Fully-Asynchronous Distributed Metropolis Sampler with Optimal Speedup

Weiming Feng * Thomas P. Hayes † Yitong Yin *

Abstract

The Metropolis-Hastings algorithm is a fundamental Markov chain Monte Carlo (MCMC) method for sampling and inference. With the advent of Big Data, distributed and parallel variants of MCMC methods are attracting increased attention. In this paper, we give a distributed algorithm that can correctly simulate sequential single-site Metropolis chains without any bias in a fully asynchronous message-passing model. Furthermore, if a natural Lipschitz condition is satisfied by the Metropolis filters, our algorithm can simulate $N$-step Metropolis chains within $O(N/n + \log n)$ rounds of asynchronous communications, where $n$ is the number of variables. For sequential single-site dynamics, whose mixing requires $\Omega(n \log n)$ steps, this achieves an optimal linear speedup. For several well-studied important graphical models, including proper graph coloring, hardcore model, and Ising model, the condition for linear speedup is weaker than the respective uniqueness (mixing) conditions.

The novel idea in our algorithm is to resolve updates in advance: the local Metropolis filters can be executed correctly before the full information about neighboring spins is available. This achieves optimal parallelism of Metropolis processes without introducing any bias.

1 Introduction

Sampling from joint distributions represented by graphical models is one of the central topics in machine learning. The boom in Big Data applications in contemporary Machine Learning has been drawing increased attention to distributed and algorithms for sampling. For instance, see [19, 6, 25, 27, 11, 1, 4, 5, 3, 15].

The Metropolis-Hastings method is a fundamental Markov chain Monte Carlo method for sampling. Consider a joint distribution $\mu$ for a set $V$ of $n$ random variables, each with domain $[q]$. Algorithm 1 is a Metropolis sampler with single-site updates for $\mu$.

Algorithm 1: single-site Metropolis sampler for $\mu$

Input: initial configuration $X_0 \in [q]^V$

1 for $t = 1$ to $T$ do
2    pick a $v \in V$ uniformly at random;
3    sample a random $x \in [q]$ and let $X' \in [q]^V$ obtained from modifying $X_t(v)$ to $x$;
4    with probability $\min\left\{1, \frac{\mu(X')}{\mu(X_t)}\right\}$, set $X_{t+1} \leftarrow X'$; otherwise, set $X_{t+1} \leftarrow X_t$;

*State Key Laboratory for Novel Software Technology, Nanjing University. Emails: fengwm@smail.nju.edu.cn, yinyt@nju.edu.cn. Supported by the National Key R&D Program of China 2018YFB1003202 and the National Science Foundation of China under Grant Nos. 61722207 and 61672275.

†Department of Computer Science, University of New Mexico. Email:hayes@cs.unm.edu. Partially supported by NSF CAREER award CCF-1150281.
In particular, when $\mu$ is represented by a graphical model, we have

\[
\frac{\mu(X')}{\mu(X_t)} = \frac{\mu_v(X'(v) \mid X_t(N_v))}{\mu_v(X_t(v) \mid X_t(N_v))}
\]

where $\mu_v(\cdot \mid X_t(N_v))$ stands for the marginal distribution at $v$ conditioning on the current configuration of the neighborhood $N_v \subseteq V$ of $v$ in the graphical model. Therefore, each update only needs access to the immediate neighbors of the variable being updated.

The traditional single-site Metropolis sampler is a sequential algorithm. In fact, the algorithm is a sequential discretization of the continuous-time parallel process, in which each variable holds an i.i.d. Poisson clock, so that upon each ringing of a clock at a variable, that variable is updated instantly as in Algorithm 1. Physicists invented this continuous-time process to study natural evolutions and dynamics [10], before computer scientists discretized it to a sequential sampling algorithm.

A major issue for discretizing this originally concurrent continuous-time process as concurrent algorithms, is that updates may no longer be implemented atomically, so that concurrent accesses to critical regions consisting of adjacent variables may cause race conditions that result in faulty sampling. For example, considering uniform sampling proper graph colorings, when two adjacent nodes concurrently try to update their colors based on their knowledges about each other’s current colors, there is a non-negligible chance that the new coloring becomes improper.

The race condition can be naturally avoided by not allowing adjacent variables to be updated concurrently; see [11, 7]. However, this only led to a suboptimal $O(n/\Delta)$ factor of parallel speedup. The issue was partly resolved by: (1) the HogWild! approach [24, 4, 3] which can be fast and approximately accurate assuming stochastically asynchronous schedulers with independently random message delays, but fails in general against adversarial schedulers; (2) the LocalMetropolis chain [7, 8], a well-synchronized parallel Markov chain that can draw correct samples with linear speedup.

**Our result.** We give a fully-asynchronous distributed Metropolis sampler that outputs correct samples, assuming adversarially asynchronous schedulers. Under a Lipschitz condition for the Metropolis filters (formally stated later as Condition 2.1), the distributed sampler achieves an optimal linear speedup for $\Omega(n \log n)$-step sequential Metropolis samplers, where $\Omega(n \log n)$ is also a general lower bound for the mixing time of single-site sequential dynamics [13]. For several well-studied graphical models, e.g., proper graph coloring, hardcore model, and Ising model, this condition for optimal speedup is weaker than the uniqueness (mixing) conditions of the respective models.

Our distributed algorithm actually provides a perfect simulation of the continuous-time process in a model with adversarially asynchronous communications. To have linear speedup, we must allow adjacent variables updated concurrently, yet still perfectly simulate the original process without introducing any bias. To achieve these seemingly contradictory goals, a crucial idea is to *resolve updates in advance*: we show that a Metropolis update can be resolved correctly before the information of the neighborhood is fully available.

More profoundly, our result shows that the continuous-time Metropolis chain, as an idealized concurrent process arising from nature, can be simulated perfectly in artificial systems without overhead on parallelism.
2 Models and Statement of Main Results

2.1 The Metropolis chain

We consider single-site dynamics defined by local Metropolis filters. Let \( G = (V, E) \) be an undirected graph with \( n \) nodes. For each node \( v \in V \), let \( N_v = N(v) \triangleq \{ u \in V \mid (u, v) \in E \} \) denote the neighborhood of \( v \). Let \( q \geq 2 \) and \( [q] = \{1, 2, \ldots, q\} \) be a finite set. Consider the following abstract Markov chain with state space \( \Omega = [q]^V \). At each step \( t \geq 0 \), the state of the chain is a configuration \( X_t \in [q]^V \), and each transition is given below.

**Transition of the single-site Metropolis chain** \( X_t \to X_{t+1} \):

- Pick a \( v \in V \) uniformly at random and denote \( c = X_t(v) \);
- propose a random value \( c' \in [q] \) according to distribution \( \nu_v \) over \([q]\), and let configuration \( X' \in [q]^V \) be constructed as \( X'(v) = c' \) and \( X' = X_t \) elsewhere;
- with probability \( f_{c,c'}^v(X_t(N_v)) \) set \( X_{t+1} \leftarrow X' \), otherwise set \( X_{t+1} \leftarrow X_t \).

The proposal \( c' \in [q] \) is drawn from a distribution \( \nu_v \) over \([q]\), called the *proposal distribution* of \( v \). Given any current \( c = X_t(v) \) and \( c' \in [q] \), the Metropolis filter at \( v \):

\[
f_{c,c'}^v : [q]^{N_v} \to [0, 1]
\]

maps the current configuration of the neighborhood \( X_t(N_v) \) to an acceptance probability. The transition rule of the chain \( X_t \) is fully specified by \((\nu_v)_{v \in V}\) and \((f_{c,c'}^v)_{c,c' \in [q], v \in V}\).

For example, for proper \( q \)-colorings of graph \( G \), domain \([q]\) represents the set of \( q \) colors. The proposal distribution \( \nu_v \) is the uniform distribution over \([q]\) and the Metropolis filter \( f_{c,c'}^v(X_t(N_v)) = \prod_{u \in N_v} I[X_t(u) \neq c'] \) indicates whether \( c' \) properly colors node \( v \) given the current coloring \( X_t(N_v) \) of the neighborhood \( N_v \). The resulting chain, the well-known Metropolis chain for proper \( q \)-coloring, has the uniform distribution over all proper \( q \)-colorings of \( G \) as its stationary distribution.

2.2 Graphical models

More generally, consider any joint distribution \( \mu \) over \([q]^V \) arising from a graphical model on \( G = (V, E) \). Let \( \mu_v \) denote the marginal distribution at node \( v \). An important property of graphical models is the following *conditional independence* property:

\[
\forall \tau \in [q]^{V \setminus \{v\}}, \quad \mu_v \left( \cdot \mid X_{V \setminus \{v\}} = \tau \right) = \mu_v \left( \cdot \mid X_{N(v)} = \tau_{N(v)} \right).
\]

At each node \( v \in V \), the proposal distribution \( \nu_v \) can be any positive distribution over \([q]\), and the Metropolis filter \( f_{c,c'}^v : [q]^{N_v} \to [0, 1] \) is defined as following:

\[
\forall \tau \in [q]^{N_v}, \quad f_{c,c'}^v(\tau) = \min \left\{ 1, \frac{\nu_v(c) \cdot \mu_v(c' \mid \tau)}{\nu_v(c') \cdot \mu_v(c \mid \tau)} \right\}.
\] (1)

To have \( f_{c,c'}^v \) defined everywhere, we may extend the definition of marginal probabilities to conditions \( \tau \) with 0 measure such as assuming \( \mu_v(\cdot \mid \tau) = 0 \) and adopt the convention \( 0/0 = 1 \). Such extension does not affect the definition of the chain over legal states.

It is well known (see, e.g., [17]) that the resulting single-site Metropolis chain \( X_t \) with positive proposal distributions \((\nu_v)_{v \in V}\) and the Metropolis filters given in (1) has stationary distribution \( \mu \). We will refer to such a chain as a *Metropolis chain* for the graphical model \( \mu \) on \( G \).
2.3 Model of asynchrony

We consider distributed algorithms in the fully-asynchronous message-passing model [18, 20, 2]. The network is $G = (V, E)$ itself. Each node represents a processor. Each undirected edge represents two unidirectional channels (one in each direction).

For the synchronous case, communications take place in synchronized rounds. In each round, each node may send messages to and receive messages from all its neighbors, and perform arbitrary local computation atomically.

In the asynchronous model, local computations are still atomic, but messages may travel through channels at arbitrary speeds, whose scheduling is determined by an adversary adaptive to the entire input, with the constraint that for each unidirectional channel, all messages will eventually be delivered in the order in which they are sent by processor (a.k.a. the reliable FIFO channel). Given an execution of an asynchronous algorithm, a time unit or an asynchronous round is defined as the maximum time that elapses between the moment that a message is sent by a processor and the moment that the message is processed by the receiver. The time complexity is measured by the number of time units from the start of the algorithm to its termination in the worst case. The synchronous model is a special case of the asynchronous model with a benign synchronous scheduler, while the HogWild! algorithms [24, 4, 3] gave an intermediate special case of an asynchronous model in which the scheduler is stochastic, creating delays that are independent random variables. For more formal discussions, see [18].

2.4 Main results

We study distributed algorithms that simulate $(X_t)_{t\geq 0}$ on network $G = (V, E)$. A distributed algorithm is said to simulate a Metropolis chain $(X_t)_{t\geq 0}$ if initially, each node $v \in V$ receives as input its initial value $X_0(v)$, the number of nodes $n$, a threshold $T \in \mathbb{R}$, and also the descriptions of proposal distribution $\nu_v$ and Metropolis filters $(f^v_{c,c'})_{c,c' \in [q]}$ if they are not already given implicitly, and upon termination, each node $v \in V$ outputs a $Y(v) \in [q]$, such that altogether the random vector $Y$ is identically distributed as $X_N$ for some sufficiently large integer $N \geq Tn$.

We define a Lipschitz condition for the Metropolis filters. Let $f : [q]^d \to \mathbb{R}$ be a $d$-variate total function. Given any $1 \leq i \leq d$, $a, b \in [q]$, we define the operator $\delta_{i,a,b} f$ on function $f$ as:

$$\delta_{i,a,b} f \triangleq \max_{x,y} |f(x) - f(y)|,$$

where the maximum is taken over all pairs of $(x, y)$ satisfying $x_i = a$, $y_i = b$ and $x_j = y_j$ for all $j \neq i$.

