{\pi(y)}{\pi(x)}. \end{aligned}$$

Therefore, the jumping rate functions under these assumptions are exactly the same class of functions as described in Theorem 3.1 as derived by [Zan17]. Note that the last assumption is there to ensure the ergodicity of the MJP with respect to the target measure π. We need this for proper convergence of the Monte Carlo estimate. Now, we have derived enough to consider the Zanella process to sample from π.

Algorithm 3 Zanella process

Given: λ (initial distribution), π (target distribution)
Goal: sample from $\pi(x)$ for $x \in \mathcal{X}$
Generate $X_0 \sim \lambda$

for $n = 1, \ldots, N$ **do**

1. For $y \in \partial x$, compute $\lambda(x \to y) = g\left(\frac{\pi(y)}{\pi(x)}\right)$.
2. Compute $\Lambda(x) = \sum_{y \in \partial x} \lambda(x \to y)$.
3. Sample a waiting time $T \sim \text{Exponential}(\text{rate} = \Lambda(x))$.
4. Sample a new location $y \in \partial x$ with probability $\frac{\lambda(x \to y)}{\Lambda(x)}$
5. Advance time by T.
6. Set $X_{n+1} = y$

end for

In Algorithm 3, it can be observed that the Zanella process is essentially a rejection-free random walk algorithm in continuous time, where at each jump the process walks to a neighbour with probability proportional to the jump rate. By also sampling the time at which the jump occurs, the Zanella process becomes rejection free. This changes the way in which the process can be tweaked to increase the efficiency of the Monte Carlo procedure. As the process is rejection free, this is indeed a component that we do not need to worry about. In addition, it is important to note that for the Zanella process the wall-clock time and the clock of the process have essentially been decoupled.

Most of the time that the sampling with the Zanella process takes will be spend on the calculation of the ratios $\frac{\pi(y)}{\pi(x)}$. Therefore, it is important to embed a graph structure in the state space such that a lot of terms in the ratio $\frac{\pi(y)}{\pi(x)}$ cancel. It can also increase efficiency if the sampling algorithm could be tweaked in such a way that travelling along edges into unexplored regions is encouraged. When we consider the algebraic structure of the state space in the next section, we will see how this can be exploited to further enhance the mixing behaviour of the chain. Similar to how the entire class of balancing functions is optimal for pointwise informed proposals [Zan17], there is no strong theory on the balancing function to choose. Later, we will however, consider a heuristic approach on how such a function can be chosen.

4.2 Algebraic structure of the state space

In order to further increase the efficiency of the Zanella process, the goal is to be able to bias the sampling algorithm to travel to regions of the state space that are yet to be explored. Especially, when such regions contain modes of the target distribution. A first step is to consider the algebraic structure of the state space, which means that there is a group G that acts on the state space $\mathcal{X}$. Essentially, this means that by performing the group action to an element $x \in \mathcal{X}$, we obtain a new state $y = g \star x$. We are then able to define the neighbours of x to be the elements y that can be obtained by performing a group action (with an element $g \in G$) to x. Later this idea will be formalized. We start by recalling the definition of a group.

Definition 4.3. *A group is a set G equipped with a binary operation $\cdot : G \times G \to G$, denoted by $(G, \cdot)$. Such that the operation $\cdot$ satisfies the following properties:*

1. *Closure: for all $g, h \in G$ it holds that $g \cdot h \in G$.*
2. *Associativity: for all $g, h, z \in G$ it holds that $(g \cdot h) \cdot z = g \cdot (h \cdot z)$.*
3. *Identity element: there exists an $e \in G$ such that for all $g \in G$, $g \cdot e = e \cdot g = g$.*
4. *Inverse: for each $g \in G$ there exists a unique $g^{-1} \in G$ such that $g \cdot g^{-1} = g \cdot g^{-1} = e$.*

Now we have a group, however, it is not yet clear how we can use the group to induce movement among the states of $\mathcal{X}$. By introducing a group action this can be performed, and the previous idea of generating neighbours for x by using the group can be formalized.

Definition 4.4. *Given a set $\mathcal{X}$ and a group $(G, \cdot)$, a group action is an operation $\star : G \times \mathcal{X} \to \mathcal{X}$ that satisfies the following:*

1. *For all $x \in \mathcal{X}$, $e \star x = x$.*
2. *For all $g, h \in G$, $g \star (h \star x) = (g \cdot h) \star x$.*

By using the group G and the group action we can now define the neighbours of x by $\partial x = \{y \in \mathcal{X} : \exists g \in G \text{ such that } y = g \star x\}$. Hence, we can embed the structure of a graph into the state space $\mathcal{X}$ by considering a group G that acts on the elements of $\mathcal{X}$. Although adhering to the assumptions that we previously made, defining the neighbours in this way could be inefficient. For sampling we do not need every possible edge in the graph to exist, it is only needed that there are enough edges such that every state can be reached. This idea has some resemblance to how only a basis is needed in order to be able to construct every vector in a vector space. Thus, it is natural to consider a generating set such that the graph structure can be simplified.

Definition 4.5. *Let $(G, \cdot)$ be a group, and let $\Gamma \subset G$. We say that Γ is generating set if $\langle \Gamma \rangle = G$, where $\langle \Gamma \rangle$ is defined by*

1. *$\Gamma \subset \langle \Gamma \rangle$.*
2. *$\gamma \in \langle \Gamma \rangle \implies \gamma^{-1} \in \langle \Gamma \rangle$.*
3. *$\gamma, \eta \in \langle \Gamma \rangle \implies \gamma \cdot \eta, \eta \cdot \gamma \in \langle \Gamma \rangle$.*

We also call the elements of Γ generators if $\langle \Gamma \rangle = G$.

Therefore, by taking a generator Γ for the group G that works on the state space $\mathcal{X}$, we can reduce the number of edges in the graph-like structure of the state space. This is significant since it reduces the computational complexity as each state x has less neighbours; each neighbour is given by applying the group action with an element from the generating set Γ to a state x in $\mathcal{X}$. Therefore, we need to calculate the ratio $\pi(y)/\pi(x)$ less often when we consider a move. However, the state space on which we sample remains unchanged as $G = \langle \Gamma \rangle$.

4.3 Tabu Sampler: Self Avoiding Walks

Backtracking behaviour is often harmful for the mixing of the simulated Markov chain. By repeatedly visiting the same states, the chain can get stuck in a small subset of the state space. This hinders convergence as the repeated samples are not at all independent. For example, a bimodal distribution with a low-probability region between the two modes usually results in a chain that gets stuck in the mode that was closest to its starting state. We saw that dependence of the current state on the starting state hints towards a chain that has not yet properly converged; this is precisely the way that the Gelman-Rubin diagnostic tests for convergence.

Similar to how we incorporated the information of the ratio $\pi(y)/\pi(x)$ to decrease the number of rejected states, we can keep track of the previously visited states to encourage the exploration of new regions of the state space. Before this can be done, it is important to study the order of the generators, since this determines how generators can be used to avoid back-tracking.

Definition 4.6. *Let $(G, \cdot)$ be a group, and let $g \in G \ \{e\}$. We say that g has order k if $g^k = e$, and if $g^j \neq e$ for $0 < j < k$. If no such k exists, then we say that g has infinite order. Whereas we say that e has order 1.*

Hence, if a generator γ has infinite order, then using it over and over would result in non-backtracking behaviour. On the other hand, a generator γ of order 2 that is re-used would result in the state we were in two moves ago as can be seen below,

$$\begin{aligned} &\text{State 1: } x; \\ &\text{State 2: } \gamma \star x; \\ &\text{State 3: } \gamma \star (\gamma \star x) = (\gamma \cdot \gamma) \star x = x. \end{aligned}$$

Furthermore, a generator γ of order 2 is also referred to as an *involution*. As we will only consider examples with involutions, non-backtracking behaviour can be induced by discouraging the re-use of generators. This is precisely the idea behind the Tabu sampler. However, before we can

introduce this sampler, we need to introduce the notion of skew-reversibility. Similar to standard reversibility, this provides a local and check-able condition to ensure that the MJP has π as its invariant distribution.

Definition 4.7. *[PG19] Let $\mathcal{X}$ be a discrete state space equipped with an involution $S : \mathcal{X} \to \mathcal{X}$. Let π be a probability measure on χ such that for all $x \in \mathcal{X}$, $\pi(x) = \pi(S(x))$. Let $Q : L^2(\pi) \to L^2(\pi)$ be the operator given by $\mathcal{Q}f(x) = f(S(x))$.*

A Markov process with generator $\mathcal{L}$ is skew-reversible with respect to the measure π and the involution S if, for all $f, g \in L^2(\pi)$, it holds that

$$\mathbb{E}_\pi[(\mathcal{Q}\mathcal{L}f)(x)g(x)] = \mathbb{E}_\pi[f(x)(\mathcal{Q}\mathcal{L}g)(x)]. \tag{4.6}$$

The following proposition shows that in certain cases skew-reversibility can be simplified.

Proposition 4.2. *By considering $f(x) = I[x = a]$, $g(x) = I[x = b]$, it can be shown that skew-reversibility is equivalent to*

$$\pi(x)\lambda(x \to y) = \pi(S(y))\lambda(S(y) \to S(x)) \qquad \text{for all } x, y \in \mathcal{X} \tag{4.7}$$

$$\Lambda(x) = \Lambda(S(x)) \qquad \text{for all } x \in \mathcal{X}. \tag{4.8}$$

Proof. Let $f(x) = I[x = a]$ and $g(x) = I[x = b]$, where we choose $a, b \in \mathcal{X}$ arbitrarily. First, notice the following

$$(\mathcal{Q}\mathcal{L}I[x = a])(x) = \sum_y \lambda(S(x) \to y)\{I[y = a] - I[S(x) = a]\}.$$

Hence, for the expectation this means the following

$$\begin{aligned}\mathbb{E}_\pi[(\mathcal{Q}\mathcal{L}f)(x)g(x)] &= \sum_x \sum_y \lambda(S(x) \to y)\{I[y = a] - I[S(x) = a]\}I[x = b]\pi(x) \\ &= \sum_y \lambda(S(b) \to y)\{I[y = a] - I[S(b) = a]\}\pi(b),\end{aligned}$$

where the sum could be simplified by considering the values for which x and y contributed to the summation. Similarly, for the other expectation,

$$\mathbb{E}_\pi[f(x)(\mathcal{Q}\mathcal{L}g)(x)] = \sum_y \lambda(S(a) \to y)\{I[y = b] - I[S(a) = b]\}\pi(a).$$

Now, if we suppose that $S(b) = a$, then the jumping rate $\lambda(S(b) \to a)$ does not exist, since jumping rates are only defined between two different states. Note that $S(b) = a$ also implies that $b = S(a)$, and vice versa, by applying the involution S to both sides of the equation. This means that if $S(a) = b$ holds, then also there is no jumping rate $\lambda(S(a) \to b)$.

Hence, if we suppose that $S(b) = a$, we obtain the following for both expressions,

$$\mathbb{E}_\pi[(\mathcal{Q}\mathcal{L}f)(x)g(x)] = \pi(b)\sum_y \lambda(a \to y);$$

$$\mathbb{E}_\pi[f(x)(\mathcal{Q}\mathcal{L}g)(x)] = \pi(S(b))\sum_y \lambda(S(a) \to y).$$

By using that $\pi(x) = \pi(S(x))$ for all $x \in \mathcal{X}$, we get that the following

$$\sum_y \lambda(a \to y) = \sum_y \lambda(S(a) \to y)$$

$$\Lambda(a) = \Lambda(S(a).$$

Remember that we choose $a, b \in \mathcal{X}$ arbitrary to see that we get $\Lambda(x) = \Lambda(S(x))$ for all $x \in \mathcal{X}$.

On the other hand, if we suppose that $S(b) \neq a$, then the following must hold

$$\pi(b)\lambda(S(b) \to a) = \pi(a)\lambda(S(a) \to b).$$

By how we choose $a, b \in \mathcal{X}$ and by the properties of S, we can set $b = S(x)$ and $a = y$. Again, remember that $\pi(x) = \pi(S(x))$ for all $x \in \mathcal{X}$ to obtain the final result:

$$\pi(x)\lambda(x \to y) = \pi(S(y))\lambda(S(y)) \to S(x).$$

Thus, we have proven the result. □

The skew-reversibility is needed as the Tabu sampler will leave π invariant on the augmented state space $(x, \alpha, \tau) \in \mathcal{X} \times \{\pm 1\}^{\Gamma} \times \{\pm 1\}$. Essentially, the state space is extended to be able to carry memory about the generators that were used previously. The idea is to assign to each generator γ a memory value of either 1 or -1, where the global memory variable $\tau \in \{\pm 1\}$ determines whether the generators carrying 1 or -1 can be used. If a generator γ was used its memory value $\alpha(\gamma)$ is set to $-\alpha(\gamma)$, and it cannot be used until the global memory variable τ is also flipped. This happens with a probability proportional to the sum of the jumping rates of all new states accessed by applying the generators to the current state x for which it holds that $\alpha(\gamma) = -\tau$. Therefore, the generators $\gamma \in \Gamma$ are partitioned in the following way,

$$\begin{aligned}
\Gamma(\alpha, \tau) &= \{\gamma \in \Gamma : \alpha(\gamma) = \tau\} \\
\Gamma(\alpha, -\tau) &= \{\gamma \in \Gamma : \alpha(\gamma) = -\tau\} \\
\Gamma(\alpha, \tau) \cap \Gamma(\alpha, -\tau) &= \emptyset \\
\Gamma(\alpha, \tau) \cup \Gamma(\alpha, -\tau) &= \Gamma
\end{aligned}$$

Intuitively, we have a set of generators that we can use ($\Gamma(\alpha, \tau)$) and cannot use ($\Gamma(\alpha, -\tau)$). If a generator is applied, then it is moved from the set of generators that we can move to the set of generators that we cannot. This process continues until we carry out a flip and swap the two sets, where the swap is performed by setting τ to $-\tau$. The exact procedure is detailed in Algorithm 4.

Algorithm 4 Tabu sampler for sampling $\pi(x)$ for $x \in \mathcal{X}$, when $\gamma^2 = e$ for all $\gamma \in \Gamma$

Given: λ (initial distribution), π (target distribution), $\{\alpha(\gamma)\}_{\gamma \in \Gamma} \in \{\pm 1\}^{\Gamma}$, $\tau \in \{\pm 1\}$
Goal: sample from $\pi(x)$ for $x \in \mathcal{X}$
Generate $X_0 \sim \lambda$
for $n = 1, \ldots, N$ **do**

1. For $\gamma \in \Gamma$ such that $\alpha(\gamma) = \tau$, compute $\lambda(\gamma; x, \alpha, \tau) = g\left(\frac{\pi(\gamma \star x)}{\pi(x)}\right)$.
2. For $\gamma \in \Gamma$ such that $\alpha(\gamma) = -\tau$, compute $\lambda(\gamma; x, \alpha, -\tau) = g\left(\frac{\pi(\gamma \star x)}{\pi(x)}\right)$.
3. Compute

$$\Gamma(x; , \alpha, \tau) = \sum_{\gamma \in \Gamma} \lambda(\gamma; x, \alpha, \tau) \cdot I[\alpha(\gamma) = \tau] \tag{4.9}$$

$$\Gamma(x; , \alpha, -\tau) = \sum_{\gamma \in \Gamma} \lambda(\gamma; x, \alpha, -\tau) \cdot I[\alpha(\gamma) = -\tau] \tag{4.10}$$

$$\Gamma(x; \alpha) = \max\{\Gamma(x; \alpha, \tau), \Gamma(x; \alpha, -\tau)\}. \tag{4.11}$$

4. Sample a waiting time $T \sim$ Exponential(rate $= \Gamma(x; \alpha)$), and advance time by T.
5. Generate $U \sim \mathcal{U}([0, 1])$.
   (a) If $U \leq \frac{\Gamma(x; \alpha, \tau)}{\Gamma(x; \alpha)}$:
      i. Sample a generator $\gamma \in \Gamma$ with probability $\frac{\lambda(\gamma; x, \alpha, \tau) I[\alpha(\gamma) = \tau]}{\Gamma(x; \alpha, \tau)}$.
      ii. Set the value of $\alpha(\gamma)$ to $-\alpha(\gamma)$.
      iii. Jump to $y = \gamma \star x$.
   (b) Else: flip τ to $-\tau$.

end for

The Tabu sampler essentially has a binary memory bank $\alpha(\gamma)$ for each generator γ. This allows the Tabu sampler to better navigate low probability regions between modes, since it prevents the process from re-using the previously applied generators over short to medium timescales. This non-reversibility is however, not guaranteed to last for a certain period of time, since the Tabu sampler is "self-tuning" by being able to flip the set of generators that can be used if the total event rate for the current state $x \in \mathcal{X}$ is much larger for the generators for which $I[\alpha(\gamma) = -\tau]$, i.e. if $\Gamma(x; \alpha, \tau)$ is heavily-dominated by $\Gamma(x; \alpha, -\tau)$. This is done in step 5 of the Tabu sampler algorithm, see algorithm 4. Hence, the Tabu sampler is able to correct its non-backtracking behaviour if more desirable states under the target are available.

The computational cost associated with running the Tabu sampler is comparable to that of the Zanella process, sometimes referred to as the Zanella sampler, since the sampling method also has the store α, τ and perform an accept-reject step for flipping τ. The total costs of these operations compared to the total use of computational resources is negligible. As the Tabu sampler is more resilient when it comes to clearing low energy barriers between modes, at no significant computational cost, it is expected that the Tabu performs better when the target distribution is multi-modal with larger barriers between the modes.

As a proxy to measure the "roughness" (multi-modality in combination with the amount of low energy barriers between modes), the mean reversion time for the Tabu sampler can be used. This is the average number of iterations, i.e. the number of times the steps $1-5$ are performed in algorithm 4, between two flips of the global memory variable τ. The reasoning behind this is that a high number of accepted moves: a lot of iterations before τ is flipped; implies that the distance between high-probability regions is larger. Indeed, a lot of low-probability moves performed subsequently will not greatly affect $\Gamma(x; \alpha, \tau)$.

4.4 Balancing function heuristic

In chapter 3 we discussed the result that balancing functions are Peskun optimal when it comes to incorporating local information in the MH proposal kernel [Zan17]. Merely in the specific case that the target distribution consists of independent Bernoulli variables, the Barker balancing function is known to be optimal. For a general target π no theoretical result exist on the optimal balancing function g to choose. Similarly, for the continuous time samplers as introduced in this chapter, the theory on the optimal balancing function to choose is yet to be established. Within this section we will briefly explore the influence of the balancing function on the mixing of the underlying Markov chain as discussed in [PG19]. In the end, a possible heuristic approach to finding a suitable balancing function choice is discussed.

Consider the *embedded jump chain* $(\hat{x})_{k\geq 1}$ of the Zanella process, i.e. the discrete time Markov chain consisting of the Markov process evaluated after each jump time τ_k. Hence, $\hat{x}_k = X_{\tau_k}$, with $(X_t)_{t\geq 0}$ the continuous time Markov chain generated by the Zanella process, where $\tau_0 = 0$. We will now consider the invariant distribution of the jump chain, which is determined by the choice of balancing function, and given by the following expression [PG19],

$$\pi_g^J(x) \propto \pi(x)\Lambda(x) = \pi(x)\left(\sum_{y\in\partial x}\lambda(x\to y)\right) = \pi(x)\left(\sum_{y\in\partial x} g\left(\frac{\pi(y)}{\pi(x)}\right)\right). \tag{4.12}$$

The idea behind studying the invariant distribution of the jump chain, is that it can be used as a proxy to study the effects of the balancing function on the mixing behaviour of the simulated chain. We will attempt to determine how similar the invariant distribution of the jumping chain is to the target π, since more similarity requires less correcting effort from the continuous sampler. As a metric for this similarity, the ratio π_g^J/π will be considered.

On the other hand, deviations can in fact be helpful if the target distribution is very peaked around its modes. In this case, a balancing function which lowers the energy barriers between the modes can be beneficial to the mixing behaviour as it becomes easier for the Markov chain to travel between the modes. Essentially, this entails increasing the metastability of the chain to further encourage mixing behaviour.

In [PG19], a simple one-dimensional example is considered on the state space $\mathcal{X} = [0, 50]$ with the neighbourhood structure $\partial x = \{x-1, x+1\}$. Among a few considered balancing functions compared for three distributions (Triangle, Beta-Binomial$(10, 20)$, and a marginal of a mixture of lattice Gaussians) the Barker balancing function $(g(t) = t/(1+t))$ seems to be the most suitable choice for all of these considered distributions. When compared to $g(t) = t$, $g(t) = \sqrt{t}$, and $g(t) = \min\{1, t\}$ (MH balancing function), the Barker balancing function is stable around the modes, finds a balance between distributing mass among high and low-probability regions, and is most stable in the tails of the distribution. In short, the Barker balancing function seems to be a safe and stable option. As the main aim of this work is to compare the samplers introduced in [Zan17] and [PG19], see chapter 3 and chapter 4, the Barker balancing function is the natural choice.

The proposed heuristic

In the general case, we know the target distribution π on the discrete state space $\mathcal{X}$ up to proportionality. The same holds for the invariant distribution of the jump chain π_g^J. Therefore, π_g^J is exactly equal to the target distribution if and only if

$$\pi_g^J(x)/\pi(x) \propto \Lambda(x) = K \text{ for all } x \in \mathcal{X}, \text{ for some } K \in \mathbb{R};$$

as a probability distribution is uniquely identifiable even if known up to proportionality. The idea is now to choose a balancing function for which this holds approximately. The first step is to sample a relatively large number of states $x \in \mathcal{X}$ from an overdispersed distribution, for example consider the overdispersed distribution on a discrete state space $\mathcal{X}$ as described in subsection 2.4.2. An overdispersed distribution is used to generate states $x \in \mathcal{X}$, since this slightly emphasizes the metastability of the chain, i.e. more probability mass is placed outside of the modes which encourages mixing behaviour. As a metric to proximity that the generated data-pairs $(x, \Lambda(x))$, where $x \in \mathcal{X}$ has been generated from the overdispersed distribution, have to being constant it is suggested that the coefficient of determination R^2 for the regression of the $\Lambda(x)$ data on a constant $K \in \mathbb{R}$ is considered.

As this regression procedure can be used to empirically tweak and construct a balancing function, it would be natural consider several balancing functions, i.e. functions g for which $g(t) = tg(1/t)$ for all $t > 0$. Furthermore, as we assumed in chapter 3, we will limit ourselves to functions of g for which it holds that $g(t) \leq at + b$ for all $t > 0$ and some $a, b \in \mathbb{R}$. A possible way to start could be considering candidate functions bounded by a linear function, and then to consider for each g for which the former holds, the balancing equivalent $\tilde{g}(t) = \min\{g(t), tg(1/t)\}$. After selecting the candidates, the best balancing function can be considered following the defined metric. Alternatively, linear combinations of the candidate functions could be considered as well, where an optimization algorithm could be used to determine the linear coefficients.

Chapter 5

Two well-known examples

Here we will present two examples that are commonly used to compare the performance of different MCMC samplers. These examples will be used to compare the discrete-time Metropolis-Hastings sampling with a random walk kernel and a locally-balanced pointwise informed proposal kernel to the continuous time Zanella and Tabu samplers.

5.1 Independent Binary Components

As a first toy example in a high-dimensional regime we will consider independent binary components [Zan17]. Hence, we find ourselves in the following setting: $\mathcal{X}^{(n)} = \{0,1\}^{(n)}$. Where the elements of $\mathcal{X}^{(n)}$ are denoted as $\mathbf{x} = (x_1, \dots, x_n)$, the target distribution takes the following form

$$\pi^{(n)}(\mathbf{x}) = \prod_{i=1}^{n} p_i^{1-x_i}(1-p_i)^{x_i}.$$

Where $p_i \in (0,1)$. The base kernel $K^{(n)}$ is the uniform distribution on the neighbourhood $N(\mathbf{x})$ given by the following expression

$$N(\mathbf{x}) = \{\mathbf{y} = (y_1, \dots, y_n) : \sum_{i=1}^{n} |x_i - y_i| = 1\}.$$

If we now assume that $\inf_{i\in\mathbb{N}} p_i > 0$ and $\sup_{i\in\mathbb{N}} p_i < 1$, then we obtain by Theorem 3.3 that locally-balanced proposals are asymptotically ($n \to \infty$) optimal when compared by using the Peskun ordering. Where indeed this hold asymptotically with respect to the dimension of the state space. This is especially relevant since the dimension of the state space is what determines the computational cost of calculating the normalizing constant and therefore the need to sample by using MCMC methods.

Proposition 5.1. *In the case of independent binary components, if we assume that* $\inf_{i\in\mathbb{N}} p_i > 0$ *and* $\sup_{i\in\mathbb{N}} p_i < 1$*, then for every* $g : (0,\infty) \mapsto (0,\infty)$ *a balancing function we obtain that pointwise informed proposals are asymptotically* ($n \to \infty$) *optimal in the Peskun ordering.*

Proof. We will show this by applying Theorem 3.3. It suffices to show that

$$c_g^{(n)} \to 1 \quad \text{as } n \to \infty$$

Fix $\mathbf{x} \in \{0,1\}^n$ and $y \in N(\mathbf{x})$. Hence, there exists an $i \in \{1, \dots, n\}$ such that $|x_i - y_i| = 1$. Furthermore, note that if $x_i = 1$, then $\frac{\pi^{(n)}(\mathbf{y})}{\pi^{(n)}(\mathbf{x})} = \frac{p_i}{1-p_i}$ and for $x_i = 0$; $\frac{\pi^{(n)}(\mathbf{y})}{\pi^{(n)}(\mathbf{x})} = \frac{1-p_i}{p_i}$. Thus, define

$$g_i^{\mathbf{x}} = \begin{cases} g(\frac{p_i}{1-p_i}) \text{ if } x_i = 1 \\ g(\frac{1-p_i}{p_i}) \text{ if } x_i = 0 \end{cases}.$$

Considering

$$I = \left[\frac{\inf_{i\in\mathbb{N}} p_i}{1-\inf_{i\in\mathbb{N}} p_i}, \frac{\sup_{i\in\mathbb{N}} p_i}{1-\sup_{i\in\mathbb{N}} p_i}\right] \cup \left[\frac{1-\sup_{i\in\mathbb{N}} p_i}{sup_{i\in\mathbb{N}}}, \frac{1-\sup_{i\in\mathbb{N}} p_i}{sup_{i\in\mathbb{N}}}\right],$$

we obtain that $\underline{g} = \inf_{t\in I} g(t) > 0$ and $\overline{g} = \sup_{t\in I} g(t) < \infty$ because g is locally bounded and I is compact. By looking at the expression for $Z_g^{(n)}(\mathbf{x})$

$$Z_g^{(n)}(\mathbf{x}) \;=\; \frac{1}{n}\sum_{j=1}^{n} g_j^x \;=\; \frac{1}{n}\left(\sum_{j=1,j\neq i}^{n} g_j^x + g_i^x\right),$$

it can be derived that

$$\sum_{j=1,j\neq i}^{n} g_j^x \;\geq\; \sum_{j=1,j\neq i}^{n} \underline{g} \;=\; (n-1)\underline{g} \;=\; O(n)$$

and

$$g_i^x \;\leq\; \overline{g} \;=\; O(1).$$

Note that for all $j \in \{1,\ldots,n\}\setminus\{i\}$ it holds that $g_j^x = g_j^y$ since by moving from $\mathbf{x}$ to $\mathbf{y}$ only the ith component has been flipped. Using this it follows that

$$\lim_{n\to\infty} \frac{Z_g^{(n)}(\mathbf{y})}{Z_g^{(n)}(\mathbf{x})} \;=\; \lim_{n\to\infty} \frac{\sum_{j=1,j\neq i}^{n} g_j^y}{\sum_{j=1,j\neq i}^{n} g_j^x} \;=\; \lim_{n\to\infty} \frac{\sum_{j=1,j\neq i}^{n} g_j^x}{\sum_{j=1,j\neq i}^{n} g_j^x} \;=\; 1.$$

Which is exactly what we needed to show. Now we have enough to apply Theorem 3.3, from which we can conclude optimality in terms of Peskun ordering. □

The unique feature of the independent binary component example is that we know the optimal balancing function among the class of balancing functions, namely the Barker balancing function: $g(t) = t/(1+t)$ [Zan17]. As it is shown that asymptotically for $n \to \infty$ the Barker balancing function leads to the smallest mixing time for all balancing functions used to incorporate local information into the Metropolis-Hastings proposal kernel. On the basis of this theoretical result, it is possible to directly infer whether continuous time samplers can in potential be an even more efficient MCMC sampling scheme.

For the continuous time sampling the generator set that we will use is:

$$\Gamma = \{i : i \in \{1,\ldots,n\}\}.$$

Thus, every generator refers to a component in the state vector $\mathbf{x}$, where by application of this generator we flip the component.

5.2 Weighted permutations

Let $S = \{1,\ldots,n\}$, then $\mathcal{S}_n$ is the space of permutations of n elements, i.e. all possible bijections from S to itself. The target distribution takes the following form

$$\pi^{(n)}(\rho) = \frac{1}{Z}\prod_{i=1}^{n} w_{i\rho(i)} \qquad \rho \in \mathcal{S}_n, \tag{5.1}$$

where Z is the normalizing constant $\sum_{\rho\in\mathcal{S}_n}\prod_{i=1}^{n} w_{i\rho(i)}$. Furthermore, $\{w_{ij}\}_{i,j}^{n}$ are positive and symmetric weights; $w_{ij} = w_{ji} > 0$ for all $i,j \in \{1,\ldots,n\}$ with $i \neq j$.

The neighbourhood structure for this example $\{N(\rho)\}_{\rho\in\mathcal{S}_n}$ is the following

$$N(\rho) = \{\rho' \in \mathcal{S}_n : \rho' = \rho \star (i,j) \text{ for some } i,j \in \{1,\ldots,n\} \text{ with } i \neq j\}, \tag{5.2}$$

where the group action $\star$ is defined by $\rho'(i) = \rho(j)$, $\rho'(j) = \rho(i)$, and $\rho'(l) = \rho(l)$ for $l \neq i$ and $l \neq j$.

Similar to proposition 5.1, we have a Peskun optimality result for pointwise informed proposals using balancing functions for the weighted permutations target distribution.