Let $(X_t)_{t\geq 0}$ be a single-site Metropolis chain with proposal distributions $(\nu_v)_{v \in V}$ and Metropolis filters $(f^v_{c,c'})_{v \in V, c,c' \in [q]}$. Recall that each $f^v_{c,c'}$ is a function $f^v_{c,c'} : [q]^{N_v} \to [0, 1]$.

**Condition 2.1.** There is a constant $C > 0$ such that for any $(u, v) \in E$, any $a, b, c \in [q]$, it holds that

$$\mathbb{E}_{v \sim \nu_v} [\delta_{u,a,b} f^v_{c,c'}] \leq \frac{C}{\Delta},$$

where $\Delta = \Delta(G)$ denotes the maximum degree of $G$.

We show that any single-site Metropolis chain satisfying Condition 2.1 can be simulated efficiently (with a linear speedup) by a distributed algorithm, even against adversarial asynchronous schedulers.
Theorem 1 (main theorem). Assume that Condition 2.1 holds for the single-site Metropolis chain \((X_t)_{t \geq 0}\). There is a fully-asynchronous distributed algorithm that simulates \((X_t)_{t \geq 0}\) such that given any \(T \in \mathbb{R}_{\geq 0}\), with high probability the algorithm outputs \(X_N \in [q]^V\) for some \(N \geq Tn\) within \(O(T + \log n)\) time units, with each message of size \(O(\log n + \log q)\).

The notions of asynchronous distributed algorithm and distributed simulation of Metropolis chains used in above theorem are as defined earlier. The \(O(\cdot)\) hides (linearly) the constant \(C\) in Condition 2.1.

The algorithm in Theorem 1 simulates \(\Omega(n \log n)\)-step sequential Metropolis chains distributedly with linear speedup. On various well-studied graphical models, this gives efficient distributed simulations of respective Metropolis chains under following conditions:

- **Proper \(q\)-coloring:** The natural Metropolis chain for proper \(q\)-coloring is as follows. For each \(v \in V\), \(\nu_v\) is the uniform over \([q]\) and
  \[\forall c, c' \in [q], \tau \in [q]^N_v : f_{c,c'}^v(\tau) = \prod_{u \in N_v} I[\tau_u \neq c].\]

  Over proper \(q\)-colorings, this chain behaves identically as the chain described in (1). Condition 2.1 applied on this chain translates to: there is a constant \(\alpha > 0\) such that
  \[q \geq \alpha \Delta.\]

In contrast, the uniqueness condition is \(q \geq \Delta + 1\) [14, 9].

- **Hardcore model:** The distribution \(\mu\) is over all configurations in \(\{0,1\}^V\) that corresponds to independent sets of \(G\). For each configuration \(\sigma \in \{0,1\}^V\) that indicate an independent set of \(G\), \(\mu(\sigma) \propto \lambda^{\sum_v \sigma(v)}\), where \(\lambda \geq 0\) is the fugacity. The natural Metropolis chain for this model is: For each \(v \in V\), \(\nu_v\) is a distribution over \([0,1]\) with \(\nu_v(0) = \frac{1}{1+\lambda}\) and \(\nu_v(1) = \frac{\lambda}{1+\lambda}\), and
  \[\forall c, c' \in \{0,1\}, \tau \in \{0,1\}^{N_v} : f_{c,c'}^v(\tau) = \prod_{u \in N_v} I[\tau_u + c' \leq 1].\]

Condition 2.1 applied on this chain translates to: there is a constant \(C > 0\) such that
  \[
  \lambda < \frac{C}{\Delta},
  \]

while the uniqueness condition is \(\lambda < \frac{(\Delta-1)\Delta-1}{(\Delta-2)^2} \approx \frac{\Delta}{2}\) [26, 22].

- **Ising model:** The distribution \(\mu\) is over all configurations in \(\{-1,+1\}^V\), such that for each \(\sigma \in \{-1,+1\}^V\), \(\mu(\sigma) \propto \exp \left(\beta \sum_{(u,v) \in E} \sigma_u \sigma_v\right)\), where \(\beta \in \mathbb{R}\) is the temperature. The natural Metropolis chain for this model is: For each \(v \in V\), \(\nu_v\) is uniform over \(\{-1,1\}\), and
  \[\forall c, c' \in \{-1,1\}, \tau \in \{-1,1\}^{N_v} : f_{c,c'}^v(\tau) = \exp \left(\min \left\{0, \beta(c' - c) \sum_{u \in N_v} \tau_u\right\}\right).\]

Condition 2.1 applied on this chain translates to: there is a constant \(C > 0\) such that
  \[1 - e^{-2|\beta|} < \frac{C}{\Delta},\]

while the uniqueness condition is \(1 - e^{-2|\beta|} < \frac{2}{\Delta}\) [21, 23].

For above graphical models, Condition 2.1 is much weaker than the uniqueness (mixing) conditions, because our goal is to simulate the sequential chain regardless of its mixing.
that simulates the single-site for such approaches since no adjacent nodes are updated concurrently.

\[ \nu \]

Proposal distribution \( v \) each node \( \text{lis chain} \left( \text{Y} \right) \).

**Definition 3.1.** We consider distributed algorithms that perfectly simulate the continuous-time Metropolis chain.

### 3.2 Distributed simulation of continuous-time chain

Let \( (Y_t)_{t \in \mathbb{R}_{\geq 0}} \) denote the continuous-time version of the single-site Metropolis chain \( (X_t)_{t \geq 0} \) defined in Section 2.1. Formally, \( (Y_t)_{t \in \mathbb{R}_{\geq 0}} \) has state space \( \Omega = \{q\} \); initially \( Y_0 = X_0 \); and as time \( t \) elapses continuously \( Y_t \) evolves by the following rules:

- Each node \( v \in V \) is associated with an i.i.d. rate-1 Poisson clock. Recall that a rate-\( k \) Poisson clock is defined as: assuming \( \{x_i\}_{i=1}^{\infty} \) to be i.i.d. exponential random variables with \( \mathbb{E}[x_i] = 1/k \), the clock rings at time \( t_1, t_2, t_3 \ldots \), where \( t_i = \sum_{j=1}^{i} x_j \).

- When the Poisson clock at node \( v \) rings, the value at \( v \) is updated in the same manner as in the discrete-time Metropolis chain. More specifically, supposed the Poisson clock at node \( v \) ring at time \( t \in \mathbb{R}_{\geq 0} \) and \( c = Y_{t-c}(v) \), node \( v \) draws \( c' \in \{q\} \) according to its proposal distribution \( \nu \), and updates \( Y_t(v) \) to the proposed value \( c' \) with probability \( f_{c,c'}^v(Y_t(N_v)) \).

The continuous-time dynamics have been extensively studied (see \([16, 17]\)). In fact, the now more popular discrete-time chains originated as a discretization of the continuous-time dynamics, which model natural evolutions and dynamics.

It is well-known that the single-site dynamics \( (X_t)_{t \geq 0} \) and its continuous-time version \( (Y_t)_{t \in \mathbb{R}_{\geq 0}} \) have the following equivalence.

**Proposition 2.** For any \( T \geq 0 \), \( Y_T \) is identically distributed as \( X_N \) where \( N \sim \text{Pois}(nT) \).

We can then focus on distributed algorithms that simulate the continuous-time Metropolis chain \( (Y_t)_{t \in \mathbb{R}_{\geq 0}} \). It is obvious to see that the continuous-time chain is a parallel process. However, naively implement the process as a distributed algorithm without proper concurrency control may cause race conditions that give faulty sampling results, because adjacent nodes may update their values concurrently using each other’s outdated information due to delay of communications.

### 3.2 Distributed simulation of continuous-time chain

We consider distributed algorithms that perfectly simulate the continuous-time Metropolis chain.

**Definition 3.1.** A distributed algorithm is said to perfectly simulate a continuous-time Metropolis chain \( (Y_t)_{t \in \mathbb{R}_{\geq 0}} \) if the followings hold. Let \( T \in \mathbb{R}_{\geq 0} \) be an arbitrary length of time. Initially each node \( v \in V \) receives as input the initial value \( Y_0(v) \) and \( T \), along with the descriptions of proposal distribution \( \nu \) and Metropolis filter \( \{f_{c,c'}^v\}_{c,c' \in \{q\}} \). Upon termination of the algorithm, each node \( v \in V \) outputs a \( Y(v) \in \{q\} \), altogether \( Y \) is identically distributed as \( Y_T \).

A straightforward approach for perfectly simulating \( (Y_t)_{t \in \mathbb{R}_{\geq 0}} \) by distributed algorithms is to resolve each update only if all prior adjacent updates are known to be resolved. Such straightforward approach works generically for continuous-time single-site dynamics (not just the Metropolis chain). In particular, it perfectly simulates \( (Y_t)_{t \in \mathbb{R}_{\geq 0}} \), such that given any \( T \geq 0 \), with high probability the algorithm terminates within \( O(\Delta T + \log n) \) time units, where \( \Delta = \Delta(G) \) denotes the maximum degree of graph \( G \). The \( \Delta \)-factor overhead seems necessary for such approaches since no adjacent nodes are updated concurrently.
3.3 Resolve updates in advance

There is actually a smarter way to resolve the updates by distributed algorithms, so that updates at adjacent nodes can be resolved concurrently, yet no race condition is ever reached and the original chain \((Y_t)_{t \in \mathbb{R}_{\geq 0}}\) is perfectly simulated without any bias.

**Theorem 3.** Let \((Y_t)_{t \in \mathbb{R}_{\geq 0}}\) be the continuous-time version of a single-site Metropolis chain \((X_t)_{t \geq 0}\). There is a fully-asynchronous distributed algorithm that perfectly simulates \((Y_t)_{t \in \mathbb{R}_{\geq 0}}\). Moreover, if Condition 2.1 holds for \((X_t)_{t \geq 0}\), then given any \(T \geq 0\), with high probability the algorithm terminates and output \(Y_T\) within \(O(T + \log n)\) time units, with each message of size \(O(\log n + \log q)\).

For the convenience of exposition, we present the algorithm in such a way that every node \(v \in V\) first locally generates all its Poisson random update times and random proposals. Specifically, each node \(v\) locally generates all the moments at which the Poisson clock at \(v\) rings before time \(T\), which gives the moments of \(v\)'s value being updated in the chain \((Y_t)_{t \in \mathbb{R}_{\geq 0}}\). We call them the *update times* of \(v\). Besides, corresponding to each update time, \(v\) also samples an independent proposed value from \(\nu_v\). Such initialization is done locally at each node before all communications.

We further assume that each node \(v \in V\) also knows the initial value \(Y_0(u)\), the random update times and random proposals of all its neighbors \(u \in N_v\). This can be achieved by a preprocessing step in which these local informations are exchanged between neighbors. Assuming each message contain a single pair of update time and proposed value, which costs \(O(\log n + \log q)\) bits with high probability\(^1\), this preprocessing step terminates within \(O(T + \log n)\) time units with high probability due to the concentration of Poisson random variable. The algorithm consists of two phases: **Phase I** for the preprocessing and **Phase II** for the actual simulation of continuous-time chain \((Y_t)_{t \in \mathbb{R}_{\geq 0}}\), outlined in following. We also remark that each node needs not to wait for others except for itself to terminates in **Phase I** before entering its own **Phase II**.

**Algorithm at node** \(v \in V\):

**Locally do:** Simulate rate-1 Poisson clock for time duration \(T\) to generate a sequence of random update times for \(v\); for each update time, sample a random proposal independently from \(\nu_v\); let \(m_v\) denote the number of pairs of update times and proposals generated.

**Phase I:** Exchange the initial value, update times and proposals with all neighbors.

**Phase II:** For \(i = 1\) to \(m_v\) do:

- Keep listening to the channels from all neighbors.
- \((\ast)\) As soon as \(v\) gets **enough information**:
  - Resolve the \(i\)-th update of \(v\) and send the update result ("ACCEPT" or "REJECT") to all neighbors.

The condition in Line \((\ast)\) plays a vital part in the algorithm. Consider the \(i\)-th iteration of the For-loop in Phase II of the algorithm, during which the algorithm has resolved \(v\)'s first \((i - 1)\) updates and is trying to resolve its \(i\)-th update. Say \(t\) being the \(i\)-th update time and \(c'\) being the \(i\)-th random proposal, both at node \(v\). Let \(c\) denote the current value of node \(v\), i.e. \(c = Y_{t- \epsilon}(v)\). To successfully resolve the \(i\)-th update at \(v\), one needs to carry out a random experiment to pass a Metropolis filter, which involves evaluating function \(f^v_{c, c'}(Y_{t}(N_v))\).