Proposition 5.2. *In the case of weighted permutations, if we assume that* $\inf_{i,j\in\mathbb{N}} w_{ij} > 0$ *and* $\sup_{i,j\in\mathbb{N}} w_{ij} < \infty$*, then for every* $g : (0,\infty) \mapsto (0,\infty)$ *a balancing function we obtain that pointwise informed proposals are asymptotically* $(n \to \infty)$ *optimal in the Peskun ordering.*

The proof is very similar to the proof for proposition 5.1 and therefore not stated.

As is not difficult to see with the defined neighbourhood structure, we will choose the following set of generators for the continuous time sampling:

$$\Gamma = \{i, j \in \{1, \ldots, n\} \text{ with } i \neq j\}. \tag{5.3}$$

Chapter 6

Bayesian Record Linkage

Bayesian record linkage will be studied as presented in [Zan17]. The aim is to merge two databases $\mathbf{x} = (x_1, \ldots, x_{N_1})$ and $\mathbf{y} = (y_1, \ldots, y_{N_2})$, where the records in both databases that refer to the same entity should be matched such that the merged database contains no redundant entries. For example, consider a government database with personal records, when we want to match two regional databases it is important not to have two files on the same person. This will be accomplished by keeping track of the matching denoted by $\mathbf{M} = (M_1, \ldots, M_{N_1})$, where $M_i = j$ if x_i and y_j refer to the same entity, and $M_i = 0$ if x_i is not matched with a record from $\mathbf{y}$. In addition, the database entries x_i and y_j are formatted in exactly the same way, where non-triviality occurs by the presence of a noise parameter β. Although most natural to include this as an unknown hyperparameter of the model, it is difficult to estimate and as stated by [Zan17], the efficiency of the sampling algorithms was not affected by different choices of β. Meaning that it is natural to choose a value for β, since this section intends to compare and analyze the performance differences between continuous and discrete-time samplers. For more discussion on this specific parameter and the caveats in estimation see [Ste14].

6.1 The generating model

We start by specifying the statistical model that is used to generate the databases $\mathbf{x}$ and $\mathbf{y}$. The first step is to sample the parameters λ and p_{match} from their respective priors, where λ is the mean of the Poisson distribution from which the total number of unique entities in the merged database is sampled. The p_{match} parameter is the probability that a record in the merged database occurs in both $\mathbf{x}$ and $\mathbf{y}$, i.e. the probability that two records in $\mathbf{x}$ and $\mathbf{y}$ are matched. The exact distributional dependencies assumptions will be explained later in this section, but the idea is that in the end we end up with the databases $\mathbf{x}$ and $\mathbf{y}$ with sampled records.

6.1.1 Composition of the x and y databases

We start by sampling λ and p_{match} from the following priors

$$p_{\text{match}} \sim \text{Unif}([0,1]) \tag{6.1}$$

$$\lambda \sim \pi(\lambda). \tag{6.2}$$

As p_{match} is a probability we have chosen a basic and uninformative prior. For λ we will consider an arbitrary prior distribution, albeit with positive support.

The total number of unique entries in the merged database T is sampled from the following distribution

$$T \mid \lambda \sim \text{Poisson}(\lambda). \tag{6.3}$$

Conditional on the matching probability (p_{match}) and the total number of records in the merged database (T), the number of matched records (N_m), the number of singletons in $\mathbf{x}$ (N_x), and the number of singletons in $\mathbf{y}$ (N_y) follow a multinomial distribution. The probability that a record

from the merged database is a matched record (occurs in both $\mathbf{x}$ and $\mathbf{y}$) is p_{match} and the probability that it occurs in either $\mathbf{x}$ or $\mathbf{y}$ as a singleton is $(1 - p_{\text{match}})/2$ for both the $\mathbf{x}$ and $\mathbf{y}$ database.

$$N_m, N_x, N_y \mid T, p_{\text{match}} \sim \text{Multinom}(T, (p_{\text{match}}, \frac{1 - p_{\text{match}}}{2}, \frac{1 - p_{\text{match}}}{2})) \tag{6.4}$$

The last step before sampling the records themselves, is sampling the total number of different ways that the number of sampled matchings can occur between $\mathbf{x}$ and $\mathbf{y}$, i.e. the possible configurations of $\mathbf{M}$ given the dimension of the vector ($N_1 = N_m + N_x$) and the number of non-zero entries (N_m). Furthermore, we assume a uniform distribution on the possible configurations of $\mathbf{M}$. Essentially, we are looking at the number of different ways in which we can pair N_m records from $\mathbf{x}$ with N_m records from $\mathbf{y}$, where we also need to look at the number of different ways in which we can choose N_m records in $\mathbf{x}$ and N_m records in $\mathbf{y}$. This is further explained in Figure 6.1, where also the corresponding entry in the $\mathbf{M}$ vector for each matching is added.

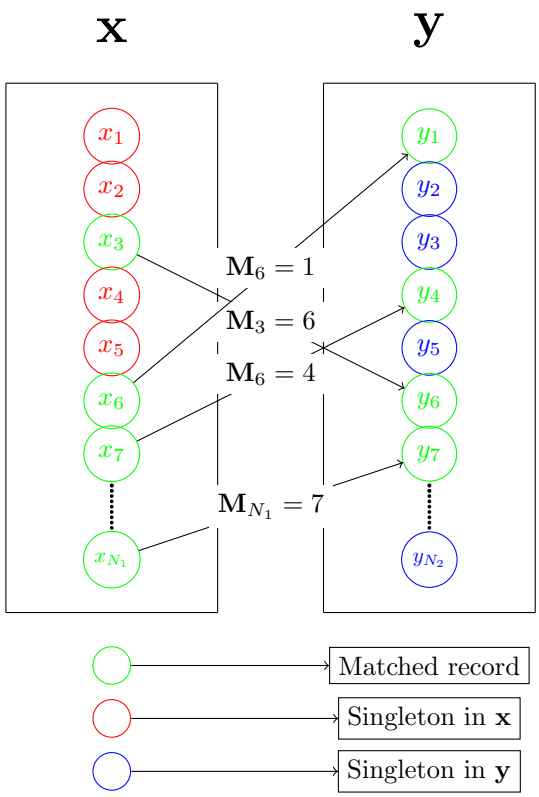

Figure 6.1: A schematic overview of a configuration of the $\mathbf{M}$ vector and how this translates to the $\mathbf{x}$ and $\mathbf{y}$ databases. Two green records are matched if connected by an arrow, the added $\mathbf{M}$ entry explains how this is captured in $\mathbf{M}$.

Hence, conditional on N_m, N_x, and N_y, the probability for each possible configuration of the $\mathbf{M}$ vector is the following

$$\begin{aligned} f_{\mathbf{M}|N_m,N_x,N_y}(\mathbf{m} \mid N_m = n_m, N_x = n_x, N_y = n_y) &= \binom{n_x + n_m}{n_m}^{-1} \binom{n_y + n_m}{n_m}^{-1} \frac{1}{n_m!} \\ &= \frac{n_m! n_x! n_y!}{(n_x + n_m)!(n_y + n_m)!} \end{aligned} \tag{6.5}$$

Note that for the total number of entries in $\mathbf{x}$, it holds that $N_1 = N_m + N_x$. Similarly, for $\mathbf{y}$ we get that $N_2 = n_m + N_y$. Therefore, this explains the appearance of the terms $n_x + n_m$ and $n_y + n_m$ in Equation 6.5. For an overview of the variables that describe the size of the databases, the reader is referred to Table 6.1.

Table 6.1: Variables and their meaning

Variable	Meaning
T	The total number of entries in the merged database
N_m	The total number of matched records (appearing in both $\mathbf{x}$ and $\mathbf{y}$)
N_x	The number of singletons in $\mathbf{x}$ (record only appearing in $\mathbf{x}$)
N_y	The number of singletons in $\mathbf{y}$ (record only appearing in $\mathbf{y}$)
N_1	The number of records in the $\mathbf{x}$ database
N_2	The number of records in the $\mathbf{y}$ database

6.1.2 Sampling the records

The distribution of the records $(\mathbf{x}, \mathbf{y}) \mid \mathbf{M}$ follows a hit and miss model as described by [CH90]. In this setting consider both databases to be lists of records $\mathbf{x} = (x_i)_{i=1}^{N_1}$ and $\mathbf{y} = (y_j)_{j=1}^{N_2}$. Where each record x_i contains l fields, by assumption each y_j has the same l fields. We will denote this by $(x_{is})_{s=1}^{l}$ and $(y_{js})_{s=1}^{l}$. Each field has m_s categories and density vector $\theta_s = (\theta_{sp})_{p=1}^{m_s}$. The density vector is assumed to be known, which is not a restrictive assumption since in data application this can always be estimated by the empirical distribution for each field. In the numerical simulations that will be explored later, each record was set to have 15 fields and the same 5 categories for each of these fields. The theta vector from which each of these fields is sampled was set to have the following form $\theta = (0.05, 0.15, 0.4, 0.2, 0.2)$.

An important assumption is now that the entities are assumed to be conditionally independent given the matching $\mathbf{M}$. In case x_i and y_j are singleton records, the fields are sampled in the following way

$$x_{is}, y_{js} \sim \theta_s \qquad s \in \{1, \dots, l\}.$$

If x_i and y_j are matched, then first a common value v_i is sampled in the same way as the singletons. The actual database entries x_i and y_j are then sampled conditionally on the common value v_i,

$$\begin{aligned} v_{is} &\sim \theta_s \qquad && s \in \{1, \dots, l\}. \\ x_{is}, y_{js} | v_s &\overset{iid}{\sim} \beta\theta_s + (1-\beta)\delta_{vs} \qquad && s \in \{1, \dots, l\}. \end{aligned}$$

Hence, with probability β, a field of the matched record is sampled as if it were a singleton. Since this could potentially cause the matched records to disagree in a field, β is called the distortion probability. Therefore, every field of a matched record is sampled from a mixture distribution, where with probability β the field is sampled as if it were a field of a singleton field and with probability $1-\beta$ it is set equal to the value in v_{is}. It is important to realize that this procedure is repeated for both the matched x_i and the matched y_j, explaining why *iid* is added above the matched record field distribution.

6.2 Joint probability distribution

Now, we have established all the (conditional) probability distributions from which the databases $\mathbf{x}$ and $\mathbf{y}$ can be sampled. Here we will combine these expressions and derive the form of the joint probability. We can write the joint probability in the following way

$$\begin{aligned} f_{\mathbf{x},\mathbf{y},\mathbf{M},N_m,N_x,N_y,T,\lambda,p_{\text{match}}}(\mathbf{x}, \mathbf{y}, \mathbf{m}, n_m, n_x, n_y, n_m + n_x + n_y, \lambda, p_{\text{match}}) = \\ f_{\mathbf{x},\mathbf{y}|\mathbf{M},N_m,N_x,N_y}(\mathbf{x}, \mathbf{y} \mid \mathbf{M} = \mathbf{m}, N_m = n_m, N_x = n_x, N_y = n_y) \cdot \\ f_{\mathbf{M},N_m,N_x,N_y,T|\lambda,p_{\text{match}}}(\mathbf{m}, n_m, n_x, n_y, n_m + n_x + n_y \mid \lambda, p_{\text{match}}) \cdot \\ \pi(\lambda) \mathbb{1}_{\{p_{\text{match}} \in [0,1]\}}. \end{aligned}$$

The term describing the probability contribution of $(\mathbf{x}, \mathbf{y}, \mid \mathbf{M}, N_m, N_x, N_y)$ will be referred to as the likelihood of the records. The term describing $(\mathbf{M}, N_m, N_x, N_y, T \mid \lambda, p_{\text{match}})$ will be referenced as the marginal of the matching structure. These two terms will be derived in the following two subsections to complete the derivation of the joint probability distribution.

6.2.1 Marginal of the matching structure

In this section we derive an expression for $f_{\mathbf{M},N_m,N_x,N_y,T|\lambda,p_{\text{match}}}$. Start by decomposing this term in the following way

$$\begin{aligned} f_{\mathbf{M},N_m,N_x,N_y,T|\lambda,p_{\text{match}}}(\mathbf{m}, n_m, n_x, n_y, n_m+n_x+n_y \mid \lambda, p_{\text{match}}) = \\ f_{\mathbf{M}|N_m,N_x,N_y}(\mathbf{m} \mid N_m = n_m, N_x = n_x, N_y = n_y)\cdot \\ f_{N_m,N_x,N_y|T,p_{\text{match}}}(n_m, n_x, n_y \mid T = n_m+n_x+n_y, p_{\text{match}})\cdot \\ f_{T|\lambda}(T \mid \lambda). \end{aligned}$$

Then, by using the distributional assumptions that we made previously,

$$\begin{aligned} &f_{N_m,N_x,N_y|T,p_{\text{match}}}(n_m, n_x, n_y \mid T = n_m+n_x+n_y, p_{\text{match}}) \cdot f_{T|\lambda}(T \mid \lambda) \\ &\quad= \left\{ \frac{(n_x+n_y+n_m)!}{(n_x)!(n_y)!(n_m)!} \left(\frac{1-p_{\text{match}}}{2}\right)^{n_x+n_y} p_{\text{match}}^{n_m} \right\} \cdot \left\{ \frac{e^{-\lambda}\lambda^{n_x+n_y+n_m}}{(n_x+n_y+n_m)!} \right\} \\ &\quad= \frac{e^{-\lambda}\lambda^{n_x+n_y+n_m}}{(n_x)!(n_y)!(n_m)!} \left(\frac{1-p_{\text{match}}}{2}\right)^{n_x+n_y} p_{\text{match}}^{n_m}. \end{aligned} \tag{6.6}$$

By now combining the expressions from Equation 6.5 and Equation 6.6, the full expression for the marginal of the matching structure is obtained,

$$\begin{aligned} f_{\mathbf{M},N_m,N_x,N_y,T|\lambda,p_{\text{match}}}(\mathbf{m}, n_m, n_x, n_y, n_m+n_x+n_y \mid \lambda, p_{\text{match}}) = \\ \frac{e^{-\lambda}\lambda^{n_x+n_y+n_m}}{(n_x+n_m)!(n_y+n_m)!} \left(\frac{1-p_{\text{match}}}{2}\right)^{n_x+n_y} p_{\text{match}}^{n_m}. \end{aligned} \tag{6.7}$$

6.2.2 Likelihood of the records

By sampling the records according to the procedure that we described previously, we obtain the following expression on the likelihood of the records $f_{\mathbf{x},\mathbf{y}|\mathbf{M},N_m,N_x,N_y}$. Let $\{i : M_i > 0\}$ be the set of indices $i \in \{1,\ldots,N_1\}$ for which $x_i \in \mathbf{x}$ is matched to $y_j \in \mathbf{y}$ for some $j \in \{1,\ldots,N_2\}$.

$$\begin{aligned} f_{\mathbf{x},\mathbf{y}|\mathbf{M},N_m,N_x,N_y}(\mathbf{x},\mathbf{y} \mid \mathbf{M} = \mathbf{m}, N_m = n_m, N_x = n_x, N_y = n_y) = \\ \prod_{s=1}^{l} \left(\left(\prod_{i=1}^{n_x+n_m} \theta_{sx_{is}} \right) \left(\prod_{j=1}^{n_y+n_m} \theta_{sy_{js}} \right) \left(\prod_{\{i:\mathbf{m}_i>0\}} \frac{P(x_{is}, y_{\mathbf{m}_i s} \mid \mathbf{M})}{\theta_{sx_{is}}\theta_{sy_{\mathbf{m}_i s}}} \right) . \right) \end{aligned}$$

Note that $n_x + n_m$ is equal to the number of records in the $\mathbf{x}$ database, since the $\mathbf{x}$ database is composed of singleton records n_x appearing only in $\mathbf{x}$, and matched records n_m appearing in both $\mathbf{x}$ and $\mathbf{y}$. Analogously, $n_y + n_m$ is equal to the number of records in the $\mathbf{y}$ database. The term $P(x_{is}, y_{M_i s} \mid \mathbf{M})$, which describes the sampling of a field of two records given that these are matched, can be decomposed by splitting over the cases $x_{is} = y_{M_i s}$ and $x_{is} \neq y_{M_i s}$. Start by considering $x_{is} = y_{M_i s}$:

$$\begin{aligned} P(x_{is} = y_{M_i s} \mid \mathbf{M}) &= (1-\beta)^2\theta_{sy_{\mathbf{m}_i s}} + 2\beta(1-\beta)\theta_{sx_{is}}\theta_{sy_{\mathbf{m}_i s}} + \beta^2\theta_{sx_{is}}\theta_{sy_{\mathbf{m}_i s}} \\ &= (1-\beta)^2\theta_{sy_{\mathbf{m}_i s}} + \beta(2-\beta)\theta_{sx_{is}}\theta_{sy_{\mathbf{m}_i s}}. \end{aligned}$$

Where $\theta_{sx_{is}} = \theta_{sy_{M_i s}}$, since we assumed that $x_{is} = y_{M_i s}$. In case both are equal to the sampled value of v_s, the term in which $(1-\beta)^2$ appears, we can replace this probability with $\theta_{sx_{is}}$ or $\theta_{sy_{M_i}s}$ since the v_s are sampled from the same distribution as the singletons. In case, $x_{is} \neq y_{M_i s}$:

$$\begin{aligned} P(x_{is} \neq y_{M_i s} \mid \mathbf{M}) &= 2\beta(1-\beta)\theta_{sx_{is}}\theta_{sy_{\mathbf{m}_i s}} + \beta^2\theta_{sx_{is}}\theta_{sy_{\mathbf{m}_i s}} \\ &= \beta(2-\beta)\theta_{sx_{is}}\theta_{sy_{\mathbf{m}_i s}}. \end{aligned}$$

Again we have used here that v_s is sampled from the same distribution as the singletons. By combining the two cases, we obtain the following

$$P(x_{is}, y_{M_i s} \mid \mathbf{M}) = \beta(2-\beta)\theta_{sx_{is}}\theta_{sy_{\mathbf{m}_i s}} + (1-\beta)^2\theta_{sy_{\mathbf{m}_i s}}\mathbb{1}(x_{is} = y_{\mathbf{m}_i s}).$$

Thus, if we now substitute this back into the expression we had starting out, then the expression for the likelihood can be obtained,

$$f_{\mathbf{x},\mathbf{y}|\mathbf{M},N_m,N_x,N_y}(\mathbf{x},\mathbf{y} \mid \mathbf{M}=\mathbf{m}, N_m=n_m, N_x=n_x, N_y=n_y) = \left(\prod_{s=1}^{l}\left(\prod_{i=1}^{n_x+n_m}\theta_{sx_{is}}\right)\left(\prod_{j=1}^{n_y+n_m}\theta_{sy_{js}}\right)\right)\cdot\prod_{s=1}^{l}\prod_{\{i:\mathbf{m}_i>0\}}\left(\beta(2-\beta)+\frac{(1-\beta)^2}{\theta_{sx_{is}}}(x_{is}=y_{\mathbf{m}_i s})\right). \tag{6.8}$$

6.2.3 The joint distribution

The complete expression for the joint distribution can be obtained by combining Equation 6.7, Equation 6.8, and the priors for λ and p_{match}. This results in the following

$$\begin{aligned} f_{\mathbf{x},\mathbf{y},\mathbf{M},N_m,N_x,N_y,T,\lambda,p_{\text{match}}}(\mathbf{x},\mathbf{y},\mathbf{m},n_m,n_x,n_y,n_m+n_x+n_y,\lambda,p_{\text{match}}) = \\ \frac{e^{-\lambda}\lambda^{n_x+n_y+n_m}}{(n_x+n_m)!(n_y+n_m)!}\left(\frac{1-p_{\text{match}}}{2}\right)^{n_x+n_y}p_{\text{match}}^{n_m} \\ \left(\prod_{s=1}^{l}\left(\prod_{i=1}^{n_x+n_m}\theta_{sx_{is}}\right)\left(\prod_{j=1}^{n_y+n_m}\theta_{sy_{js}}\right)\right) \\ \prod_{s=1}^{l}\prod_{\{i:\mathbf{m}_i>0\}}\left(\beta(2-\beta)+\frac{(1-\beta)^2}{\theta_{sx_{is}}}(x_{is}=y_{\mathbf{m}_i s})\right)\cdot\pi(\lambda)\cdot\mathbb{1}_{\{p_{\text{match}}\in[0,1]\}} \end{aligned} \tag{6.9}$$

6.3 Posterior distribution

6.3.1 Metropolis-within-Gibbs sampler

After having established the statistical model for the Bayesian record linkage. A Metropolis-within-Gibbs scheme is used to sample from the posterior distribution $(\mathbf{M},\lambda,p_{\text{match}}) \mid (\mathbf{x},\mathbf{y})$ [Zan17]. Given $(\mathbf{x},\mathbf{y},\mathbf{M})$ the two parameters λ and p_{match} are conditionally independent, with the following distributions:

$$p_{\text{match}} \mid \mathbf{x},\mathbf{y},\mathbf{M} \sim \text{Beta}(1+N_m, 1+N_1+N_2-2N_m) \tag{6.10}$$

$$\lambda \mid \mathbf{x},\mathbf{y},\mathbf{M} \sim \text{Gamma}_{[\max\{N_1,N_2\},N_1+N_2]}(1+N_1+N_2-N_m,1). \tag{6.11}$$

Where the $\text{Gamma}_{[\min\{N_1,N_2\},N_1+N_2]}$ refers to the truncated Gamma distribution. This means that for $X \sim$ Gamma, $X|\{X\in[\min\{N_1,N_2\},N_1+N_2]\} \sim \text{Gamma}_{[\min\{N_1,N_2\},N_1+N_2]}$. Sampling from the posterior distributions for λ and p_{match} is straightforward. However, for the posterior distribution of $\mathbf{M}$ this is less straightforward. Thus, MCMC sampling will be used to generate insight into this distribution, which is proportional to

$$P(\mathbf{M} \mid \mathbf{x},\mathbf{y},\lambda,p_{\text{match}}) \propto \prod_{\{i:\mathbf{m}_i>0\}}\left(\frac{4p_{\text{match}}}{\lambda(1-p_{\text{match}})^2}\prod_{s=1}^{l}\left(\beta(2-\beta)+\frac{(1-\beta)^2}{\theta_{sx_{is}}}\mathbb{1}(x_{is}=y_{\mathbf{m}_i s})\right)\right). \tag{6.12}$$

This sampling scheme alternates between Gibbs updates for λ and p_{match}, and Metropolis-Hastings updates for $\mathbf{M}$. The challenge lies in efficiently updating the matchings for $\mathbf{M}$. This is also where continuous-time samplers will be compared to the pointwise informed proposals as found in [Zan17]. The basis of the Metropolis-Hastings sampling algorithm is the random walk proposal kernel $Q(\mathbf{M},\mathbf{M}')$. It starts by sampling a random couple $(i,j)\in\{1,\ldots,N_1\}\times\{1,\ldots,N_2\}$. Based on the current configuration of the matching $\mathbf{M}$, an update is made to obtain the proposed state $\mathbf{M}'$. As the move that is performed depends on whether the x_i and y_j are currently matched to another record, we also need to be able to reason about the record in x to which the record y_j could potentially be coupled, i.e. to see whether it is still considered a singleton. Therefore, define

the following

$$\mathbf{M}_j^{-1} = \begin{cases} i \in \{1, \ldots, N_1\} \text{ such that } M_i = j & \{i \in \{1, \ldots, N_1\} : M_i = j\} \neq \emptyset \\ 0 & \text{otherwise.} \end{cases} \tag{6.13}$$

Note that $M_j^{-1} = i$, if and only if $M_i = j$. Hence, instead of reasoning about the matches from the entries in the $\mathbf{x}$ database, we reason from the $\mathbf{y}$ database when we consider M_j^{-1}. Thus, given a current $\mathbf{M}$ and a random couple (i, j), the following operations are performed to obtain the proposal state $\mathbf{M}'$.

Algorithm 5 Updating $\mathbf{M}$

Given: $\mathbf{M}$ (current configuration), (i, j) (random couple)
Goal: update to $\mathbf{M}'$

if $M_i = 0$ and $M_j^{-1} = 0$ **then**
  Move 1: $M_i = j$
else if $M_i = j$ **then**
  Move 2: $M_i = 0$
else if $M_i = 0$ and $M_j^{-1} = i'$ for some $i' \in \{1, \ldots, N_1\}$ **then**
  Move 3: $M_i = j$ and $M_{i'} = 0$
else if $M_i = j'$ for some $j' \in \{1, \ldots, N_2\}$ and $M_j^{-1} = 0$ **then**
  Move 4: $M_i = j$
else if $M_i = j'$ for some $j' \in \{1, \ldots, N_2\}$ and $M_j^{-1} = i'$ for some $i' \in \{1, \ldots, N_1\}$ **then**
  Move 5: $M_i = j$ and $M_{i'} = j'$
end if

The accept-reject probability of this proposal kernel, for standard Metropolis-Hastings, takes the following form:

$$\alpha(\mathbf{M}, \mathbf{M}') = \min\left\{1, \frac{P(\mathbf{M}' \mid \mathbf{x}, \mathbf{y}, \lambda, p_{\text{match}})}{P(\mathbf{M} \mid \mathbf{x}, \mathbf{y}, \lambda, p_{\text{match}})}\right\}. \tag{6.14}$$

Note that the part of the ratio relating to the proposal kernel Q always vanishes, since $Q(\mathbf{M}', \mathbf{M}) = Q(\mathbf{M}, \mathbf{M}')$ for every $\mathbf{M} \in \chi$ and every $\mathbf{M}' \in \chi$ such that $\mathbf{M}' = (i, j) \star \mathbf{M}$. As every pair $(i, j) \in \{1, \ldots, N_1\} \times \{1, \ldots, N_2\}$ is drawn with equal probability, we only need to count the number of pairs which correspond to a transition from $\mathbf{M}$ to $\mathbf{M}'$ and the number of pairs with which we can go back from $\mathbf{M}'$ to $\mathbf{M}$, to ensure that the proposal kernel Q is symmetric. We will do this by considering each of the five possible moves as stated in Algorithm 5.

Before we will do this, the situations in which each one of the possible five moves occurs will be explained. Note that a chosen generator is always of the form (i, j), where $i \in \{1, \ldots, N_1\}$ refers to an index of a record in $\mathbf{x}$ and $j \in \{1, \ldots, N_2\}$ to an index of a records in $\mathbf{y}$. Thus, when a generator (i, j) is drawn we want to try the matching x_i and y_j. However, the exact action that we propose also depends on the the current configuration of the vector $\mathbf{M}$, as can be seen in Table 6.2.

Table 6.2: An overview of the five moves

Move	**Situation**	**Action**
1	x_i and y_j are currently seen as singletons	x_i and y_j are matched
2	x_i and y_j are currently matched	unmatch and consider both x_i and y_j as singletons
3	x_i is seen as a singleton and y_j is matched to some $x_{i'}$	match x_i and y_j, $x_{i'}$ is considered a singleton
4	x_i is matched to some $y_{j'}$ and y_j is considered a singleton	match x_i and y_j, and set $y_{j'}$ to be a singleton
5	x_i is matched to some $y_{j'}$, and y_j is matched to some $x_{i'}$	match x_i and y_j, and match $x_{i'}$ and $y_{j'}$

Suppose that $\mathbf{M}$ transitions to $\mathbf{M}'$ by a move 1 operation, then this can only occur if the unique pair (i, j) is sampled, where the i and j are as described in the updating algorithm. Hence, in the

case of a type 1 move, both the x_i and the y_j record were unmatched in $\mathbf{M}$, and are linked to each other in $\mathbf{M}'$ as $M_i = j$. On the other hand, $\mathbf{M}'$ can only be changed back to $\mathbf{M}$ if the proposed move for the pair (i, j) is accepted, since this deletes the newly made matching $M_i = j$ only if (i, j) were matched in the previous configuration. Therefore, if $\mathbf{M}$ becomes $\mathbf{M}'$ by a type 1 move, then for these two states the proposal kernel is symmetric. Similarly, we can see that a type 2 move is also only obtained for a unique pair (i, j), where the move is set back if the previously deleted linked records $M_i = j$ are linked again.

Alternatively, suppose that $\mathbf{M}$ is set to $\mathbf{M}'$ by a type 3 move. This means that we have sampled a pair (i, j), where the x_i record is currently unmatched and the y_j is matched to some $x_{i'}$, i.e. $M_{i'} = j$. After performing the move, we set $M_i = j$ and match the x_i and the y_j record, and we unmatch the y_j record. As we set $M_i = j$, the sampled pair (i, j) uniquely determines the state $\mathbf{M}'$, meaning that we can only go from $\mathbf{M}$ to $\mathbf{M}'$ in one way. When we want go back from $\mathbf{M}'$ to $\mathbf{M}$, we need to match the pair $x_{i'}$ and y_j, and unmatch the pair X_i and y_j. This can be done by sampling the pair (i', j), which is again a type 3 move, where y_j is already matched to x_i and $x_{i'}$ is still unmatched. Hence, after setting $M_{i'} = j$ and unmatching x_i we are back at state $\mathbf{M}$ again.

In case a type 4 move is performed to go from $\mathbf{M}$ to $\mathbf{M}'$, the situation is similar to the one for a type 3 move, only with the $\mathbf{x}$ and $\mathbf{y}$ databases inverted. Again, a pair (i, j) is sampled where x_i is already matched to some $y_{j'}$, hence $M_i = j'$, and y_j unmatched. When performing the move we set $M_i = j$ and unmatch $y_{j'}$. This can indeed only be done in one way since we set $M_i = j$ for the sampled pair (i, j). If we now want to go back from $\mathbf{M}'$ to $\mathbf{M}$, then we need to sample the pair (i, j'). After which we set $M_i = j'$ and unmatch the record y_j, which means that we are back in the state $\mathbf{M}$.

Lastly, the case in which a type 5 move is performed to go from $\mathbf{M}$ to $\mathbf{M}'$. If the pair (i, j) is sampled where x_i and y_j are already currently matched, i.e. $M_i = j'$ and $M_{i'} = j$ for some $x_{i'}$ and $y_{j'}$, then we switch both the matched pairs and set $Mi = j$ and $M_{i'} = j'$. As the two pairs of record are essentially swapped with respect to their current matching, the same state $\mathbf{M}'$ could have been reached if the pair (i', j') was sampled. Similarly, if we want to go from $\mathbf{M}'$ back to $\mathbf{M}$, we can do this by sampling either (i', j) or (i, j') by the same reasoning as before. Therefore, this shows that the proposal kernel Q is symmetric, and further explains the way in which the current state $\mathbf{M}$ is updated.