\(^1\)The update times can be represented with bounded precision using \(O(\log n)\) bits each, which is enough for the simulation of chain \((Y_t)_{t \in \mathbb{R}_{\geq 0}}\) because with high probability all update times are distinct.
Here both $c$ and $c'$ are known to $v$. Obviously $v$ can resolve the update once it also knows the correct $Y_t(N_v)$.

Line (⋆) is much more sophisticated: node $v$ can resolve an update in advance, much earlier than everything about $Y_t(N_v)$ is known. Abstractly, our algorithm achieves this by coupling the coins $f^u_{c,c'}(Y_t(N_v))$ for all potential $Y_t(N_v)$, based on the partial information collected so far by $v$ about $Y_t(N_v)$. Such coupling gives the lower bounds for the probability masses of ACCEPT and REJECT, which are used to resolve updates in advance.

For example, consider the proper graph coloring. Here $c$ is the current color of node $v$ and $c'$ is the randomly proposed color for the update time $t$. The function $f^u_{c,c'}(Y_t(N_v))$ is given as $f^u_{c,c'}(Y_t(N_v)) = \prod_{a \in N_v} I[Y_t(u) \neq c]$. Node $v$ can resolve the update as long as it has enough information about $Y_t(N_v)$ to guarantee one of the following two conditions hold:

- $\forall u \in N_v, Y_t(u) \neq c$, in which case $f^u_{c,c'}(Y_t(N_v)) = 1$ and $c'$ must be accepted;
- $\exists u \in N_v, Y_t(u) = c$, in which case $f^u_{c,c'}(Y_t(N_v)) = 0$ and $c'$ must be rejected.

These conditions can be checked before everything about $Y_t(N_v)$ is known.

For general Metropolis chains, this is generalized to a coupling of all accept/reject distributions specified by all possible $Y_t(N_v)$ consistent with the partial information known to $v$. The next section is dedicated to the rigorous exposition of this idea.

4 Advanced Resolution of Metropolis Filters

We now formally describe the main algorithm outlined in last section, which consists of two phases. The description of Phase I is clear in last section. We focus on Phase II.

Fix a node $v \in V$. Suppose that $v$ updates $m_v$ times before time $T$ in the chain $(Y_t)_{t \in \mathbb{R}_{\geq 0}}$. We define the following notations.

- We use $0 < t(v,1) < t(v,2) < \ldots < t(v,m_v) < T$ to denote the update times generated by a rate-1 Poisson clock. By convention, we set $t(v,0) = 0$ and $t(v,m_v + 1) = T$.
- We use $c(v,1), c(v,2), \ldots, c(v,m_v) \in [q]$ to denote the corresponding proposals sampled i.i.d. from distribution $\nu_v$.

In Phase I of the algorithm, these informations are generated locally at each node $v \in V$, and are exchanged between neighbors along with the initial values $Y_0(v)$. Once a node collected the initial values $Y_0(u)$, the update times $t(u,1) < \ldots < t(u,m_u) < T$ and the proposals $c(u,1), \ldots, c(u,m_u) \in [q]$, of all neighbors $u \in N_v$, it enters its Phase II, whose pseudocode (Algorithm 2) is given below.\footnote{Different nodes may enter Phase II asynchronously due to the delays in Phase I. Observe that no message is ever sent from a Phase-I node in to a Phase-II node. We assume that the messages sent from a Phase-II node $u$ to a Phase-I node $v$ are stored in a queue at $v$ and processed by $v$ in the same order they are sent once $v$ enters its own Phase II.}

The algorithm at node $v$ resolves all the updates of $v$ one by one in the order they are issued. In the $i$-th step, node $v$ has resolved the first $i - 1$ updates and is currently trying to resolving the $i$-th update. The algorithm effectively generates a continuous-time chain $(Y_t)_{t \in \mathbb{R}_{\geq 0}}$, such that for any time $0 \leq t \leq T$ and any node $v \in V$,

$$Y_t(v) = Y_{t(v,i)}(v) \text{ where } i \text{ satisfies } t(v,i) \leq t < t(v,i + 1).$$

Our goal is to guarantee that this continuous-time chain is identically distributed as the continuous-time Metropolis chain the algorithm wants to simulate.

Node $v$ also maintains the following quantities during the execution of Algorithm 2:
Algorithm 2: Pseudocode for Phase II of the main algorithm at node v

**Assumption:** node v ∈ V knows the initial values Y₀(u) and the lists of update times and proposals \((t(u, i), c(u, i))\)₁≤𝑖≤𝑚ᵣ of all u ∈ Nᵥ ∪ \{v\}.

1. Initialize \(j = (j_u)_{u ∈ N(v)}, \) and \(Y = (Y(u, j))_{u ∈ N(v), 0 ≤ j ≤ j_u - 1}\) respectively as:
2. for all u ∈ Nᵥ: \(j_u ← 1\) and \(Y(u, 0) ← Y_0(u);\)
3. for \(i = 1\) to \(m_q\) do
4. \((Y_{i(v, i)}(v), j, Y) ← Resolve(i, Y_{i(v, i-1)}(v), j, Y) \triangleright \text{resolve the } i\text{-th update}\)
5. return \(Y_{i(v, m_q)}(v);\)

- a vector \(j \triangleq (j_u)_{u ∈ N(v)},\) such that each \(j_u\) tells that from v’s perspective, the neighbor u is resolving its \(j_u\)-th update, and the results of its first \((j_u - 1)\) updates are known;
- a table \(Y = (Y(u, j))_{u ∈ N(v), 0 ≤ j ≤ j_u - 1},\) such that each \(Y(u, j)\) memorizes the value of \(Y_{t(u, j)}(u).\)

The subroutine \(\text{Resolve}(i, Y(v), j, Y)\) needs to satisfy the following invariant:

\[
\forall u ∈ Nᵥ, \forall 0 ≤ k ≤ j_u - 1:\ Y(u, k) = Y_{t(u,k)}(u).
\] (3)

It means that for each neighbor \(u ∈ Nᵥ,\) node v knows that \(u\) has already resolved all its first \(j_u - 1\) updates and \(Y(u, .\)) stores all historical values of \(u\) before time \(t(u, j_u - 1)\).

The following identity is implied by the invariant (3):

\[
\forall 0 ≤ t < t(u, j_u) : \ Y_t(u) = Y_{t(u,k-1)}(u) = Y(u, k - 1) \text{ where } t(u, k - 1) ≤ t < t(u, k).
\] (4)

Therefore, node v can infer \(Y_t(u)\) for any neighbor \(u ∈ Nᵥ\) at any time \(t\) before \(t(u, j_u)\).

For later time \(t ∈ [t(u, j_u), T]\), \(v\) can no longer infer the precise value of \(Y_t(u)\) but rather a set of possible values of \(Y_t(u)\).

**Definition 4.1** (set of possible values). Fix a node \(v ∈ V\). For any neighbor \(u ∈ Nᵥ\) and any time \(0 ≤ t < T\), define the set of possible values \(S_t(u) ⊆ [q]\) for node \(u\) at time \(t\) with respect to \(j = (j_u)_{u ∈ N(v)}\) as

\[
S_t(u) \triangleq \begin{cases} 
\{Y_t(u)\} & \text{if } 0 ≤ t < t(u, j_u) \\
\{Y_{t(u,j_u-1)}(u)\} \cup \{c(u, k) \mid t(u, j_u) ≤ t(u, k) ≤ t\} & \text{if } t(u, j_u) ≤ t < T.
\end{cases}
\] (5)

It is easy to verify by the transition rule of Metropolis chain that \(Y_t(u) ∈ S_t(u)\) always holds for any \(u ∈ Nᵥ\) and \(0 ≤ t < T\). This guarantees the soundness of the definition. Furthermore, given any \((j, Y)\) satisfying the invariant (3), node v can locally compute sets \(S_t(u)\) for all \(u ∈ Nᵥ\) and \(0 ≤ t < T\) by (4) and (5).

The set \(S_t(u)\) captures the partial information about \(Y_t(u)\) for the neighbors \(u ∈ Nᵥ\). Next, we show how to use such information to resolve an update determined by the Metropolis filter \(f_{c,v}^p(Y_t(Nᵥ))\) before everything about \(Y_t(Nᵥ)\) is known.

### 4.1 Example: proper graph coloring

We first give the subroutine \(\text{Resolve}\) on a special case: the uniform proper \(q\)-coloring of graph \(G\). To be distinguished from the general case, we use \(\text{Resolve-Coloring}(i, Y(v), j, Y)\) to denote the subroutine for this special case. The pseudocode is given in Algorithm 3.

The goal of Algorithm 3 is to resolve the \(i\)-th update \((t(v, i), c(v, i))\) for proper coloring. Node \(v\) maintains for each neighbor \(u ∈ Nᵥ\) a set \(S_{t(v, i)}(u)\) of possible colors of \(u\) at time \(t(v, i)\).
Algorithm 3: \(\text{Resolve-Coloring}(i, Y(v), j, \mathcal{Y})\) at node \(v\)

\textbf{input:} index \(i\) of the current update; current color \(Y(v)\) of \(v\); vector \(j\) and table \(\mathcal{Y}\) to store the neighbors’ current steps and historical colors from \(v\)’s perspective;

1. construct \(S_{t(v,i)}(u)\) as (5) for all \(u \in N_v\);
2. \textbf{upon} \(c(v, i) \notin \bigcup_{u \in N_v} S_{t(v,i)}(u)\) do
3. \hspace{1em} send message “ACCEPT” to all neighbors \(u \in N_v\);
4. \hspace{1em} return \(c(v,i), j, \mathcal{Y}\);
5. \textbf{upon} \(\exists u \in N_v\) s.t. \(S_{t(v,i)}(u) = \{c(v, i)\}\) do
6. \hspace{1em} send message “REJECT” to all neighbors \(u \in N_v\);
7. \hspace{1em} return \((Y(v), j, \mathcal{Y})\);
8. \textbf{upon} receiving “ACCEPT” from a neighbor \(u \in N_v\) do
9. \hspace{2em} \(\mathcal{Y}(u, j_u) \leftarrow c(u, j_u)\);
10. \hspace{2em} \(j_u \leftarrow j_u + 1\);
11. \hspace{2em} recompute \(S_{t(v,i)}(u)\) as (5);
12. \textbf{upon} receiving “REJECT” from a neighbor \(u \in N_v\) do
13. \hspace{2em} \(\mathcal{Y}(u, j_u) \leftarrow \mathcal{Y}(u, j_u - 1)\);
14. \hspace{2em} \(j_u \leftarrow j_u + 1\);
15. \hspace{2em} recompute \(S_{t(v,i)}(u)\) as (5);

based on information collected so far by \(v\). The algorithm is event-driven. The set \(S_{t(v,i)}(u)\) of each neighbor \(u \in N_v\) is up to updates once a message “ACCEPT” or “REJECT” is heard from \(u\). Node \(v\) is also constantly monitoring the sets \(S_{t(v,i)}(u)\) for all \(u \in N_v\). The update \((t(v, i), c(v, i))\) is finally resolved once one of the following two events occurs:

- \(c(v, i) \notin \bigcup_{u \in N_v} S_{t(v,i)}(u)\), in which case the proposed color \(c(v, i)\) does not conflict with all possible colors of any neighbors at time \(t(v, i)\), thus \(c(v, i)\) must be accepted;
- \(\exists u \in N_v\) s.t. \(S_{t(v,i)}(u) = \{c(v, i)\}\), in which case the proposed color \(c(v, i)\) is blocked by neighbor \(u\)’s color at time \(t(v, i)\), thus \(c(v, i)\) must be rejected.

These two events are mutually exclusive, so that they cannot occur simultaneously. We will also see that at least one of them must occur eventually so that the algorithm must terminate. This is formally proved for the general \text{Resolve} algorithm where proper graph coloring is a special case.