6.3.2 Continuous-time sampling

Continuous-time sampling, with the Zanella and the Tabu sampler, will be performed for the posterior distribution $\mathbf{M} \mid \mathbf{x}, \mathbf{y}, \lambda, p_{\text{match}}$. The state space $\mathcal{X}$ contains all the possible maps $\psi : \{1, \ldots, N_1\} \to \{1, \ldots, N_2\} \cup \{0\}$ such that the map is injective when restricted to $\psi^{-1}[\{1, \ldots, N_2\}]$. By construction of the forward model, only two records can refer to the same entity. Whereas, each database is not limited to a certain number of singletons a priori.

The neighbouring structure of the state space takes the following form

$$\partial M = \{\mathbf{M}' : \exists (i, j) \in \{1, \ldots, N_1\} \times \{1, \ldots, N_2\} \text{ such that } \mathbf{M}' = (i, j) \star \mathbf{M}\}. \tag{6.15}$$

Where the group action $\star$ is defined by the steps described in Algorithm 5. Hence, the considered generators will be $(i, j) \in \{1, \ldots, N_1\} \times \{1, \ldots, N_2\}$.

6.3.3 Derivation of the posterior

Here we will derive the posteriors from which we sample in the Metropolis-within-Gibbs sampler. The starting point will indeed be the joint distribution as stated in Equation 6.9. Firstly, we will replace the random variables N_x and N_y, the number of singleton records in $\mathbf{x}$ and $\mathbf{y}$ respectively, with the random variables N_1 and N_2. As N_1 and N_2 refer to the lengths of the $\mathbf{x}$ and $\mathbf{y}$ database, this is more convenient when we observe $\mathbf{x}$ and $\mathbf{y}$ directly. In addition, this substitution is straightforward as it holds that $N_1 = N_x + N_m$ and $N_2 = N_y + N_m$. Lastly, we will change to Bayesian notation when we refer to the posterior distributions.

The first step is to condition on the databases $\mathbf{x}$ and $\mathbf{y}$, from which it follows that we end up with the following conditional density:

$$f_{\mathbf{M}, N_m, N_1, N_2, T, \lambda, p_{\text{match}} \mid \mathbf{x}, \mathbf{y}}(\mathbf{m}, n_m, n_1, n_2, n_1 + n_2 - n_m, \lambda, p_{\text{match}} \mid \mathbf{x}, \mathbf{y}) = \frac{f_{\mathbf{x}, \mathbf{y}, \mathbf{M}, N_m, N_1, N_2, T, \lambda, p_{\text{match}}}(\mathbf{x}, \mathbf{y}, \mathbf{m}, n_m, n_1, n_2, n_1 + n_2 - n_m, \lambda, p_{\text{match}})}{f_{\mathbf{x}, \mathbf{y}}(\mathbf{x}, \mathbf{y})}.$$

Since the two databases $\mathbf{x}$ and $\mathbf{y}$ are observed, the term $f_{\mathbf{x},\mathbf{y}}(\mathbf{x},\mathbf{y})$ is essentially a constant under this observed information. Therefore, the posterior density is proportional to the joint,

$$f_{\mathbf{M},N_m,N_1,N_2,T,\lambda,p_{\text{match}}|\mathbf{x},\mathbf{y}}(\mathbf{m},n_m,n_1,n_2,n_1+n_2-n_m,\lambda,p_{\text{match}} \mid \mathbf{x},\mathbf{y}) \propto \\ f_{\mathbf{x},\mathbf{y},\mathbf{M},N_m,N_1,N_2,T,\lambda,p_{\text{match}}}(\mathbf{x},\mathbf{y},\mathbf{m},n_m,n_x,n_y,n_1+n_2-n_m,\lambda,p_{\text{match}}).$$

Before we can derive the posteriors that we use to sample from the Metropolis-within-Gibbs sampler, we need to choose the specific priors for λ and p_{match}. For λ we have chosen a uniform distribution on the range of possible sizes of the merged database, namely $[\max\{N_1,N_2\},N_1+N_2]$. Since p_{match} is a probability, we have opted for a uniform distribution on the interval $[0,1]$. Resulting in the following priors:

$$\lambda \sim \text{Unif}([\max\{N_1,N_2\},N_1+N_2]); \tag{6.16}$$

$$p_{\text{match}} \sim \text{Unif}([0,1]). \tag{6.17}$$

Furthermore, we can omit the explicit mention of the random variables N_m, N_1, N_2, and T by noticing a few things:

$$N_1 : \text{The number of records in } \mathbf{x}; \tag{6.18}$$

$$N_2 : \text{The number of records in } \mathbf{y}; \tag{6.19}$$

$$N_m = \#\{i : M_i > 0\}; \tag{6.20}$$

$$T = N_1 + N_2 - N_m. \tag{6.21}$$

Thus, by being a conditional probability density for $\mathbf{m},\lambda,p_{\text{match}} \mid \mathbf{x},\mathbf{y}$, given the databases $\mathbf{x}$ and $\mathbf{y}$ and for a certain vector $\mathbf{m}$ all of the random variables in Equation 6.18 are already determined. This means that we do not need to explicitly mention them in the conditional density. Therefore, this fact and the specific priors lead to the following conditional density:

$$f_{\mathbf{M},\lambda,p_{\text{match}}|\mathbf{x},\mathbf{y}}(\mathbf{m},\lambda,p_{\text{match}} \mid \mathbf{x},\mathbf{y}) \propto \\ \frac{e^{-\lambda}\lambda^{n_1+n_2-n_m(\mathbf{m})}}{n_1!n_2!}\left(\frac{1-p_{\text{match}}}{2}\right)^{n_1+n_2-2n_m} p_{\text{match}}^{n_m} \\ \prod_{s=1}^{l}\prod_{\{i:\mathbf{m}_i>0\}}\left(\beta(2-\beta)+\frac{(1-\beta)^2}{\theta_{sx_{is}}}(x_{is}=y_{\mathbf{m}_i s})\right) \\ \cdot \mathbb{1}(p_{\text{match}}\in[0,1]) \cdot \frac{\mathbb{1}(\lambda\in[\max\{n_1,n_2\},n_1+n_2])}{\min\{n_1,n_2\}}.$$

Now, we can derive the posterior distributions for the Gibbs sampling scheme. From this point we will also replace the conditional densities with the Bayesian conditional probability notation. Note that conditional on $(\mathbf{x},\mathbf{y},\mathbf{M})$ the two parameters λ and p_{match} are conditionally independent [Zan17]. We start by considering $p_{\text{match}} \mid \mathbf{x},\mathbf{y},\mathbf{M}$. Then we obtain

$$P(p_{\text{match}} \mid \mathbf{x},\mathbf{y},\mathbf{M}) \propto p_{\text{match}}^{n_m}(1-p_{\text{match}})^{n_1+n_2-n_m} \\ \propto \frac{p_{\text{match}}^{n_m}(1-p_{\text{match}})^{n_1+n_2-2n_m}}{\mathcal{B}(1+n_m,1+n_1+n_2-2n_m)}.$$

Which is exactly the density function of a $\text{Beta}(1+N_m,1+N_1+N_2-2N_m)$ distribution.

Similarly, by considering $\lambda \mid \mathbf{x},\mathbf{y},\mathbf{M}$, it can be concluded that

$$P(\lambda \mid \mathbf{x},\mathbf{y},\mathbf{M}) \propto e^{-\lambda}\lambda^{n_1+n_2-n_m}\mathbb{1}(\lambda\in[\max\{n_1,n_2\},n_1+n_2]) \\ \propto \frac{e^{-\lambda}\lambda^{n_1+n_2-n_m}\mathbb{1}(\lambda\in[\max\{n_1,n_2\},n_1+n_2])}{\Gamma(\alpha)P(\lambda\in[\max\{n_1,n_2\},n_1+n_2] \mid \mathbf{x},\mathbf{y},\mathbf{M})}.$$

We see that this is exactly the density function of a truncated (or conditional) $\text{Gamma}_{[\min\{N_1,N_2\},N_1+N_2]}(1+N_1+N_2-N_m,1)$ distribution.

Lastly, we consider $\mathbf{M} \mid \mathbf{x},\mathbf{y},\lambda,p_{match}$,

$$P(\mathbf{M} \mid \mathbf{x},\mathbf{y},\lambda,p_{\text{match}}) \propto \frac{p_{\text{match}}4^{n_m}}{\lambda(1-p_{\text{match}})^{n_m}}\cdot \\ \prod_{s=1}^{l}\prod_{\{i:\mathbf{m}_i>0\}}\left(\beta(2-\beta)+\frac{(1-\beta)^2}{\theta_{sx_{is}}}(x_{is}=y_{\mathbf{m}_i s})\right).$$

Now, we can get rid of the N_m random variable by noting that $N_m = \#\{i : M_i > 0\}$ and by rearranging the term in the product operator. Therefore, we end up with the following posterior (up to proportionality):

$$P(\mathbf{M}|\mathbf{x}, \mathbf{y}, \lambda, p_{\text{match}}) \propto \prod_{\{i:\mathbf{m}_i>0\}} \left(\frac{4p_{\text{match}}}{\lambda(1-p_{\text{match}})^2} \prod_{s=1}^{l} \left(\beta(2-\beta) + \frac{(1-\beta)^2}{\theta_{sx_{is}}} \mathbb{1}(x_{is} = y_{\mathbf{m}_i s}) \right) \right).$$

Chapter 7

Numerical simulations

In this section we will perform numerical simulations using the target distributions that arise from the examples as presented in chapter 5 and chapter 6. All of these simulations were performed using code written in Python 3 and available at `https://github.com/BjarneHerben/CodeDiscreteStochSim`. The aim of these simulations is to compare the pointwise informed proposals as described in chapter 3 and the continuous time samplers introduced in chapter 4. The connection between these sampler philosophies is that both use the same balancing functions to incorporate local information about the target in the sampling process. Therefore, as the aim is to compare these approaches, the Barker balancing function $g(t) = t/(1+t)$ has been used for each sampler.

7.1 The considered MCMC schemes

In this section four MCMC schemes are compared: random walk Metropolis-Hastings (MH), pointwise informed proposal with the Barker balancing function, the Zanella process, and the Tabu sampler. The random walk MH and the informed proposal sampler are discrete time Markov processes. Whereas, the Zanella and the Tabu sampler are Markov Jump Processes (MJP) with the jumping rates being calculated with the Barker balancing function, where the rate to jump from $y \in \mathcal{X}$ to $x \in \mathcal{X}$ is calculated by $g(\pi(y)/\pi(x))$. With $\mathcal{X}$ the considered discrete state space and π the target density.

The starting point for the discrete-time samplers is the uniform kernel on a neighbourhood of the current state x, hence resulting in the following kernel: $K(x, \cdot) = \text{Unif}(N(x))$, with $N(x)$ the neighbourhood of x. In the considered examples, these neighbourhoods are regular in the sense that every state has the same number of neighbours. For example, in the case of independent binary components this amounts to each state that can be accessed by flipping one component. For the continuous time samplers, we reason not from the perspective of possible moves, but from generators that can be applied to the current state to obtain a new one, i.e. a move is performed when a generator is applied to the current state. However, by indexing each possible move as a generator that can be applied, this translates to a similar traversion of the state space.

7.2 Continuous-time implementation

As described by algorithm 3 and algorithm 4, a MJP can be simulated directly on a discrete state space. Therefore, if we wish to estimate $\mathbb{E}_\pi[f(X)]$ for $f : \chi \to \mathbb{R}$ by simulating a MJP $(X)_{t\in[0,T]}$, we will obtain an estimate that is close to the actual value of $\mathbb{E}_\pi[f(X)]$ if we simulate the Markov chain long enough. This follows from the fact the empirical average of the process converges to the actual value in the limit [PG19]:

$$\lim_{T\to\infty} \frac{1}{T} \int_{[0,T]} f(X_s)\, ds = \mathbb{E}_\pi[f(X)]. \tag{7.1}$$

The integral on the left-hand side, which is used as an estimator for some T finite, will be calculated by applying a thinning procedure to the simulated Markov chain. A value $0 < \phi < T$ is chosen such that T/ϕ is an integer. At each multiple of T/ϕ the state of the process is recorded. Furthermore,

ϕ is also chosen to be approximately the mean event time after the MJP has reached stationarity. Therefore, this would result in the following estimate for the expectation:

$$\mathbb{E}_\pi[f(X)] \approx \frac{\phi}{T} \sum_{i=0}^{\frac{T}{\phi}} f(X_i). \tag{7.2}$$

7.3 Independent Binary Components

Simulating independent binary components is a toy example, since we are in fact able to simulate samples from this target density directly. However, as we know that the Barker balancing function ($g(t) = t/(1+t)$) is the optimal balancing function leading to the smallest mixing time of the chain [Zan17], it is interesting to consider whether the continuous time samplers could still be an improvement over the optimal pointwise informed sampler.

Recall from section 5.1 that $\mathbf{x} \in \chi^{(n)} = \{0,1\}^{(n)}$, where the target distribution takes the following form:

$$\pi^{(n)}(\mathbf{x}) = \prod_{i=1}^{n} p_i^{1-x_i}(1-p_i)^{x_i}. \tag{7.3}$$

The parameter n is used to control the dimensionality of the state space and has been set to $n = 1000$ for the considered simulations. In addition, each probability p_i has been simulated from a uniform distribution, i.e. $p_i \sim \text{Unif}([0,1])$ for $i \in \{1, \ldots, n\}$.

Figure 7.1: Traceplot of the log energy of the target density of the four considered samplers for independent binary components.

The traceplots in Figure 7.1 and Figure 7.2 are similar and suggest that the Zanella process and Tabu sampler are performing the best, where the Zanella process is doing marginally better for the effective sample sizes. As expected, the random walk MH sampler performs the worst with the pointwise informed proposal being a considerable improvement over the aforementioned sampling scheme. In the autocorrelation plot in Figure 7.3, the same conclusion can be drawn, only differing in the fact that the Zanella process and Tabu sampler are indistinguishable when it comes to the autocorrelation of the summary statistics.

The Gelman-Rubin diagnostic, see Table 7.1, suggests that all samplers do indeed properly convergence. The Tabu sampler converges slightly faster than the Zanella process. This is to be expected as the Tabu sampler is non-reversible on short to medium timescales. However, the

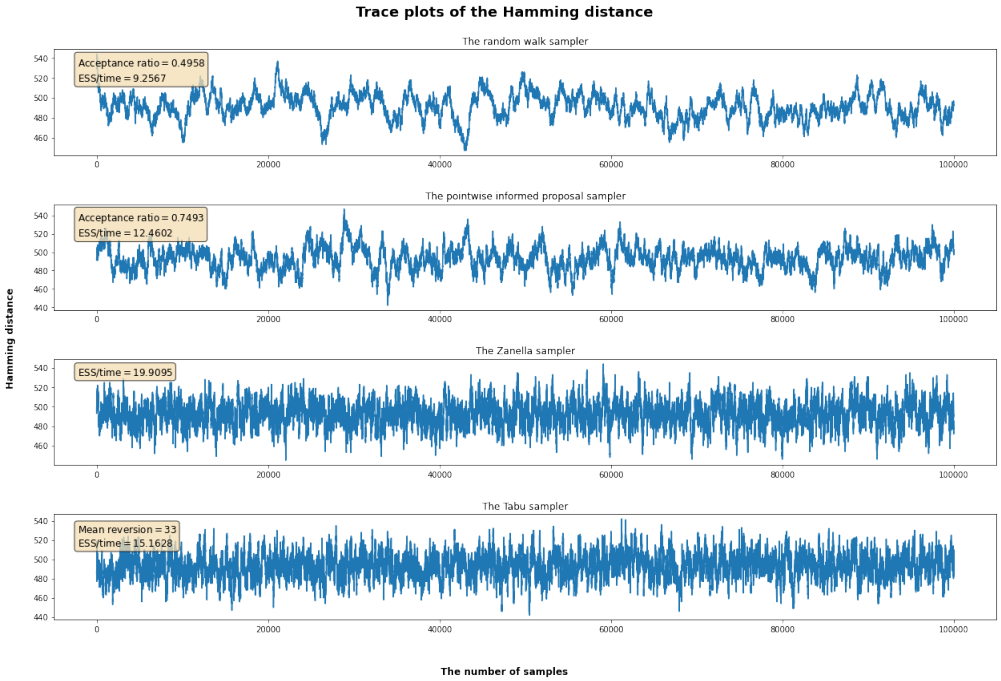

Figure 7.2: Traceplot of the Hamming distance to the state where each binary component is flipped to 1.

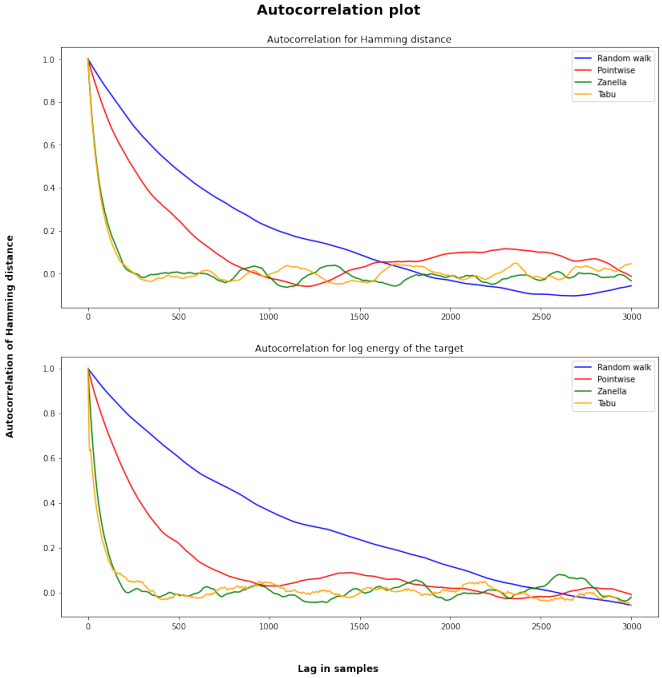

Figure 7.3: Autocorrelation plot for all four samplers for the two considered summary statistics: log energy of the target density and Hamming distance. The autocorrelation has been calculated for the traces as displayed in Figure 7.1 and Figure 7.2.

difference is hardly significant by the perceived smoothness of the target distribution, since the mean reversion time is merely 33 for the Tabu sampler. Therefore, in this case the Tabu sampler and Zanella process perform equally well, where the Zanella process has a higher effective sample size per unit of runtime as each iteration requires slightly less computations when compared to the Tabu sampler.

The main takeaway is that the continuous time samplers do seem to be able to outperform the

Sampler	**N = 1000**	**N = 5000**	**N = 10000**
Random walk	1.4986	1.0698	1.0480
Pointwise sampler	1.3327	1.0332	1.0675
Zanella sampler	1.0703	1.0259	1.0092
Tabu sampler	1.0376	1.0146	1.0051

Table 7.1: The $\hat{R}$ value of the Gelman-Rubin diagnostic of the four considered samplers for different values of N. For each iterative simulation $m = 10$; 10 different iterative sequences were simulated for comparison.

pointwise informed proposal sampler in spite of the former being optimal for this class of MCMC schemes.

7.4 Weighted permutations

Sampling from a distribution of weighted permutations is a non-trivial task often accomplished by MCMC schemes. Recall from section 5.2, that we have following target density

$$\pi^{(n)}(\rho) \propto \prod_{i=1}^{n} w_{i\rho(i)}.$$

The base kernel $K(\rho, \cdot)$ is uniform on the neighbourhood defined in Equation 5.2. The generators used for the Tabu sampler and Zanella process are defined in Equation 5.3.

Sampler	$\lambda = \mathbf{1}$	$\lambda = \mathbf{2}$	$\lambda = \mathbf{3}$	$\lambda = \mathbf{4}$	$\lambda = \mathbf{5}$
Random walk	0.3876	0.1816	0.1620	0.1422	0.1322
Pointwise sampler	0.7154	0.7778	0.8544	0.8708	0.8684

Table 7.2: Evolution of the acceptance ratios of the pointwise and the random walk sampler in the setting: $n = 100$, $N = 1000$, and $\log(w_{ij}) \overset{iid}{\sim} \mathcal{N}(0, \lambda^2)$. The acceptance ratios have been obtained by averaging over 5 runs, where for each run the weights were re-sampled. The starting position has been the identity permutation for each run, i.e. $\rho(1) = 1$, $\rho(2) = 2$, etc.

Sampler	$\lambda = \mathbf{1}$	$\lambda = \mathbf{2}$	$\lambda = \mathbf{3}$	$\lambda = \mathbf{4}$	$\lambda = \mathbf{5}$
Random walk	83.2670	25.5077	26.1798	24.1437	55.5518
Pointwise sampler	1.5926	0.3407	0.4444	0.3520	1.0972
Zanella sampler	1.3632	7.5518	8.4628	77.7086	165.1725
Tabu sampler	2.1678	7.3387	21.7478	67.3799	185.4858

Table 7.3: Evolution of the effective sample size of the Hamming distance relative to the identity permutation, in the setting: $n = 100$, $N = 1000$, and $\log(w_{ij}) \overset{iid}{\sim} \mathcal{N}(0, \lambda^2)$. The acceptance ratios have been obtained by averaging over 5 runs, where for each run the weights were re-sampled. The starting position has been the identity permutation for each run, i.e. $\rho(1) = 1$, $\rho(2) = 2$, etc.

For the simulations the weights are sampled i.i.d. according to $\log(w_{ij}) \sim \mathcal{N}(0, \lambda^2)$. This means that the parameter λ can be used to control the smoothness of the target distribution, where decreasing λ results in an increase of the smoothness of the target. For example, by taking $\lambda = 0$, the pointwise informed proposal sampler collapses to the random walk MH sampler. Furthermore, both the Zanella process and Tabu sampler become a MJP with uniform jump rates to neighbouring states. As a larger value of λ causes the weights of the permutations w_{ij} to exhibit a higher degree of differentiation, this is expected to result in a higher acceptance rate for the pointwise sampler and a lower acceptance rate for the random walk MH sampler. The results in Table 7.2 confirm this. Similarly, by increasing λ the target distribution is also expected to express a higher level of multi-modality. In order to analyze how this affects the performance of the samplers, the effective sample size of the log energy of the target density has been considered in Table 7.3. The obtained values for the discrete time samplers are mostly inconclusive. However, the higher value for the

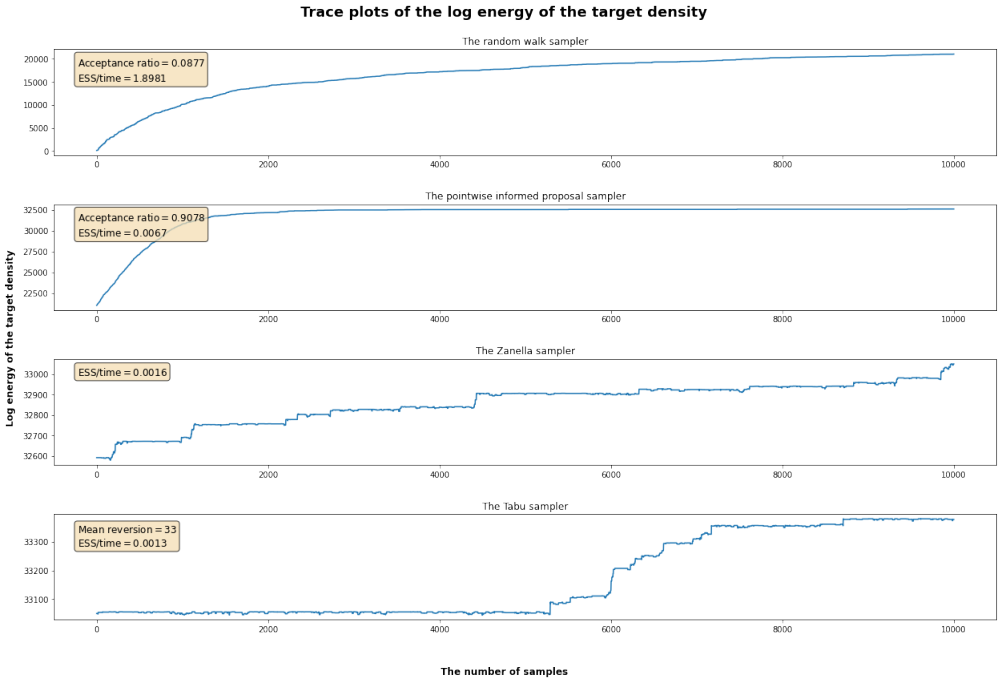

Figure 7.4: Traceplot of the log energy of the target density of the four considered samplers for weighted permutations. The plot has been generated in the setting: $n = 500$, $N = 10000$, and $\log(w_{ij}) \overset{iid}{\sim} \mathcal{N}(0, \lambda^2)$ for $\lambda = 5$.

Sampler	**N = 1000**	**N = 5000**	**N = 10000**
Random walk	36.211	22.5390	15.9962
Pointwise sampler	102.8996	35.4290	12.3601
Zanella sampler	7.5457	14.4057	13.5076
Tabu sampler	22.0393	83.9910	39.9017

Table 7.4: The $\hat{R}$ value of the Gelman-Rubin diagnostic of the four considered samplers for different values of N. The setting: $n = 100$ and $\log(w_{ij}) \overset{iid}{\sim} \mathcal{N}(0, \lambda^2)$ for $\lambda = 5$. For each iterative simulation $m = 10$; 10 different iterative sequences were simulated for comparison.

random walk sampler could suggest that the pointwise sampler is more likely to get stuck in a single mode of the target. Between the Zanella process and Tabu sampler it seems that the Tabu sampler performs better when the distribution becomes more multi-modal, this could be caused by the Tabu sampler being self-avoiding over short to medium timescales. However, also the Zanella process performs significantly better when the "roughness" of the target distribution increases. A possible explanation could be that by also sampling the time at which the moves occur, resulting in a rejection-free sampling algorithm (see algorithm 3), the sampler is able to overcome the fact that moves to lower probability states are typically rejected.

For the traceplots in Figure 7.5 and Figure 7.4 we have set $\lambda = 5$ and $n = 500$. In Figure 7.5, in which a traceplot of the Hamming distance to the highest probability state is displayed, it seems that only the pointwise sampler has reached stationarity. The Zanella process and Tabu sampler are still slowly getting closer to the highest probability state. On the other hand, it seems that the random walk sampler has had too few accepted moves to reach stationarity. However, as this sampler takes the shortest time to compute, it has the highest effective sample size per unit of time. The traceplot of the log energy of the target density π in Figure 7.4 is in line with what the observations made based on the traceplot of the Hamming distance. Despite behaving as a stationary Markov chain, the log energy of the target density for the Zanella process and Tabu sampler are significantly higher than for the pointwise sampler. As both seem to slowly jump to higher and higher probability states, with the Tabu sampler doing so slightly faster. This seems to hint to the fact that the pointwise sampler is getting stuck in a single mode of the target distribution, which could also be an explanation for the especially high acceptance rate.

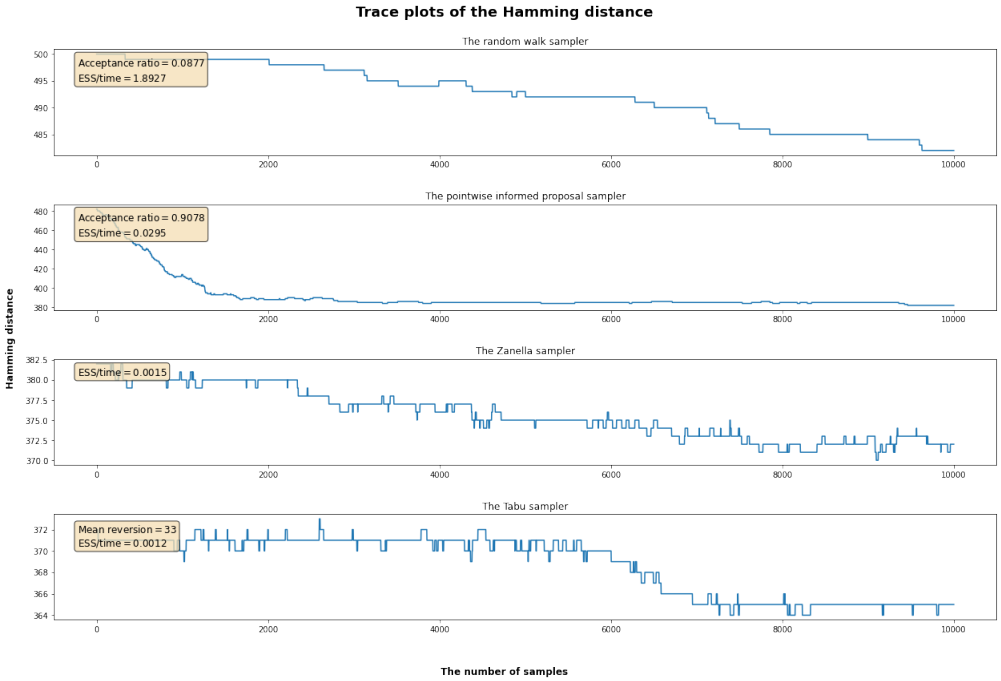

Figure 7.5: Traceplot of the Hamming distance to the highest probability state of the four considered samplers for weighted permutations. The plot has been generated in the setting: $n = 500$, $N = 10000$, and $\log(w_{ij}) \overset{iid}{\sim} \mathcal{N}(0, \lambda^2)$ for $\lambda = 5$

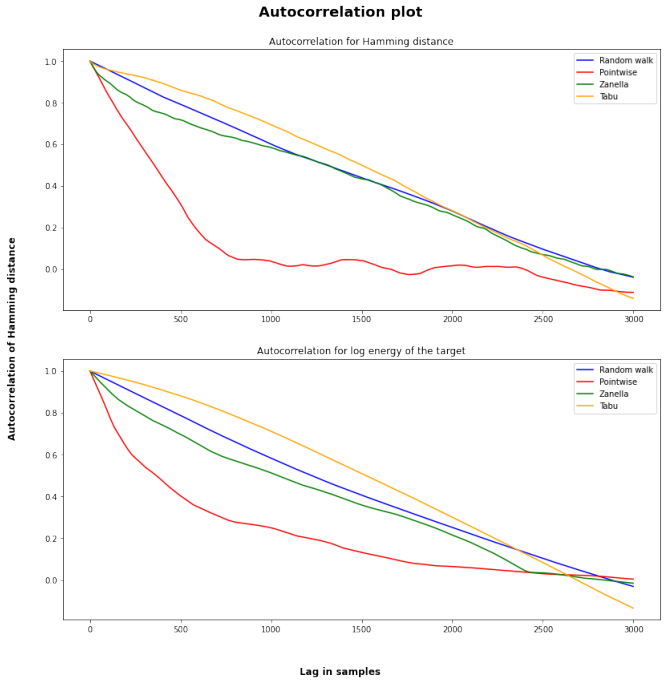

Figure 7.6: Autocorrelation plot for all four samplers for the two considered summary statistics: log energy of the target density and Hamming distance. The autocorrelation has been calculated for the traces as displayed in Figure 7.4 and Figure 7.5.