4.2 General Metropolis chains

We then give the subroutine \text{Resolve}(i, Y(v), j, \mathcal{Y})\) for general Metropolis chain. The goal of the subroutine is to resolve the \(i\)-th update \((t(v, i), c(v, i))\). To do so, the algorithm needs to flip a coin according to the Metropolis filter \(f_{c,c'}^{v} (Y(t(v,i))(N_v))\) to determine whether the proposed color \(c(v, i)\) is accepted. Here both the current value \(c = Y(v) = Y_{t(v,i-1)}(v)\) and the proposed value \(c' = c(v, i)\) of \(v\) are known. Only the configuration \(Y_{t(v,i)}(N_v)\) is partially known by the sets \(S_{t(v,i)}(u)\) of possible values of neighbors \(u \in N_v\) at time \(t(v, i)\). We further define the set of all possible configurations on the neighborhood \(N_v\) as

\[C_{t(v,i)} \triangleq \bigotimes_{u \in N_v} S_{t(v,i)}(u),\]

where \(S_{t(v,i)}(u)\) are as constructed in Definition 4.1. \(C_{t(v,i)}\) must contains the correct configuration \(Y_{t(v,i)}(N_v)\) because \(Y_{t(v,i)}(u) \in S_{t(v,i)}(u)\) for all individual \(u \in N_v\). Besides, \(C_{t(v,i)}\) may
contains various other candidate configurations $\tau \in [q]^V$, each gives a coin with bias $f_{c,c}(\tau)$. The subroutine resolves the update in advance by coupling all these coins optimally.

Specifically, for each $\tau \in \mathcal{C}_{l(v,i)}$, define the indicator random variable $I_{\mathcal{AC}}(\tau) \in \{0,1\}$ as $\Pr[I_{\mathcal{AC}}(\tau) = 1] = f_{Y(v),c(v,i)}(\tau)$. The coins $I_{\mathcal{AC}}(\tau)$ for all $\tau \in \mathcal{C}_{l(v,i)}$ are coupled as follows: Let $\beta \in [0,1)$ be uniformly distributed.

$$I_{\mathcal{AC}}(\tau) = \begin{cases} 1 & \text{if } \beta < f_{Y(v),c(v,i)}(\tau), \\ 0 & \text{if } \beta \geq f_{Y(v),c(v,i)}(\tau). \end{cases}$$

Since the true $Y_{l(v,i)}(N_v) \in \mathcal{C}_{l(v,i)}$, the update can be resolved once all the indicator random variables $I_{\mathcal{AC}}(\tau)$ for $\tau \in \mathcal{C}_{l(v,i)}$ are perfectly coupled, i.e. $\forall \tau_1, \tau_2 \in \mathcal{C}_{l(v,i)} : I_{\mathcal{AC}}(\tau_1) = I_{\mathcal{AC}}(\tau_2)$.

To check whether all the indicator random variables are perfectly coupled, define the minimum accept probability $P_{\mathcal{AC}}$ and the minimum reject probability $P_{\mathcal{RE}}$ as:

$$P_{\mathcal{AC}} \triangleq \min_{\tau \in \mathcal{C}_{l(v,i)}} f_{Y(v),c(v,i)}(\tau);$$

$$P_{\mathcal{RE}} \triangleq \min_{\tau \in \mathcal{C}_{l(v,i)}} \left(1 - f_{Y(v),c(v,i)}(\tau)\right) = 1 - \max_{\tau \in \mathcal{C}_{l(v,i)}} f_{Y(v),c(v,i)}(\tau).$$

Then all coins $I_{\mathcal{AC}}(\tau)$ for $\tau \in \mathcal{C}_{l(v,i)}$ are perfectly coupled if $\beta < P_{\mathcal{AC}}$ or $\beta \geq 1 - P_{\mathcal{RE}}$. The former case corresponds to the event that all $I_{\mathcal{AC}}(\tau) = 1$, while the latter corresponds to the event that all $I_{\mathcal{AC}}(\tau) = 0$.

The pseudocode for subroutine Resolve($i,Y(v),j,\mathcal{Y}$) at node $v$ is given in Algorithm 4.

---

**Algorithm 4: Resolve($i,Y(v),j,\mathcal{Y}$) at node $v$**

**input:** index $i$ of the current update; current value $Y(v)$ of $v$; vector $j$ and table $\mathcal{Y}$ to store the neighbors’ current steps and historical values from $v$’s perspective;

1. sample $\beta \in [0,1)$ uniformly at random;
2. construct $S_{l(v,i)}(u)$ as (5) for all $u \in N_v$;
3. compute $P_{\mathcal{AC}}$ and $P_{\mathcal{RE}}$ as (6);
4. **upon** $\beta < P_{\mathcal{AC}}$ **do**
5. send message “ACCEPT” to all neighbors $u \in N_v$;
6. **return** $(c(v,i),j,\mathcal{Y})$;
7. **upon** $\beta \geq 1 - P_{\mathcal{RE}}$ **do**
8. send message “REJECT” to all neighbors $u \in N_v$;
9. **return** $(Y(v),j,\mathcal{Y})$;
10. **upon** receiving “ACCEPT” from a neighbor $u \in N_v$ **do**
11. \[ \mathcal{Y}(u,j_u) \leftarrow c(u,j_u); \]
12. \[ j_u \leftarrow j_u + 1; \]
13. recompute $S_{l(v,i)}(u)$ as (5);
14. recompute $P_{\mathcal{AC}}$ and $P_{\mathcal{RE}}$ as (6);
15. **upon** receiving “REJECT” from a neighbor $u \in N_v$ **do**
16. \[ \mathcal{Y}(u,j_u) \leftarrow \mathcal{Y}(u,j_u - 1); \]
17. \[ j_u \leftarrow j_u + 1; \]
18. recompute $S_{l(v,i)}(u)$ as (5);
19. recompute $P_{\mathcal{AC}}$ and $P_{\mathcal{RE}}$ as (6);

In the algorithm, a random number $\beta \in [0,1)$ is sampled only once in the beginning and used during the entire execution (hence the coupling). The algorithm is event-driven. The
two thresholds $P_{AC}, P_{RE} \in [0, 1]$ are updated dynamically upon receiving message. The update $(t(v, i), c(v, i))$ is accepted once $\beta < P_{AC}$ and is rejected once $\beta \geq 1 - P_{RE}$. These two events are mutually exclusive because $P_{AC} + P_{RE} \leq 1$, so that they cannot occur simultaneously. Furthermore, eventually at least one of them must occur when $|\mathcal{C}_{t(v,i)}| = 1$, i.e. when the correct configuration $Y_{t(v,i)}(N_v)$ is fully known to $v$.

**Remark 4.1.** The subroutine Resolve-Coloring (Algorithm 3) is indeed a special case of the subroutine Resolve (Algorithm 4) by fixing $f_{Y(v),c(v,i)}(\tau) = \prod_{u \in N(v)} I[\tau_u \neq c(v, i)]$, in which case $P_{AC} \in \{0, 1\}$ indicates the event $c(v, i) \notin \bigcup_{u \in N_v} S_{t(v,i)}(u)$ and $P_{RE} \in \{0, 1\}$ indicates the event $\exists u \in N_v$ s.t. $S_{t(v,i)}(u) = \{c(v, i)\}$.

The cost for local computation of the algorithm is dominated by the costs for computing the two thresholds $P_{AC}$ and $P_{RE}$, which are easy to compute for graphical models defined by edge factors, e.g. Markov random fields, including all specific models mentioned in Section 2.4. For such graphical models, the Metropolis filter $f_{c,c'}^v(\tau)$ can be written as:

$$\forall \tau \in [q]^{N(v)} : f_{c,c'}^v(\tau) = \min \left\{ 1, \prod_{u \in N(v)} f_{c,c'}^{v,u}(\tau_u) \right\}, \quad \text{where } f_{c,c'}^{v,u} : [q] \to \mathbb{R}_{\geq 0}.$$

For this broad class of Metropolis filters, the thresholds $P_{AC}$ and $P_{RE}$ can be computed by the closed-forms:

$$P_{AC} = \min \left\{ 1, \prod_{u \in N(v)} \left( \min_{c \in S_{t(v,i)}(u)} f_{Y(v),c(v,i)}^{v,u}(c) \right) \right\},$$

$$P_{RE} = 1 - \min \left\{ 1, \prod_{u \in N(v)} \left( \max_{c \in S_{t(v,i)}(u)} f_{Y(v),c(v,i)}^{v,u}(c) \right) \right\},$$

where the sets $S_{t(v,i)}(u)$ are easy to compute by definition in (5).

## 5 Outline of Proofs

The main theorem (Theorem 1) is implied by Theorem 3, Proposition 2 and the concentration for Poisson distribution (Proposition 4).

The following are concentration inequalities for Poisson distribution [12, Lemma 11].

**Proposition 4.** Let $N \in \mathbb{Z}_{\geq 0}$ be a Poisson random variable with mean $\mu$, the following concentration inequalities hold for any $\epsilon < 1$:

$$\Pr[N \leq (1 - \epsilon)\mu] \leq \exp(-\epsilon^2\mu/2)$$

$$\Pr[N \geq (1 + \epsilon)\mu] \leq \exp(-\epsilon^2\mu/3).$$

Furthermore, if $t \geq 5\mu$, then

$$\Pr[N \geq t] \leq 2^{-t}.$$

The main theorem (Theorem 1) is proved as follows. Let $(Y_t)_{t \in \mathbb{R}_{\geq 0}}$ be the continuous-time version of the single-site Metropolis chain $(X_t)_{t \geq 0}$. Let

$$T' = 2T + 8 \log n = O(T + \log n).$$
By Theorem 3, there is a fully-asynchronous distributed algorithm that perfectly simulates \((Y_t)_{t \geq 0}\). Since Condition 2.1 holds for \((X_t)_{t \geq 0}\), then given \(T' = O(T + \log n)\), with high probability the algorithm terminates and outputs \(Y_{T'}\) within \(O(T + \log n)\) time units.

By Proposition 2, \(Y_{T'}\) is identically distributed as \(X_N\) where \(N \sim \text{Pois}(nT')\). By the concentration for Poisson distribution (Proposition 4), we have

\[
\Pr[N \geq Tn] \geq 1 - \frac{1}{n}.
\]

A union bound proves that with high probability, the algorithm in Theorem 3 outputs \(X_N \in [q]^V\) for some \(N \geq Tn\) within \(O(T + \log n)\) time units.

Hence, the main theorem holds if Theorem 3 is proved. Here we give an outline of the proof of Theorem 3.

**Correctness:** Let \((Y_t^C)_{t \geq 0}\) denote the continuous-time Metropolis chain that the algorithm wants to simulate, and \((Y_t^A)_{t \geq 0}\) the continuous-time chain generated by main algorithm (the \((Y_t)_{t \geq 0}\) in (2)). The algorithm is correct if \((Y_t^A)_{t \geq 0}\) and \((Y_t^C)_{t \geq 0}\) can be coupled perfectly.

Both \((Y_t^A)_{t \geq 0}\) and \((Y_t^C)_{t \geq 0}\) use the following randomness. For each node \(v \in V\), there is an i.i.d. rate-1 Poisson clock at node \(v\) for generating update times. For each update time \(t\) at each node \(v \in V\), there is a random pair \((c, \beta)\) such that \(c \in [q]\) is a proposed value sampled independently from \(\nu_v\) and \(\beta\) is a random real number sampled uniformly and independently \([0, 1]\). For \((Y_t^A)_{t \geq 0}\), \(\beta\) is sampled in Line 1 of Algorithm 4, while for \((Y_t^C)_{t \geq 0}\), \(\beta\) is used implicitly as the random source for passing a Metropolis filter.

We prove that the two chains can be perfectly coupled by using the same randomness.

**Running time:** Let \((Y_t)_{t \geq 0}\) be the continuous-time version of a single-site Metropolis chain \((X_t)_{t \geq 0}\). If Condition 2.1 is satisfied by \((X_t)_{t \geq 0}\), then given any \(T\), we prove that with high probability, the main algorithm outputs \(Y_T\) within \(O(T + \log n)\) time units.

The **Phase I** costs \(O(T + \log n)\) time units with high probability due to the concentration for Poisson distribution. We focus on the analysis of the **Phase II**.

The time complexity of the **Phase II** is analyzed by a *dependency chain* argument. The dependency chain is constructed as follows. Consider the \(i\)-th update of node \(v \in V\). Given an execution of the main algorithm, the \(i\)-th update of \(v\) must be resolved in **Phase II**, and we can find the unique predecessor of the \(i\)-th update of \(v\) as follows:

- If node \(v\) resolves its \(i\)-th update upon receiving the message “ACCEPT” or “REJECT” from a neighbor \(u \in N_v\) that indicates the result of \(u\) resolving the \(j\)-th update of \(u\), then the predecessor is the \(j\)-th update of \(u\).

- If node \(v\) resolves its \(i\)-th update upon computing \(P_{AC}\) and \(P_{RE}\) in Line 3 of Algorithm 4, then the predecessor is the \((i - 1)\)-th update of \(v\) if \(i > 1\) or there is no predecessor if \(i = 1\).

Repeating the above process gives a sequence of updates, with everyone preceding the subsequent update. Such a sequence is called a dependency chain.

The time complexity of the **Phase II** is bounded by the length of longest dependency chain because each dependency takes place within at most one time unit. We show that when Condition 2.1 holds, as a dependency chain becomes longer, the probability it occurs decays exponentially.
References

[1] Amr Ahmed, Moahmed Aly, Joseph Gonzalez, Shravan Narayanamurthy, and Alexander J Smola. Scalable inference in latent variable models. In Proceedings of the 5th ACM International Conference on Web Search and Data Mining (WSDM), pages 123–132. ACM, 2012.

[2] Hagit Attiya and Jennifer Welch. Distributed computing: fundamentals, simulations, and advanced topics, volume 19. John Wiley & Sons, 2004.

[3] Constantinos Daskalakis, Nishanth Dikkala, and Siddhartha Jayanti. Hogwild!-gibbs can be panaccurate. In Proceedings of the 31st Advances in Neural Information Processing Systems (NIPS), pages 32–41, 2018.

[4] Christopher De Sa, Kunle Olukotun, and Christopher Ré. Ensuring rapid mixing and low bias for asynchronous Gibbs sampling. In Proceedings of the 33rd International Conference on Machine Learning (ICML), pages 1567–1576, 2016.

[5] Christopher De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Rapidly mixing Gibbs sampling for a class of factor graphs using hierarchy width. In Proceedings of the 28th Advances in Neural Information Processing Systems (NIPS), pages 3097–3105, 2015.

[6] Finale Doshi-Velez, Shakir Mohamed, Zoubin Ghahramani, and David A Knowles. Large scale nonparametric Bayesian inference: Data parallelisation in the Indian buffet process. In Proceedings of the 22nd Advances in Neural Information Processing Systems (NIPS), pages 1294–1302, 2009.

[7] Weiming Feng, Yuxin Sun, and Yitong Yin. What can be sampled locally? In Proceedings of the 36th ACM Symposium on Principles of Distributed Computing (PODC), pages 121–130, 2017.

[8] Manuela Fischer and Mohsen Ghaffari. A simple parallel and distributed sampling technique: Local glauber dynamics. In Proceedings of the 32nd International Symposium on Distributed Computing (DISC), 2018.

[9] Andreas Galanis, Daniel Štefankovič, and Eric Vigoda. Inapproximability for antiferromagnetic spin systems in the tree nonuniqueness region. Journal of the ACM (JACM), 62(6):50, 2015.

[10] Roy J Glauber. Time-dependent statistics of the ising model. Journal of mathematical physics, 4(2):294–307, 1963.

[11] Joseph E Gonzalez, Yucheng Low, Arthur Gretton, and Carlos Guestrin. Parallel Gibbs sampling: From colored fields to thin junction trees. In Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS), volume 15, pages 324–332, 2011.

[12] Thomas P. Hayes. Local uniformity properties for glauber dynamics on graph colorings. Random Struct. Algorithms, 43(2):139–180, 2013.

[13] Thomas P Hayes and Alistair Sinclair. A general lower bound for mixing of single-site dynamics on graphs. The Annals of Applied Probability, pages 931–952, 2007.

[14] Johan Jonasson. Uniqueness of uniform random colorings of regular trees. Statistics & Probability Letters, 57(3):243–248, 2002.
[15] Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabás Póczos. Parallelised bayesian optimisation via thompson sampling. In Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS), pages 133–142, 2018.

[16] Samuel Karlin. A first course in stochastic processes. Academic press, 2014.

[17] David A Levin and Yuval Peres. Markov chains and mixing times, volume 107. American Mathematical Soc., 2017.

[18] Nancy A Lynch. Distributed algorithms. Elsevier, 1996.

[19] David Newman, Arthur Asuncion, Padhraic Smyth, and Max Welling. Distributed inference for latent Dirichlet allocation. In Proceedings of the 20th International Conference on Neural Information Processing Systems (NIPS), pages 1081–1088, 2007.

[20] David Peleg. Distributed computing: a locality-sensitive approach. SIAM, 2000.

[21] Alistair Sinclair, Piyush Srivastava, and Marc Thurley. Approximation algorithms for two-state anti-ferromagnetic spin systems on bounded degree graphs. Journal of Statistical Physics, 155(4):666–686, 2014.

[22] Allan Sly. Computational transition at the uniqueness threshold. In Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pages 287–296, 2010.

[23] Allan Sly, Nike Sun, et al. Counting in two-spin models on d-regular graphs. The Annals of Probability, 42(6):2383–2416, 2014.

[24] Alexander Smola and Shravan Narayanamurthy. An architecture for parallel topic models. Proceedings of the VLDB Endowment, 3(1-2):703–710, 2010.

[25] Padhraic Smyth, Max Welling, and Arthur U Asuncion. Asynchronous distributed learning of topic models. In Proceedings of the 22nd Advances in Neural Information Processing Systems (NIPS), pages 81–88, 2009.

[26] Dror Weitz. Counting independent sets up to the tree threshold. In Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC), pages 140–149, 2006.

[27] Feng Yan, Ningyi Xu, and Yuan Qi. Parallel inference for latent Dirichlet allocation on graphics processing units. In Proceedings of the 22nd Advances in Neural Information Processing Systems (NIPS), pages 2134–2142, 2009.
Appendix A  Proof of Correctness

Lemma 5. Given any continuous-time Metropolis chain \((Y_t)_{t \in \mathbb{R}_{\geq 0}}\), the main algorithm perfectly simulates \((Y_t)_{t \in \mathbb{R}_{\geq 0}}\) once terminates.

Proof. Let \((Y^C_t)_{t \in \mathbb{R}_{\geq 0}}\) (\(Y^C\) in short) denote the continuous-time Metropolis chain defined in Section 3.1. Let \((Y^A_t)_{t \in \mathbb{R}_{\geq 0}}\) (\(Y^A\) in short) denote the continuous-time chain generated by main algorithm when the algorithm simulates \((Y^C_t)_{t \in \mathbb{R}_{\geq 0}}\). Specifically, given any \(T\) to the main algorithm, \(Y^A_t(v) = Y^C_t(v)\) for any time \(0 \leq t \leq T\) and any node \(v \in V\), where \(Y^C_t(v)\) is defined in (2). Suppose \(Y^A_0 = Y^C_0\). We prove that the main algorithm outputs \(Y^A_T \in [q]^V\) such that \(Y^A_T\) and \(Y^C_T\) are equally distributed.

We actually prove a stronger result: for any \(0 \leq t \leq T\), \(Y^A_t\) and \(Y^C_t\) are equally distributed. This result is proved by constructing an identical coupling between two chains \((Y^A_t)_{t \in \mathbb{R}_{\geq 0}}\) and \((Y^C_t)_{t \in \mathbb{R}_{\geq 0}}\).

Recall the continuous-time Metropolis chain \(Y^C\) is defined as follows. Each node \(v \in V\) is associated with an i.i.d. rate-1 Poisson clock. Suppose the Poisson clock at node \(v \in V\) rings at time \(t \in \mathbb{R}_{\geq 0}\) and \(c = Y^C_t(v)\), node \(v\) draws \(c' \in [q]\) according to its proposal distribution \(\nu_c\), samples a real number \(\beta \in [0,1]\) uniformly at random, then updates the value \(Y^C_t(v)\) as the proposed value \(c'\) if \(\beta < f^C_{c,c'}(Y^C_t(v))\) or keeps its value unchanged if otherwise \(\beta \geq f^C_{c,c'}(Y^C_t(v))\).

The coupling between \(Y^A\) and \(Y^C\) is defined as follows:

- \(Y^A\) and \(Y^C\) use the same Poisson clock at each node \(v \in V\).
- For each node \(v \in V\) and each update time of node \(v\), node \(v\) proposes the same random value \(c'\) from the proposal distribution \(\nu_c\) and samples the same random real number \(\beta\) uniformly from \([0,1]\) to resolve such update, where in main algorithm, \(\beta\) is sampled in Line 1 of Algorithm 4 when \(v\) is trying to resolve such update.

Fix the randomness of all Poisson clocks (before time \(T\)), all proposals and all real numbers. Then we obtain the following sequence

\[
(u_1, t_1, c_1, \beta_1), (u_2, t_2, c_2, \beta_2), \ldots, (u_m, t_m, c_m, \beta_m),
\]

where \(0 < t_1 < t_2 < \ldots < t_m < T\). Each tuple \((u_i, t_i, c_i, \beta_i)\) represents the Poisson clock at \(u_i \in V\) rings at time \(t_i\); node \(u_i\) proposes \(c_i \in [q]\) for such update and samples \(\beta_i \in [0,1]\) to resolve such update. And \(m\) is the total number of rings of all \(n\) Poisson clocks. Given the above sequence, both \((Y^A_t)_{0 \leq t \leq T}\) and \((Y^C_t)_{0 \leq t \leq T}\) are determined.

Remark that given the sequence in (7), the main algorithm must terminate and output \(Y^A_T\) in finite number of time units. Recall in message-passing model, all unidirectional channels are reliable FIFO channels. The Phase I must terminate within \(O(m)\) time units. In Phase II, any update \((t, c)\) of node \(v\) must be resolved if \(v\) knows that its neighbors \(u \in N_v\) have resolved all their updates with update time less than \(t\). Hence, all updates in (7) can be resolved within finite time units.

Given the sequence in (7), we prove that \(Y^C_t = Y^A_t\) for all \(0 \leq t \leq T\). Note that \(Y^A\) and \(Y^C\) change their states only at times \(t_1, t_2, \ldots, t_m\). Assume \(t_0 = 0\). It is sufficient to prove \(Y^A_{t_k} = Y^C_{t_k}\) for all \(0 \leq k \leq m\). We prove the result by induction on \(k\).

The base case \(Y^A_0 = Y^C_0\) is trivial. Fix an integer \(1 \leq k \leq m\). Suppose \(Y^A_{t_i} = Y^C_{t_i}\) for all \(0 \leq i \leq k - 1\). We prove \(Y^A_{t_k} = Y^C_{t_k}\). Since \(Y^A\) and \(Y^C\) change their states only at times \(t_1, t_2, \ldots, t_m\), the induction hypothesis implies

\[
\forall 0 \leq t < t_k : \quad Y^A_t = Y^C_t.
\]
Observe that in both two chains, only node $u_k$ can update its value at time $t_k$. Then for all $v \in V \setminus \{u_k\}$, it holds that $Y_{t_k}^{A-\epsilon}(v) = Y_{t_k}^{A}(v)$ and $Y_{t_k}^{C-\epsilon}(v) = Y_{t_k}^{C}(v)$. By (8), we have
\[ \forall v \neq u_k : \quad Y_{t_k}^{A}(v) = Y_{t_k}^{C}(v). \] (9)
Hence, it is sufficient to prove $Y_{t_k}^{A}(u_k) = Y_{t_k}^{C}(u_k)$.