The Gelman-Rubin diagnostic, for which the results can be found in Table 7.4, strongly confirms the suspicion that the simulation chains are yet to exhibit proper mixing convergence. Although, the Gelman-Rubin diagnostic is not directly interpretable in terms of the cause of the convergence issue, the traceplots in Figure 7.4 and Figure 7.5 suggest that the difficulty to traverse the entire state space is the main candidate cause to consider. Therefore, we have studied the performance of

the four samplers for varying degrees of roughness of the target distribution, from which it can be concluded that the continuous time samplers appear to have a higher level of resilience to especially multi-modal target distributions.

7.5 Bayesian Record Linkage

Record linkage is done to discover which records in a database refer to the same entity. This is useful when merging two databases as it can be used to avoid the occurrence of duplicate records. The use of Bayesian statistics to do this allows for an extra level of interpretability of the results, since the likelihood that two records are matched can be directly observed. For a detailed explanation of the example, the reader is referred to chapter 6. Recall that we intend to merge two databases $\mathbf{x}$ and $\mathbf{y}$ with respective sizes N_1 and N_2. We assume that duplicates only occur between the databases and not within. The posterior to which we will apply the four considered MCMC schemes takes the following form:

$$P(\mathbf{M} \mid \mathbf{x}, \mathbf{y}, \lambda, p_{\text{match}}) \propto \prod_{\{i:\mathbf{m}_i>0\}} \left(\frac{4p_{\text{match}}}{\lambda(1-p_{\text{match}})^2} \prod_{s=1}^{l} \left(\beta(2-\beta) + \frac{(1-\beta)^2}{\theta_{sx_{is}}} \mathbb{1}(x_{is} = y_{\mathbf{m}_i s}) \right) \right). \tag{7.4}$$

Contrary to the proposed Gibbs sampling scheme in section 6.3, we will suppose λ and p_{match} to be known. In the simulations these will be set equal to their ground truth values. When sampling the databases used for the simulations, we will set $\lambda = 50$ and generate $p_{\text{match}} \sim \text{Unif}([0, 1])$. The parameter λ has been set to 50 to restrict the computational complexity for sampling from the posterior, p_{match} has still been generated to account for the level of variability at which duplicates occur in databases that need to be merged. The main reason for keeping λ and p_{match} fixed is to reduce the computational overhead, since sampling λ and p_{match} from their posteriors at each iteration would not allow for a smart update of the rates $\pi(y)/\pi(x)$ as a different value for λ and p_{match} would cause each rate to change. Furthermore, as the main aim of the simulations is to compare the performance of the different MCMC samplers on discrete spaces, the detriment of this simplification to the results is rather limited. For the same reason, the Barker balancing function, $g(t) = t/(1+t)$, has been used throughout.

Sampler	$\beta = \mathbf{0.01}$	$\beta = \mathbf{0.05}$	$\beta = \mathbf{0.10}$	$\beta = \mathbf{0.20}$	$\beta = \mathbf{0.30}$
Random walk	0.2359	0.3093	0.6482	0.2613	0.5529
Pointwise sampler	47.8985	104.3751	86.3000	3.8337	1.1877
Zanella sampler	7.3942	135.3167	12.2272	0.4706	0.4020
Tabu sampler	282.6679	210.7233	3.7774	0.3639	0.4165

Table 7.5: Comparing the effective sample size of the Hamming distance to the golden truth matching $\mathbf{M}$ for different values of β. This was considered for all four samplers, in the setting: $\lambda = 50$ and $N = 2000$. For each value of λ the results are an average over 5 runs, where the databases $\mathbf{x}$ and $\mathbf{y}$ have been regenerated for each run.

Sampler	$\mathbf{N = 100}$	$\mathbf{N = 500}$	$\mathbf{N = 1000}$
Random walk	2.2083	3.4042	3.2635
Pointwise sampler	1.4769	1.4860	1.1625
Zanella sampler	2.2376	1.2449	1.4690
Tabu sampler	1.4130	1.5915	1.3832

Table 7.6: The $\hat{R}$ value of the Gelman-Rubin diagnostic of the four considered samplers for different values of N. For each iterative simulation $m = 10$; 10 different iterative sequences were simulated for comparison. This was done in the setting: $\lambda = 50$, $\beta = 0.3$, and $p_{\text{match}} = 0.95$.

The β parameter is used to describe the noise that occurs between records referring to the same entity; β is the probability that a category of a matched record is sampled as if it were a singleton and not set equal to the golden truth value. Therefore, the β parameter can be used to control the

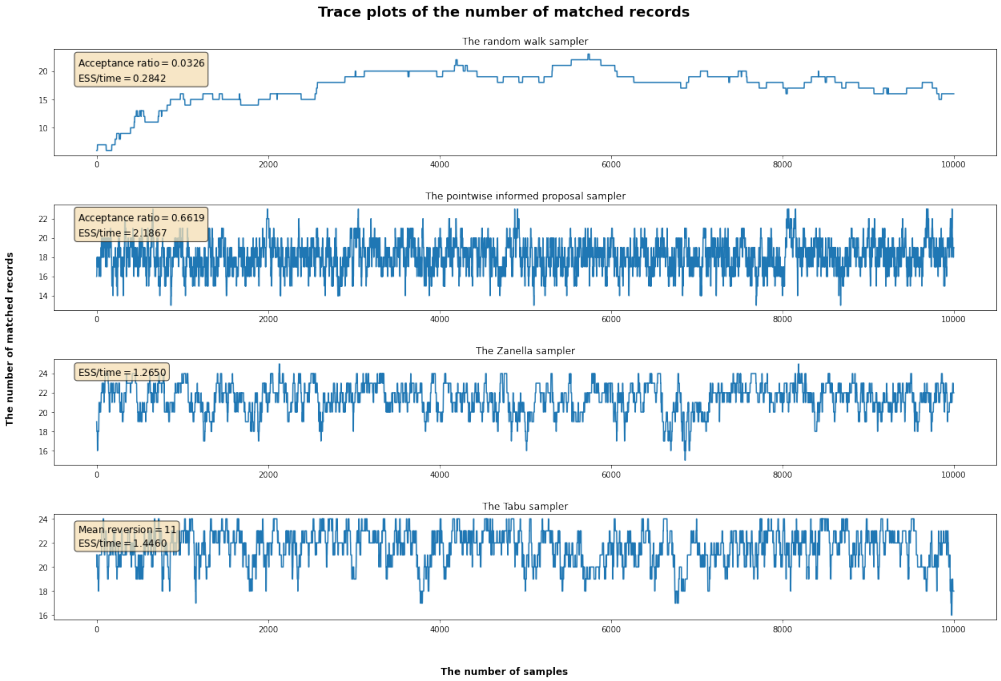

Figure 7.7: Traceplot of the number of matched records of the Bayesian Record Linkage problem for all four considered samplers. The setting: $\beta = 0.3$, $\lambda = 50$, and $p_{\text{match}} = 0.50$.

degree of multi-modality and smoothness of the target distribution. For $\beta = 0$ the target becomes extremely rough with a single mode, as two records can be only be matched together with non-zero probability if they are exactly equal, i.e. the same category in each field. On the other hand, for $\beta = 0$ the distribution becomes uniform as the level of similarity between two records carries no information about whether the records refer to the same entity. The effect of different values of β for the effective sample sizes of the summary statistic being the Hamming distance to the golden truth matching can be found in Table 7.5. In line with the observations from the previous example, the Tabu sampler seems to perform the best when the target is especially rough, i.e. for $\beta = 0.01$. The Zanella process seems to be better suited for situations where the target is slightly smoother having an effective sample size peak for $\beta = 0.05$. The performance of the pointwise informed proposal sampler seems to be similar, albeit being less sensitive to an increasing level of smoothness. Overall, it is clear that incorporation of local information about the target has the best return on computational investment when the target is more peaked at its modes. For the rest of the simulations we set $\beta = 0.30$, since this seems to lead to effective sample sizes for all the considered samplers being somewhat similar. This has been chosen as it is deemed most suitable to compare the qualitative performance of the samplers.

The traceplots in Figure 7.7, Figure 7.8, and Figure 7.9 suggest that all samplers reach stationarity relatively quickly. The autocorrelation plot in Figure 7.10 confirms this observation, where it also becomes apparent that the random walk sampler takes the longest time to converge. This also results in the lowest effective sample size per unit of runtime for each considered summary statistic. Despite being the quickest sampler to run, a low acceptance rate of proposed states leads to the worst performance overall. The traceplot for the pointwise informed proposal looks the best, with the traceplots for the Zanella process and Tabu sampler being nearly identical. A similar conclusion can be drawn based on the effective sample sizes. For the Zanella process and Tabu sampler this high level of similarity is not surprising with the mean reversion, average number of iterations before the global move indicator τ is flipped (see algorithm 4), being merely 11. In addition, this suggest that the target is relatively smooth. Moreover, in the Hamming distance traceplot in Figure 7.8 it can be observed that all samplers are relatively close to the golden truth matching, with the discrete time samplers being slightly closer compared to the Zanella process and Tabu sampler.

The consistency of the estimates for the matches probabilities in Figure 7.1 reinforces the previous observations that the pointwise sampler performs the best, then the Zanella process and

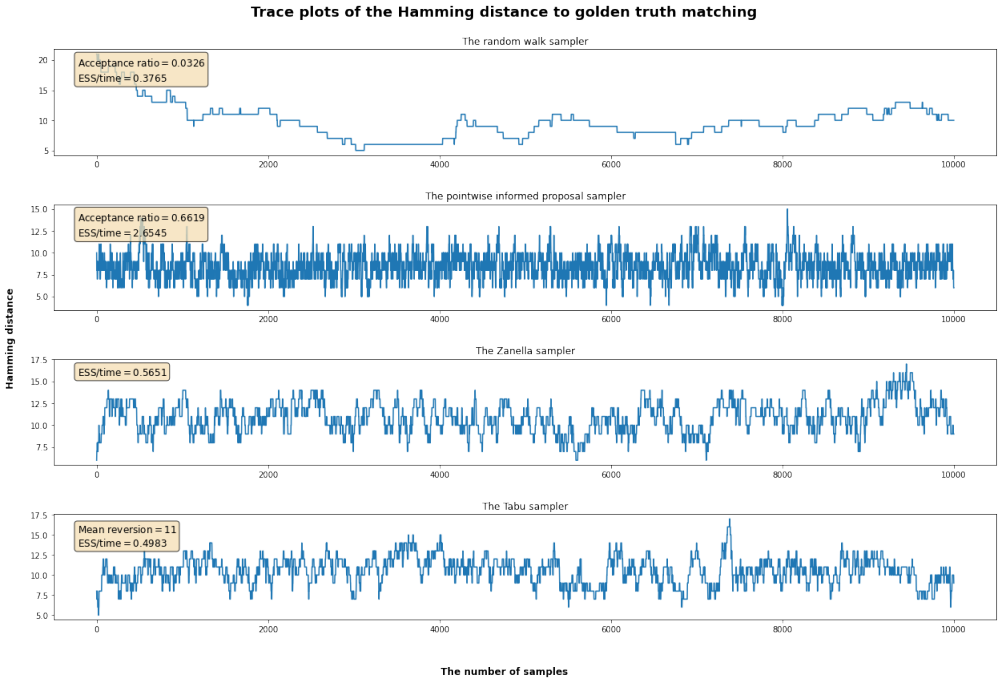

Figure 7.8: Traceplot of the Hamming distance to the golden truth matching $\mathbf{M}$ for all four considered samplers. The setting: $\beta = 0.3$, $\lambda = 50$, and $p_{\text{match}} = 0.50$.

Tabu sampler with comparable results, and lastly, quite a bit worse, the random walk MH sampler. Ideally, all points should be on the blue line as this indicates no variation between subsequent matching estimates. However, the random walk sampler results have significant spread. Whereas, the estimates are quite close to the blue line for the pointwise sampler. The Zanella process and Tabu sampler estimates have marginally more spread, but seem to be positioned about the blue line. The Gelman-Rubin diagnostic presents a similar ordering in the performances of the samplers, for $N = 2000$ the pointwise informed proposal sampler is nearly below the threshold of $\hat{R} < 1.1$. This is important since a $\hat{R}$ value below 1.1 is typically interpreted as an indication of proper convergence; no convergence issues are detected. The $\hat{R}$ values for the Zanella process and Tabu sampler are not monotone decreasing with respect to the N values. This could be due to the fact that especially for $N = 100$, the sample size could be too small to conclude on convergence. However, due to the computational resources needed for the Gelman-Rubin diagnostic, larger values of N were infeasible. On the other hand, as the $\hat{R}$ values are not far off from 1.1 it is not unreasonable to assume that for $N = 10000$, as is the case for the traceplots, the Zanella process and Tabu sampler have reached stationarity. The $\hat{R}$ values for the MH random walk sampler are in line with the slow convergence as observed in the other plots.

The performance of the four samplers is as expected when it comes to sampling from the posterior for Bayesian record linkage. The pointwise sampler performs the best for $\beta = 0.30$, where it should be remarked that this results in quite a smooth posterior distribution. The continuous time MCMC sampling schemes, Zanella and Tabu, are not significantly worse. The fact that the effective sample sizes are less could be explained by the fact that for the continuous time samplers also the clock of the process need to be sampled, which results in more computations needed to sample. In case the posterior has a high level of "roughness", as can also be seen in Table 7.5, this process time sampling seems to be an advantage in adding some extra freedom in generating samples. However, as the distribution becomes smoother, the returns on doing these extra computations decrease. In the extreme case, for a totally uniform distribution on the state space the random walk sampler will certainly be most efficient as incorporating local information has no added benefit.

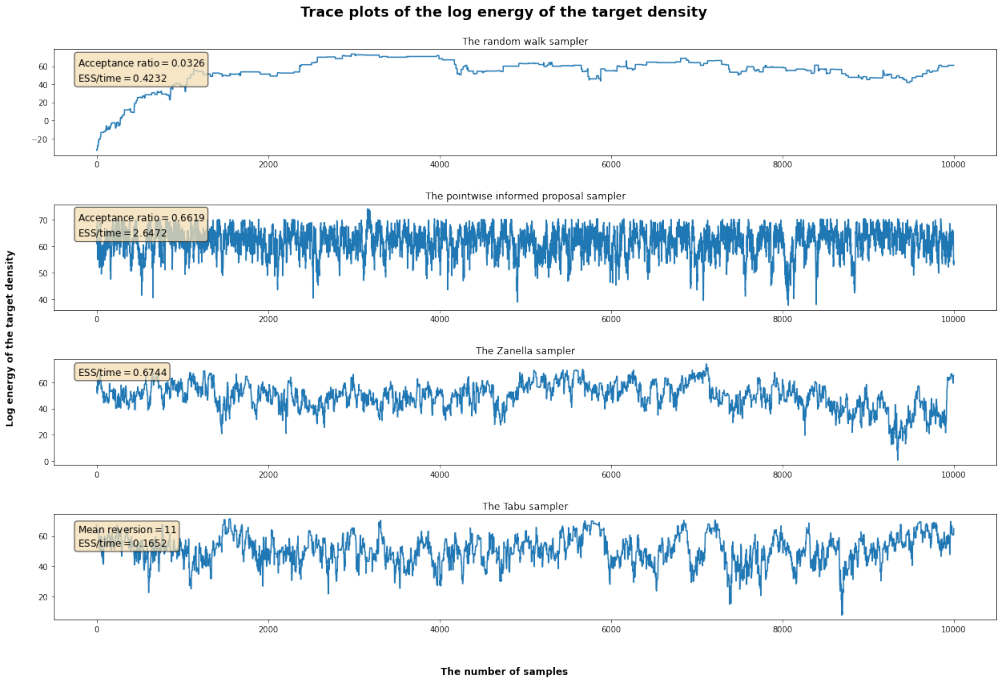

Figure 7.9: Traceplot of the log energy of the target density for all four considered samplers. The setting: $\beta = 0.3$, $\lambda = 50$, and $p_{\text{match}} = 0.50$.