Consider the moment when node $u_k$ resolves the update $(t_k, c_k)$ in Algorithm 4. Let $j = (j_a)_{a \in N_{u_k}}$ and $\mathcal{Y}$ be the vector and table in Algorithm 4 when $u_k$ resolves the update $(t_k, c_k)$. For each neighbor $u \in N(u_k)$, node $u_k$ computes the set $S_{t_k}(u)$ in Definition 4.1 based on $j$ and $\mathcal{Y}$:
\[
S_{t_k}(u) = \begin{cases} 
\{Y_{t_k}^{A}(u)\} & \text{if } t_k < t(u, j_a) \\
\{Y_{t_k}^{A}(u)\} \cup \{c(u, i) \mid t(u, j_a) \leq t(u, i) \leq t_k\} & \text{if } t_k \geq t(u, j_a).
\end{cases}
\]
Recall $t(u, j)$ and $c(u, j)$ are update time and proposal of the $j$-th update of node $u$, which are uniquely determined by the sequence in (7). Remark that the set $S_{t_k}(u)$ can be correctly computed due to the invariant (3). Such invariant holds because each unidirectional channel is a reliable FIFO channel. We claim that $Y_{t_k}^{C}(u) \in S_{t_k}(u)$. This can be verified in two cases. Case 1: If $t_k < t(u, j_a)$, then $Y_{t_k}^{C}(u) \in S_{t_k}(u)$ holds due to (9). Case 2: If $t_k \geq t(u, j_a)$, then $Y_{t_k}^{A}(u) = Y_{t_k}^{C}(u)$ due to (8), besides, $S_{t_k}(u)$ also contains all proposals of $u$ between time $t(u, j_a)$ and time $t_k$, hence $Y_{t_k}^{C}(u) \in S_{t_k}(u)$ holds due to the transition rule of Metropolis chain. Thus, we have
\[ Y_{t_k}^{C}(N_{u_k}) \in \mathcal{C}(t_k), \quad \text{where } \mathcal{C}(t_k) = \bigotimes_{u \in N_{u_k}} S_{t_k}(u). \]

In $Y^A$, let $c = Y_{t_k}^{A-\epsilon}(u_k)$, node $u_k$ computes $P_{AC}$ and $P_{RE}$ to resolve the update $(t_k, c_k)$:
\[ P_{AC} = \min_{\tau \in \mathcal{C}(t_k)} f_{c, c_k}^{u_k}(\tau), \quad P_{RE} = 1 - \max_{\tau \in \mathcal{C}(t_k)} f_{c, c_k}^{u_k}(\tau). \]

In $Y^C$, note that $Y_{t_k}^{C-\epsilon}(u_k) = Y_{t_k}^{A-\epsilon}(u_k) = c$ due to (8), node $u_k$ computes the acceptance probability to resolve the update $(t_k, c_k)$:
\[ P_{AC}^{M} = f_{c, c_k}^{u_k}(Y_{t_k}^{C}(N_{u_k})). \]
Since $Y_{t_k}^{C}(N_{u_k}) \in \mathcal{C}(t_k)$, then we have
\[ P_{AC}^{M} \geq P_{AC} \]
\[ 1 - P_{AC}^{M} \geq P_{RE}. \]

If the proposal $c_k$ is accepted in $Y^A$, then $\beta_k < P_{AC}$, which implies $\beta_k < P_{AC}^{M}$ and the proposal $c_k$ must be accepted $Y^C$. If the proposal $c_k$ is rejected in $Y^A$, then $\beta_k \geq 1 - P_{RE}$, which implies $\beta_k \geq P_{AC}^{M}$ and the proposal $c_k$ must be rejected in $Y^C$. Note that $Y_{t_k}^{C-\epsilon}(u_k) = Y_{t_k}^{A-\epsilon}(u_k)$. Combining two cases implies $Y_{t_k}^{A}(u_k) = Y_{t_k}^{C}(u_k)$. Combining with (9) implies $Y_{t_k}^{A} = Y_{t_k}^{C}$. \hfill \Box

**Appendix B  Analysis of Running Time**

**Lemma 6.** Let $(Y_t)_{t \in \mathbb{R}^{\geq 0}}$ be the continuous-time version of a single-site Metropolis chain $(X_t)_{t \geq 0}$. If Condition 2.1 holds for $(X_t)_{t \geq 0}$, then given any $T \geq 0$, with high probability the main algorithm outputs $Y_T$ and terminates within $O(T + \log n)$ time units.

17
The Lemma is proved by analyzing the time complexity of the following modified algorithm, which is an upper bound of the time complexity of our main algorithm.

- Let all nodes start the Phase I at the moment $T_0$. Let $T_1$ denote the moment when the last node completes the Phase I. For any node $v \in V$, if $v$ completes the Phase I before moment $T_1$, node $v$ becomes idle and does not enter the Phase II.

- All nodes start the Phase II synchronously at moment $T_1$. Let $T_2$ denote the moment when the last node completes the Phase II.

Let $R_1$ denote the number of time units between $T_0$ and $T_1$. Let $R_2$ denote the number of time units between $T_1$ and $T_2$. The time complexity of the main algorithm is at most $R_1 + R_2$, because some nodes may enter the Phase II before $T_1$ in the main algorithm.

For each node $v \in V$, let $R_1^v$ denote the running time of the Phase I at node $v$. Node $v$ completes the Phase I once $v$ receives all the initial values and the lists of update times and proposals from all neighbors $u \in N_v$. Then, we have

$$R_1^v \leq 1 + \max_{u \in N_v} m_u,$$

where $m_u$ is the number of update times generated by node $u$. Since each $m_u$ is a Poisson random variable with mean $T$, by the concentration of Poisson distribution (Proposition 4), we have

$$\Pr[R_1^v > 5T + 3\log n] \leq \sum_{u \in N_v} \Pr[m_u + 1 > 5T + 3\log n] \leq \frac{1}{n^2}.$$  

Taking a union bound over all nodes implies

$$\Pr[R_1 > 5T + 3\log n] \leq \frac{1}{n}. \quad (10)$$

Suppose all nodes start the Phase II synchronously. The following lemma bounds the time complexity of the Phase II (Algorithm 2) at a fixed node $v \in V$.

**Lemma 7.** Let $(Y_t)_{t \in \mathbb{R}_{\geq 0}}$ be the continuous-time version of a single-site Metropolis chain $(X_t)_{t \geq 0}$. Fix a node $v \in V$. If Condition 2.1 holds for $(X_t)_{t \geq 0}$, then given any $T \geq 0$, suppose all nodes start the Phase II synchronously, with probability at least $1 - \frac{1}{n}$, the Algorithm 2 at node $v$ terminates within $O(T + \log n)$ time units.

Taking a union bound over all nodes implies $R_2 = O(T + \log n)$ with probability at least $1 - \frac{1}{n}$. Combining with (10) implies with high probability, the main algorithm terminates within $O(T + \log n)$ time units. This proves Lemma 6.

The rest of this section is dedicated to the proof of Lemma 7. And we always assume all nodes start the Phase II synchronously in the rest analysis.

**B.1 The dependency chain**

We introduce the definition of the dependency chain, which can be used to bound the time complexity of Algorithm 2.

We fix all the randomness of the main algorithm. Specifically, for each node $v \in V$, fix $m_v$ and $t(v, i), c(v, i), \beta(v, i)$ for $1 \leq i \leq m_v$, where $t(v, i)$ and $c(v, i)$ are update time and proposal of the $i$-th update of node $v$, and $\beta(v, i) \in [0, 1)$ is the random real number sampled in Line 1 of Algorithm 4 to resolve the $i$-th update of node $v$. Thus our main algorithm becomes a deterministic procedure.
Let the pair \((v, i)\) denote the \(i\)-th update of node \(v\) for any \(v \in V\) and any \(1 \leq i \leq m_v\).

In Algorithm 4, the resolution of each update \((v, i)\) belongs to one of following two types of resolutions.

- **Type-I resolution**: node \(v\) resolves the update \((v, i)\) upon computing \(P_{AC}\) and \(P_{RE}\) in Line 3 of Algorithm 4.

- **Type-II resolution caused by \((u, j)\)**: node \(v\) resolves the update \((v, i)\) upon receiving the message “ACCEPT” or “REJECT” from a neighbor \(u \in N_v\), which indicates the result of \(u\) resolving update \((u, j)\).

Given an update \((v, i)\), we use \(D_{v,i}\) to denote the dependency chain end at \((v, i)\). The dependency chain \(D_{v,i}\) is a sequence of updates, which is recursively defined as follows.

- If the resolution of \((v, i)\) is the type-I resolution, construct \(D_{v,i}\) as
  \[
  D_{v,i} = \begin{cases} 
  (v, i) & \text{if } i = 1 \\
  D_{v,i-1}, (v, i) & \text{if } i > 1.
  \end{cases}
  \]

- If the resolution of \((v, i)\) is the type-II resolution caused by \((u, j)\), construct \(D_{v,i}\) as
  \[
  D_{v,i} = D_{u,j}, (v, i).
  \]

Given an execution of the main algorithm, for each update \((v, i)\), the dependency chain \(D_{v,i}\) can be uniquely constructed. Given a dependency chain \(D_{v,i}\), for each adjacent pairs \((v_j, i_j), (v_{j+1}, i_{j+1})\) in \(D_{v,i}\), the resolution of update \((v_j, i_j)\) must occur earlier than the resolution of update \((v_{j+1}, i_{j+1})\). Hence, each dependency chain \(D_{v,i}\) has finite length, where the length of \(D_{v,i}\) is the number of pairs in \(D_{v,i}\).

The following proposition shows the relation between dependency chain and time complexity, which can be easily verified by the definition of time complexity.

**Proposition 8.** Fix a node \(v \in V\). For any \(\ell > 0\), if the time complexity of Algorithm 2 at node \(v\) is \(\ell\), then the length of dependency chain \(D_{v,m_v}\) is at least \(\ell\), where \(m_v\) is the total number of updates generated by node \(v\).

Proposition 8 implies the following lemma.

**Lemma 9.** Fix a node \(v \in V\) and an integer \(\ell > 0\). If the time complexity of Algorithm 2 at node \(v\) is \(\ell'\) such that \(\ell' \geq \ell\), then there exists a time sequence \(0 < t_1 < t_2 < \ldots < t_\ell < T\) and a path \(P = (v_1, v_2, \ldots, v_\ell)\) satisfying \(v_i = v\) and \(v_{j+1} \in N(v_j) \cup \{v_j\}\) for all \(1 \leq j \leq \ell - 1\) such that the following holds.

1. For all \(1 \leq j \leq \ell\), \(t_j\) is an update time of node \(v_j\);

2. For all \(1 \leq j \leq \ell - 1\), suppose \(t_j\) is the \(k\)-th update time of node \(v_j\) and \(t_{j+1}\) is the \(k'\)-th update time of node \(v_{j+1}\), if \(v_j \neq v_{j+1}\), then the resolution of the update \((v_{j+1}, k')\) is the type-II resolution caused by \((v_j, k)\).

**Proof.** Suppose the time complexity of Algorithm 2 at node \(v\) is \(\ell'\) such that \(\ell' \geq \ell\). By Proposition 8, the length of the dependency chain \(D_{v,m_v}\) is at least \(\ell\). Then we can truncation the dependency chain \(D_{v,m_v}\) and only keep the last \(\ell\) pairs

\[
(v_1, i_1), (v_2, i_2), \ldots, (v_\ell, i_\ell),
\]
where \( v_t = v \) and \( i_t = m_v \). Define the time sequence \( t_1, t_2, \ldots, t_\ell \), where \( t_j = t(v_j, i_j) \) is the \( i_j \)-th update time of node \( v_j \). We claim that the time sequence \( t_1, t_2, \ldots, t_\ell \) and the path \( P = (v_1, v_2, \ldots, v_\ell) \) satisfy the properties stated in the lemma.

We prove that \( t_1 < t_2 < \ldots < t_\ell \). Other properties can be verified easily by the definition of dependency chain.

Fix an integer \( 1 \leq j \leq \ell - 1 \). There are two cases for nodes \( v_j \) and \( v_{j+1} \).

- **Case 1:** \( v_j = v_{j+1} \). It must hold that \( i_{j+1} = i_j + 1 \) by the definition of dependency chain. This implies \( t_j < t_{j+1} \) because \( t(v_j, i_j) < t(v_j, i_j + 1) \).
- **Case 2:** \( v_j \neq v_{j+1} \). The resolution of the update \( (v_{j+1}, i_{j+1}) \) is the type-II resolution caused by \( (v_j, i_j) \). Let \( \mathcal{M} \) be the message from \( v_j \) to \( v_{j+1} \) indicating the result of \( v_j \) resolving \( (v_j, i_j) \). Consider the Algorithm 4 at node \( v_{j+1} \) for resolving the update \( (v_{j+1}, t_{j+1}) \). After received the message \( \mathcal{M} \), the set \( S_{j+1}(v_j) \) must be updated. If otherwise, node \( v_{j+1} \) still cannot resolve the update \( (v_{j+1}, i_{j+1}) \). By the definition of \( S_{j+1}(v_j) \) (Definition 4.1), this occurs only if \( t_j = t(v_j, i_j) < t(v_{j+1}, i_{j+1}) = t_{j+1} \).

Combining above two cases proves \( t_1 < t_2 < \ldots < t_\ell \). \( \square \)

### B.2 Proof of Lemma 7

Fix a node \( v \in V \). Let random variable \( R_2^v \) denote the time complexity of Algorithm 2 at node \( v \). We bound the tail probability of the random variable \( R_2^v \).

Fix an integer \( \ell > 0 \). We bound the probability of the event \( R_2^v \geq \ell \). If \( R_2^v \geq \ell \), there must exists a path \( v_1, v_2, \ldots, v_\ell \) together with a time sequence \( t_1, t_2, \ldots, t_\ell \) satisfying the properties stated in Lemma 9.

Fix an integer \( 0 \leq s < \ell \). We define a set of path \( \mathcal{P}(\ell, s) \). Let \( P = v_1 v_2, \ldots, v_\ell \) be a path. The path \( P \in \mathcal{P}(\ell, s) \) if and only if

- \( v = v_1 \);
- for any \( 1 \leq j < \ell \), \( v_{j+1} \in N(v_j) \cup \{v_j\} \);
- \( s = |\{1 \leq j < \ell - 1 \mid v_j \neq v_{j+1}\}| \).

Fix a path \( P \in \mathcal{P}(\ell, s) \) where \( P = v_1, v_2, \ldots, v_\ell \). We say \( P \) is a dependency path if there exists a time sequence \( t_1 < t_2 < \ldots < t_\ell \) such that the properties in Lemma 9 hold with respect to path \( P \) and time sequence \( t_1 < t_2 < \ldots < t_\ell \). By Lemma 9 and a union bound over all paths, we have

\[
\Pr[R_2^v \geq \ell] \leq \sum_{s=0}^{\ell-1} \sum_{P \in \mathcal{P}(\ell, s)} \Pr[P \text{ is dependency path}]. \tag{11}
\]

We bound the probability of the event that the fixed path \( P \) is dependency path. Let random variable \( N \in \mathbb{Z}_{\geq 0} \) denote the total number of rings of \( n \) rate-1 Poisson clocks up to time \( T \). Note that \( N \) is a Poisson random variable with mean \( nT \). We have

\[
\Pr[P \text{ is dependency path}] = \sum_{m \geq 0} \Pr[N = m] \Pr[P \text{ is dependency path} \mid N = m] = e^{-nT} \sum_{m \geq 0} \frac{(nT)^m}{m!} \Pr[P \text{ is dependency path} \mid N = m]. \tag{12}
\]
Fix an integer \( m \). We bound the conditional probability in above equation. Since \( \mathcal{N} = m \), we define the following sequence of random variables 

\[
(U_1, T_1, C_1, \beta_1), (U_2, T_2, C_2, \beta_2), \ldots, (U_m, T_m, C_m, \beta_m).
\]  

(13)

In above sequence, each \( U_i \in V \) is a random node, \( 0 < T_1 < T_2 < \ldots < T_m < T \) are random times, each \( C_i \in [q] \) is a random value sampled from the distribution \( \nu_{U_i} \), each \( \beta_i \in [0, 1) \) is uniformly distributed over \([0, 1)\). Besides, each tuple \((U_i, T_i, C_i, \beta_i)\) satisfies:

- the Poisson clock at node \( U_i \) rings at time \( T_i \);
- node \( U_i \) proposes the random value \( C_i \) for the update at time \( T_i \);
- node \( U_i \) samples the random real number \( \beta_i \in [0, 1) \) to resolve the update at time \( T_i \) in Line 1 of Algorithm 4.

For each \( 1 \leq k \leq m \), let the pair \((T_k, C_k)\) denote the update of node \( U_k \) with update time \( T_k \) and proposal \( C_k \). For any \( 0 < j < i \leq m \), we say that the resolution of an update \((T_i, C_i)\) is caused by the resolution of update \((T_j, C_j)\) if the following occurs: in Algorithm 4 at node \( U_i \), \( U_i \) resolves its update \((T_i, C_i)\) upon \( U_i \) receiving the message “ACCEPT” or “REJECT” from \( U_j \) indicating the result of \( U_j \) resolving update \((T_j, C_j)\).

Conditioning on \( \mathcal{N} = m \), if \( P \) is dependency path, then by Lemma 9, there must exist \( \ell \) indices \( 1 \leq p(1) < p(2) < \ldots < p(\ell) \leq m \) such that the following events occur simultaneously:

- **event** \( \mathcal{A}_1 \): for all \( 1 \leq j \leq \ell \), \( U_{p(j)} = v_j \), where \( v_j \) is the \( j \)-th node in path \( P \);
- **event** \( \mathcal{A}_2^{(j)} \), where \( 1 \leq j \leq \ell - 1 \): \( U_{p(j)} = U_{p(j+1)} \) or the resolution of \((T_{p(j+1)}, C_{p(j+1)})\) is caused by the resolution of \((T_{p(j)}, C_{p(j)})\).

Fix \( \ell \) indices \( 1 \leq p(1) < p(2) < \ldots < p(\ell) \leq m \). We bound the following probability

\[
\Pr \left[ \mathcal{A}_1 \land \left( \bigwedge_{j=1}^{\ell-1} \mathcal{A}_2^{(j)} \right) \mid \mathcal{N} = m \right]
\]

\[
= \Pr[\mathcal{A}_1 \mid \mathcal{N} = m] \prod_{j=1}^{\ell-1} \Pr \left[ \mathcal{A}_2^{(j)} \mid \mathcal{N} = m \land \mathcal{A}_1 \land \left( \bigwedge_{k=1}^{j-1} \mathcal{A}_2^{(k)} \right) \right].
\]  

(14)

Due to the memoryless property of exponential random variable, conditioning on any historical rings of Poisson clocks, once a Poisson clock rings, such clock is chosen uniformly at random from \( n \) clocks. Hence, conditioning on \( \mathcal{N} = m \), each \( U_i \) is a uniform and independent random node in \( V \).

\[
\Pr[\mathcal{A}_1 \mid \mathcal{N} = m] \leq \left( \frac{1}{n} \right) ^\ell.
\]  

(15)

Further, we claim the following holds for the event \( \mathcal{A}_2^{(j)} \).

**Claim 10.** If Condition 2.1 is satisfied, then for any \( 1 \leq j < \ell \) satisfying \( v_j \neq v_{j+1} \), it holds that

\[
\Pr \left[ \mathcal{A}_2^{(j)} \mid \mathcal{N} = m \land \mathcal{A}_1 \land \left( \bigwedge_{k=1}^{j-1} \mathcal{A}_2^{(k)} \right) \right] \leq \frac{2C}{\Delta},
\]

where \( C \) is the constant in Condition 2.1.
Since $P \in \mathcal{P}(\ell, s)$, there are exactly $s$ indices $j$ such that $v_j \neq v_{j+1}$. Combining (14), (15) and Claim 10 yields

$$\Pr \left[ A_1 \land \left( \bigwedge_{j=1}^{\ell-1} A_{\ell}^j \right) \mid N = m \right] \leq \left( \frac{1}{n} \right)^\ell \left( \frac{2C}{\Delta} \right)^s.$$

Taking a union bound over $\binom{m}{\ell}$ possible indices $1 \leq p(1) < p(2) < \ldots < p(\ell) \leq m$ yields

$$\Pr[\text{P is dependency path} \mid N = m] \leq \binom{m}{\ell} \left( \frac{1}{n} \right)^\ell \left( \frac{2C}{\Delta} \right)^s.$$

Finally, note that $|\mathcal{P}(\ell, s)| \leq \binom{\ell-1}{s} \Delta^s$, using (11) and (12) yields

$$\Pr[R_{\ell}^w \geq \ell] \leq \sum_{s=0}^{\ell-1} \sum_{P \in \mathcal{P}(\ell, s)} e^{-nT} \sum_{m=0}^{\infty} \frac{(nT)^m}{m!} \left( \frac{m}{\ell} \right)^\ell \left( \frac{1}{n} \right)^\ell \left( \frac{2C}{\Delta} \right)^s \leq \sum_{s=0}^{\ell-1} \binom{\ell-1}{s} \Delta^s e^{-nT} \sum_{m=\ell}^{\infty} \frac{(nT)^m}{m!} \left( \frac{m}{\ell} \right)^\ell \left( \frac{1}{n} \right)^\ell \left( \frac{2C}{\Delta} \right)^s \leq \frac{T^\ell}{\ell!} (1 + 2C)^\ell.$$

Note that $\ell! \geq \left( \frac{\ell}{e} \right)^\ell$. We have

$$\Pr[R_{\ell}^w \geq \ell] \leq \left( \frac{Te(1 + 2C)}{\ell} \right)^\ell.$$

Let $r = \max \{2e(1 + 2C)T, 2\log n\}$. We have

$$\Pr[R_{\ell}^w \geq r] \leq \left( \frac{1}{2} \right)^{2\log n} = \frac{1}{n^2}.$$

Hence, Algorithm 2 at node $v$ has time complexity $O(T + \log n)$ with probability at least $1 - 1/n^2$. This proves the lemma.

Proof. (Proof of Claim 10) Recall $P = v_1, v_2, \ldots, v_\ell$ is the fixed path and $1 \leq p(1) < p(2) < \ldots < p(\ell) \leq m$ is the $\ell$ fixed indices. Fix an integer $1 \leq j \leq \ell - 1$ such that $v_j \neq v_{j+1}$. Consider the random variables

$$(U_1, T_1, C_1, \beta_1), (U_2, T_2, C_2, \beta_2), \ldots, (U_m, T_m, C_m, \beta_m)$$

defined in (13). We fix the values of random variables as follows.

- $\mathcal{F}_1$: Fix the randomness of $n$ rate-1 Poisson clocks such that $N = m$ and event $A_1$ occurs. Then the values of all variables $U_1, U_2, \ldots, U_m$ and $T_1 < T_2 < \ldots < T_m$ are fixed and $U_{p(k)} = v_k$ for all $1 \leq k \leq \ell$.

- $\mathcal{F}_2$: Fix the values of all variables $C_k, \beta_k$ for $1 \leq k < p(j + 1)$.

Furthermore, fix the the delays of messages as follows:
• \( \mathcal{F}_3 \): Fix the delays of messages \( \mathcal{M}_k(u) \) for all \( 1 \leq k < p(j + 1) \) and all \( u \in N(U_k) \), where \( \mathcal{M}_k(u) \) denotes the message from \( U_k \) to \( u \) indicating the result of \( U_k \) resolving the update \((T_k, C_k)\). All fixed the delays must be consistent with the message-passing model, where each unidirectional channel is a reliable FIFO channel.

Given any \( \mathcal{F}_1, \mathcal{F}_2 \) and \( \mathcal{F}_3 \) defined as above, we show the following two results:

• the occurrences of events \( \mathcal{N} = m \), \( \mathcal{A}_1 \) and \( \mathcal{A}_2^{(k)} \) for all \( 1 \leq k \leq j - 1 \) are determined.

• if Condition 2.1 is satisfied, then event \( \mathcal{A}_2^{(j)} \) occurs with probability at most \( \frac{2C}{\Delta} \):

\[
\Pr[\mathcal{A}_2^{(j)} | \mathcal{F}_1 \land \mathcal{F}_2 \land \mathcal{F}_3] \leq \frac{2C}{\Delta},
\]

where the probability takes over the randomness of unfixed variables.

The claim follows by combining above two results.

The events \( \mathcal{N} = m \) and \( \mathcal{A}_1 \) must occur due to \( \mathcal{F}_1 \). We prove that the occurrences of events \( \mathcal{A}_2^{(k)} \) for all \( 1 \leq k \leq j - 1 \) are determined.

Given \( \mathcal{F}_1 \) and \( \mathcal{F}_2 \), all values \( Y_t(v) \) for \( v \in V \) and \( 0 \leq t < T_{p(j+1)} \) are fixed, where \( (Y_t)_{t \in \mathbb{R}_{\geq 0}} \) is continuous-time chain generated by the main algorithm. This is because all update information before time \( T_{p(j+1)} \) is fixed. Fix any \( 1 \leq e < p(j + 1) \). Consider the Algorithm 4 at node \( U_e \) for resolving update \((T_e, C_e)\). Node \( U_e \) maintains two thresholds \( P_{AC}, P_{RE} \), which are functions of \( Y_{T\_e}(U_e), C_e \) and all sets \( S_{T\_e}(u) \) in (5) for \( u \in N(U_e) \). The update \((T_e, C_e)\) is resolved once \( \beta_e < P_{AC} \) or \( \beta_e \geq 1 - P_{RE} \). Hence, node \( U_e \) only uses the following information when resolving the update \((T_e, C_e)\):

(i) The value \( Y_{T\_e}(U_e) \), the proposal \( C_e \) and the random real number \( \beta_e \), which are fixed by \( \mathcal{F}_1 \) and \( \mathcal{F}_2 \);

(ii) Proposals \( C_{e'} \) for all \( e' < e \) satisfying \( U_{e'} \in N(U_e) \), which are fixed by \( \mathcal{F}_2 \);

(iii) Messages \( \mathcal{M}_{e'}(U_e) \) for all \( e' < e \) satisfying \( U_{e'} \in N(U_e) \), where \( \mathcal{M}_{e'}(U_e) \) is the message from \( U_{e'} \) to \( U_e \) indicating the result of \( U_{e'} \) resolving the update \((T_{e'}, C_{e'})\). The contents of these messages are fixed by \( \mathcal{F}_1 \) and \( \mathcal{F}_2 \), and the delays of these messages are fixed by \( \mathcal{F}_3 \).