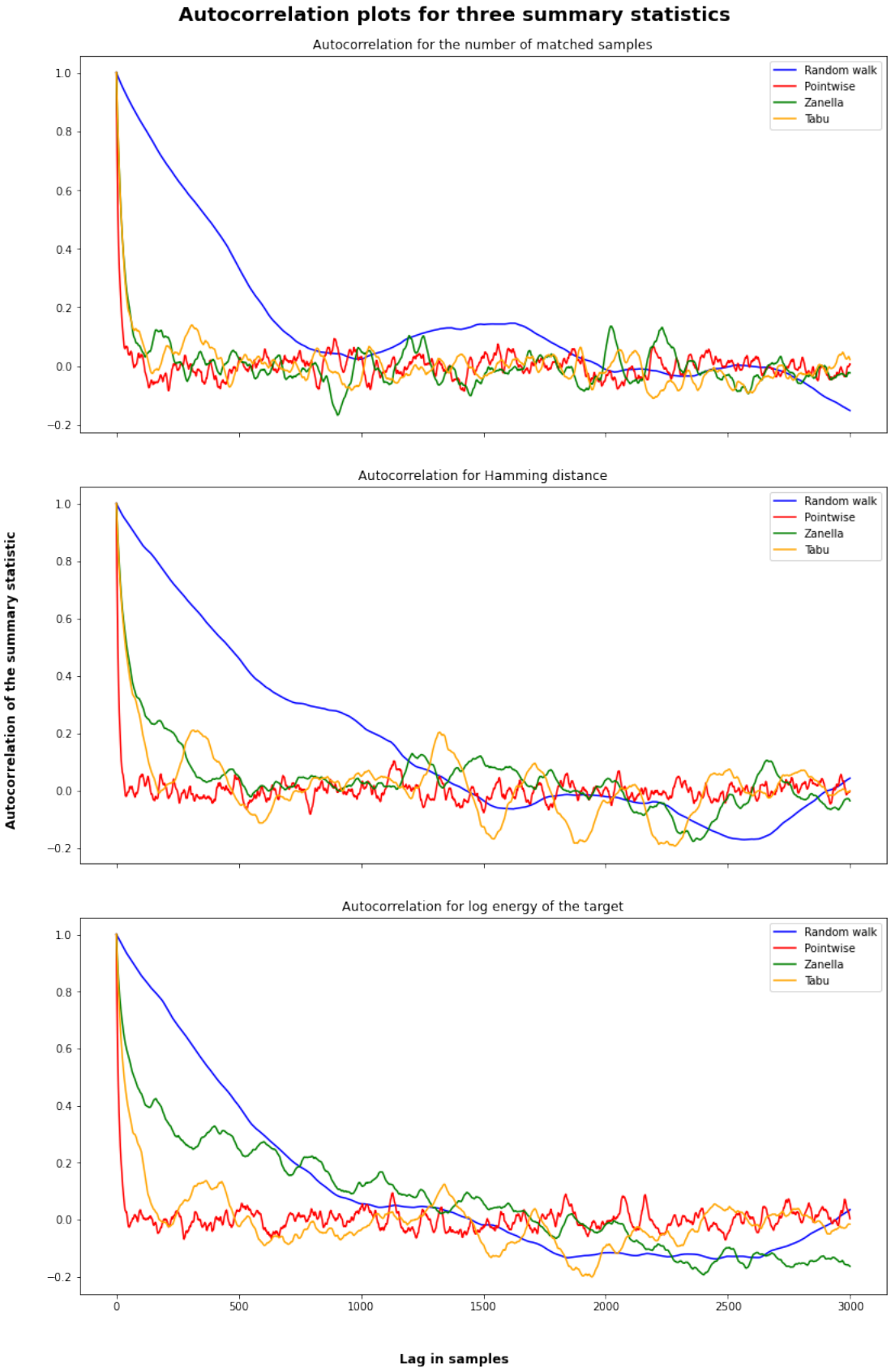

Figure 7.10: Autocorrelation plot for all four samplers for the three considered summary statistics: number of matched records, log energy of the target density, and Hamming distance. The autocorrelation has been calculated for the traces as displayed in Figure 7.7, Figure 7.8, and Figure 7.9.

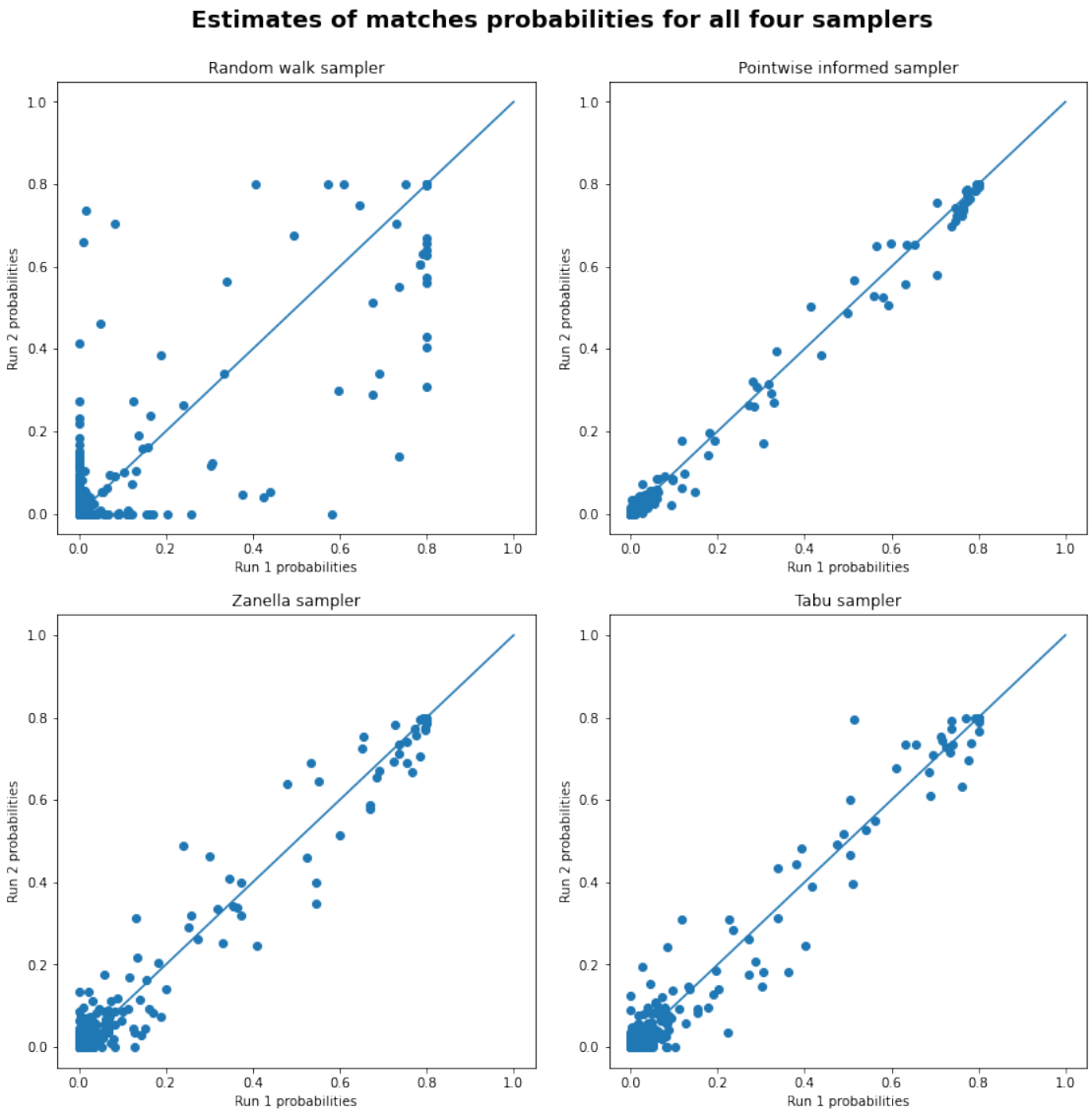

Figure 7.11: The estimated matching probabilities for each pair of records for two runs of all the four considered samplers. This has been done in the setting: $\lambda = 50$, $\beta = 0.3$, and $p_{\text{match}} = 0.53$. For both runs the same set of generated databases $\mathbf{x}$ and $\mathbf{y}$ was used. The starting state was random and resampled for the second iteration.

Chapter 8

Discussion and future work

In this work we compared the MCMC sampling schemes on discrete state spaces as presented in [Zan17] and [PG19]. These two papers both used balancing functions, see theorem 3.1. The discrete-time Markov chains used the balancing functions to created pointwise informed Metropolis-Hastings proposals [Zan17]. The continuous-time samplers used these functions to calculate the jumping rates for the simulated Markov jumping process. The theory on these samplers was introduced in chapter 3 and chapter 4.

Performance of the MCMC sampling schemes

The two discrete time samplers considered were the random walk MH and the pointwise informed MH. The two continuous time samplers were the Zanella process and the Tabu sampler. For a performance-wise comparison three examples were used, most notably a variant of the Bayesian record linkage problem was considered. Based on the simulations it can be envisaged that the returns for using the continuous time samplers, in comparison to the discrete time samplers, increase with the "roughness" of the target distribution; where roughness refers to the difference between the high and low probability states and the number of modes of the target, i.e. how peaked the distribution is around its modes and how many modes there are. This could cause the MCMC scheme to get stuck in a single mode of the target, however, the rejection free continuous time algorithms are more encouraging toward the mixing of the simulated chain. Part of this could be explained by the fact that for the continuous time samplers a move to a low probability state only needs to be proposed, on the other hand, if such a move was proposed in the framework of MH, the proposed step also needs to accepted in the accept-reject step. Especially the Tabu sampler with its non-reversibility on short to medium timescales can be a useful remedy to getting stuck in a subset of the state space. On the other hand, the pointwise informed proposal (MH) sampler is more prone to this problem. Thus, the continuous time samplers seem to be more robust in exploring a highly irregular posterior distribution. In particular, the Tabu sampler can be useful in making inferences from a posterior with a high level of multi-modality. Although, there are not yet theoretical results that can solidify this conjectured efficiency of the Tabu sampler for highly irregular target distributions.

Theoretical outlook

The theory on choosing the optimal balancing function is far from complete. Currently, the main theoretical result is on the balancing functions being Peskun optimal among the functions that can be used to incorporate local information into the proposal kernel for Metropolis-Hastings sampling, see theorem 3.3. Based on the discussion in [PG19], a heuristic choosing procedure for the balancing function in the Zanella process was proposed. The procedure, aimed at making a reasonable trade-off between similarity of the invariant jump measure to the target distribution, and the encouragement of mixing behaviour. More work on the validity of the heuristic, considered metric, and overdispersed distribution can provide new insights into the characteristics of a successful balancing function. Furthermore, the application of more advanced machine learning techniques on the considered data could catalyze new ideas and stimulate theoretical advancements.

Implementation practicalities

The simulated datasets used for the Bayesian record linkage problem were relatively small, with the number of total unique records in the databases $\mathbf{x}$ and $\mathbf{y}$ following a Poisson(50) distribution. The main reason for doing this is that the number of possible pairings between records in the $\mathbf{x}$ and $\mathbf{y}$ databases scales with a factor $N_1 \cdot N_2$, where N_1 and N_2 refer to the number of records in the $\mathbf{x}$ and $\mathbf{y}$ databases respectively. Hence, calculating the rates for each possible move consumes considerable computational resources. A way to reduce this effect, which works if the term $\pi(y)/\pi(y)$ factorizes conveniently as is often the case, is by only smartly updating the move probabilities $g(\pi(y)/\pi(x))$ after a move has been performed.

Another possible solution could be the consideration of a block-wise implementation as suggested in [Zan17]. This would mean that a smaller sub-neighbourhood of the current state $N'(x) \subseteq N(x)$ for $x \in \mathcal{X}$ is selected. For the continuous time samplers, the exact same approach can be implemented. Interesting for this approach is also to consider the trade-off between computational and statistical efficiency. Smaller neighbourhoods $N'(x)$ require less moves for which $g(\pi(y)/\pi(x))$ has to be calculated. However, as less moves are considered at each iteration, the statistical efficiency of the generated samples decreases. It would also be interesting to compare the effects of a block-wise implementation to the discrete and continuous time samplers.

Another reason for the implemented samplers being relatively slow is that all the implementations were done in Python 3. Despite using as much build-in functions as possible, especially from the NumPy module. Implementation in a faster, more advanced, programming language like C or $FORTRAN$ would significantly reduce the time needed to complete the simulations. A minor step in this direction has been taken with the use of the Numba module, which compiles Python code into significantly faster machine code. However, even this module can be relatively tricky to apply to more complex operations. As writing bug-free code was already quite a challenge, and took more rewriting than a priori expected, it was outside of the scope of this project unfortunately to also master usage of the Numba module.

Bibliography

[CH90] J. B. Copas and F. J. Hilton. Record linkage: Statistical models for matching computer records. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 153(3):287–320, 1990.

[GR92] Andrew Gelman and Donald B. Rubin. Inference from iterative simulation using multiple sequences. *Statistical Science*, 7(4):457–472, 1992.

[GWW86] G. Grimmett, D.J.A. Welsh, and D. Welsh. *Probability: An Introduction*. Oxford University Press. Clarendon Press, 1986.

[JP04] J. Jacod and P. Protter. *Probability Essentials*. Universitext. Springer Berlin Heidelberg, 2004.

[KS+60] John G Kemeny, James Laurie Snell, et al. *Finite markov chains*, volume 356. van Nostrand Princeton, NJ, 1960.

[Lam18] B. Lambert. *A Student's Guide to Bayesian Statistics*. SAGE Publications, 2018.

[Nob21] Fabio Nobile. Lecture notes: Stochastic simulation, 2021.

[Pen22] Roger D. Peng. *Advanced Statistical Computing*. 2022.

[Pes73] P. H. Peskun. Optimum monte-carlo sampling using markov chains. *Biometrika*, 60(3):607–612, 1973.

[PG19] Samuel Power and Jacob Vorstrup Goldman. Accelerated sampling on discrete spaces with non-reversible markov processes, 2019.

[RC04] C.P. Robert and G. Casella. *Monte Carlo statistical methods*. Springer Verlag, 2004.

[Rip87] Brian D. Ripley. *Stochastic Simulation*. Wiley, 1987.

[Ste14] Rebecca C. Steorts. Entity resolution with empirically motivated priors, 2014.

[Zan17] Giacomo Zanella. Informed proposals for local mcmc in discrete spaces, 2017.