The (ii) and (iii) contain all information for node \( U_e \) to compute the sets \( S_{T\_e}(u) \) in (5) for all \( u \in N(U_e) \). Note that the update \((T_e, C_e)\) must be resolved no later than the moment when all messages \( \mathcal{M}_{e'}(U_e) \) for \( e' < e \) satisfying \( U_{e'} \in N(U_e) \) are delivered. Once \((T_e, C_e)\) is resolved, node \( U_e \) sends \( \mathcal{M}_e(u) \) to all \( u \in N(U_e) \). An induction on \( e \) from 1 to \( p(j + 1) - 1 \) shows that given \( \mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3 \), the procedures of resolving updates \((T_e, C_e)\) for all \( 1 \leq e < p(j + 1) \) are fully determined. This implies the occurrences of \( \mathcal{A}_2^{(k)} \) for all \( 1 \leq k \leq j - 1 \) are fully determined.

Consider the event \( \mathcal{A}_2^{(j)} \). Recall \( U_{p(j)} \in N(U_{p(j+1)}) \) due to \( \mathcal{F}_1 \). Recall the event \( \mathcal{A}_2^{(j)} \) occurs if node \( U_{p(j+1)} \) resolves its update \((T_{p(j+1)}, C_{p(j+1)})\) upon receiving the message \( \mathcal{M}_{p(j)}(U_{p(j+1)}) \) from its neighbor \( U_{p(j)} \). Given \( \mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3 \), the moment \( T \) at which node \( U_{p(j+1)} \) receives the message \( \mathcal{M}_{p(j)}(U_{p(j+1)}) \) is fixed. Let \( T - \epsilon \) be the moment right before the moment \( T \). For each neighbor \( u \in N(U_{p(j+1)}) \), let \( S(u) \) denote the set \( S_{T\_e}(u) \) in (5) evaluated by node \( U_{p(j+1)} \) at moment \( T - \epsilon \) and let \( S'(u) \) denote the set \( S_{T\_e}(u) \) in (5) evaluated by node \( U_{p(j+1)} \) at moment \( T \). The following holds for sets \( S(u), S'(u) \) for all \( u \in N(U_{p(j+1)}) \):

• For all \( u \in N(U_{p(j+1)}) \), two sets \( S(u), S'(u) \) are fixed given \( \mathcal{F}_1, \mathcal{F}_2 \) and \( \mathcal{F}_3 \).

• For all \( u \in N(U_{p(j+1)}) \) \( \setminus \{U_{p(j)}\} \), \( S(u) = S'(u) \).
\[ S'(U_{p(j)}) \subseteq S(U_{p(j)}) \text{ and } |S(U_{p(j)})| - |S'(U_{p(j)})| \leq 1. \]

For all \( u \in N(U_{p(j+1)}) \), \( |S'(u)| \geq 1 \).

The first result holds because computing each set \( S_{T_{p(j+1)}}(u) \) depends only on proposals \( C_k \) and messages \( M_k(U_{p(j+1)}) \) for all \( k < p(j+1) \) satisfying \( U_k = u \), which are fixed by \( F_1, F_2, F_3 \) (including the delays of these messages). The second and the third results hold because only the message \( M_{p(j)}(U_{p(j+1)}) \) is received by \( U_{p(j+1)} \) between the moment \( T - \epsilon \) and the moment \( T \). The fourth result holds because the set \( S_{T_{p(j+1)}}(u) \) must contain at least one element by (5).

Let \( c \) denote \( Y_t(U_{p(j+1)}) \) where \( t = T_{p(j+1)} - \epsilon \). Note that the value \( c \) is fixed by \( F_1 \) and \( F_2 \). Let \( C' \) denote the random proposal \( C_{p(j+1)} \). Note that \( C' \) is still a random proposal from proposal distribution \( \nu_{p(j+1)} \) given \( F_1, F_2, F_3 \). Define \( P_{AC}, P_{RE}, P'_{AC} \) and \( P'_{RE} \) as:

\[
P_{AC} = \min_{\tau \in C} f_{c,C'}(U_{p(j+1)}(\tau)) \quad \text{and} \quad P_{RE} = 1 - \max_{\tau \in C} f_{c,C'}(U_{p(j+1)}(\tau))
\]

\[
P'_{AC} = \min_{\tau \in C'} f_{c,C'}(U_{p(j+1)}(\tau)) \quad \text{and} \quad P'_{RE} = 1 - \max_{\tau \in C'} f_{c,C'}(U_{p(j+1)}(\tau))
\]

where

\[
C = \bigotimes_{w \in N(U_{p(j+1)})} S(w), \quad C' = \bigotimes_{w \in N(U_{p(j+1)})} S'(w).
\]

If the event \( A_2^{(j)} \) occurs, then the following event \( B_j \) must occur.

- **event \( B_j \)**: \( (P_{AC} \leq \beta_{p(j+1)} < 1 - P_{RE}) \land (\beta_{p(j+1)} < P'_{AC} \lor \beta_{p(j+1)} \geq 1 - P'_{RE}) \).

If the event \( A_2^{(j)} \) occurs, then node \( U_{p(j+1)} \) resolves the update \( (T_{p(j+1)}, C_{p(j+1)}) \) at moment \( T \). Recall \( T \) is the moment at which node \( U_{p(j+1)} \) receives the message \( M_{p(j)}(U_{p(j+1)}) \) from node \( U_{p(j)} \). Node \( U_{p(j+1)} \) cannot resolve the update at moment \( T - \epsilon \), this implies \( P_{AC} \leq \beta_{p(j+1)} < 1 - P_{RE} \). Node \( U_{p(j+1)} \) resolves the update at moment \( T \), this implies \( \beta_{p(j+1)} < P'_{AC} \lor \beta_{p(j+1)} \geq 1 - P'_{RE} \).

Note that \( \beta_{p(j+1)} \) is a random real number uniformly distributed over \([0,1)\) and \( C' = C_{T_{p(j+1)}} \) is a random value from proposal distribution \( \nu_{p(j+1)} \) given \( F_1, F_2, F_3 \). Recall that the value \( c = Y_t(U_{p(j+1)}) \) (where \( t = T_{p(j+1)} - \epsilon \)) and all sets \( S(u), S'(u) \) for \( u \in N(U_{p(j+1)}) \) are fixed by \( F_1, F_2, F_3 \). Then we have

\[
Pr[A_2^{(j)} \mid F_1 \land F_2 \land F_3] \leq Pr[B_j \mid F_1 \land F_2 \land F_3] \leq \mathbb{E}_{C' \sim \nu^*} [(P'_{AC} - P_{AC}) + (P'_{RE} - P_{RE}) \mid F_1 \land F_2 \land F_3], \tag{16}
\]

where the distribution \( \nu^* = \nu_{p(j+1)} \). The last inequality holds because \( \beta_{p(j+1)} \) is uniformly distributed over \([0,1)\), \( P'_{AC} \geq P_{AC} \) and \( P'_{RE} \geq P_{RE} \) (because \( S'(u) \subseteq S(u) \) for all \( u \in N(U_{p(j+1)}) \)).

Finally, to bound the probability in (16), we introduce the following optimization problem.
Fix two nodes \( \{v, u\} \in E \), define an optimization problem \( \Psi(v, u) \) as follows.

\[
\begin{align*}
\text{variables} & \quad S_1(w) \subseteq [q], S_2(w) \subseteq [q] & \forall w \in N_v \\
& \quad c \in [q] \\
\text{maximize} & \quad \sum_{c' \in [q]} \nu_c(c') \left( \min_{\tau \in \mathcal{C}_2} f_{c,c'}^v(\tau) - \min_{\tau \in \mathcal{C}_1} f_{c,c'}^v(\tau) + \max_{\tau \in \mathcal{C}_2} f_{c,c'}^v(\tau) - \max_{\tau \in \mathcal{C}_1} f_{c,c'}^v(\tau) \right) \\
\text{subject to} & \quad \mathcal{C}_1 = \bigotimes_{w \in N_v} S_1(w), \quad \mathcal{C}_2 = \bigotimes_{w \in N_v} S_2(w) \\
& \quad S_2(u) \subset S_1(u) \\
& \quad |S_1(u)| - |S_2(u)| = 1 \\
& \quad |S_2(w)| \geq 1 & \forall w \in N_v \\
& \quad S_2(w) = S_1(w) & \forall w \in N_v \setminus \{u\}
\end{align*}
\]

The probability in (16) must be upper bounded by the optimal value of the objective function in above optimization problem \( \Psi(v, u) \) with \( v = U_{p(j+1)} \) and \( u = U_{p(j)} \), because it takes the worst case over all possible sets \( S(w), S'(w) \) (which are variables \( S_1(w), S_2(w) \) in above optimization problem) for \( w \in N(U_{p(j+1)}) \) and the value \( c = Y_t(U_{p(j+1)}) \) where \( t = T_{p(j+1)} - \epsilon \) that maximize the expectation in (16). Remark that we use constraint \( |S_1(u)| - |S_2(u)| = 1 \) rather than \( |S_1(u)| - |S_2(u)| \leq 1 \) because the value of the objective function is 0 if \( |S_1(u)| = |S_2(u)| \).

We claim the optimal value of the objective function in problem \( \Psi(v, u) \) with \( v = U_{p(j+1)} \) and \( u = U_{p(j)} \) is at most \( 2 \max_{a,b,c \in [q]} \mathbb{E}_{c' \sim u^*} \left[ \delta_{U_{p(j)},a,b} f^v_{c,c'} \right] \), where \( u^* = \nu_{U_{p(j+1)}}^* \). This result is proved in Section B.3. By Condition 2.1, such value is at most \( \frac{2C}{\Delta} \). For any \( \mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3 \), we have

\[
\Pr[A_2^{(j)} | \mathcal{F}_1 \land \mathcal{F}_2 \land \mathcal{F}_3] \leq \frac{2C}{\Delta}.
\]

Besides, given any \( \mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3 \), the occurrences of the events \( N = m, A_1 \) and \( A_2^{(k)} \) for all \( 1 \leq k \leq j-1 \) are determined. This implies

\[
\Pr \left[ A_2^{(j)} | N = m \land A_1 \land \left( \bigwedge_{k=1}^{j-1} A_2^{(k)} \right) \right] \leq \frac{2C}{\Delta}.
\]

\( \square \)

### B.3 Analysis of the optimization problem

**Lemma 11.** Fix an edge \( \{v, u\} \in E \). Let \( OPT \) denote the objective function value of the optimal solution to problem \( \Psi(v, u) \) defined in (17). It holds that

\[
OPT \leq 2 \max_{a,b,c \in [q]} \mathbb{E}_{c' \sim u^*} \left[ \delta_{U_{p(j)},a,b} f^v_{c,c'} \right].
\]

**Proof.** Suppose we replace the objective function in problem \( \Psi(v, u) \) defined in (17) as

\[
\begin{align*}
\text{maximize} & \quad \sum_{c' \in [q]} \nu_c(c') \left( \min_{\tau \in \mathcal{C}_2} f_{c,c'}^v(\tau) - \min_{\tau \in \mathcal{C}_1} f_{c,c'}^v(\tau) + \max_{\tau \in \mathcal{C}_2} f_{c,c'}^v(\tau) - \max_{\tau \in \mathcal{C}_1} f_{c,c'}^v(\tau) \right)
\end{align*}
\]

and keep all variables and constraints unchanged. We obtain a new optimization problem. Let \( OPT_1 \) denote the objective function value of the optimal solution to this problem.

25
Similarly, suppose we replace the objective function in problem \( \mathcal{P}(v, u) \) defined in (17) as
\[
\max \sum_{e' \in [q]} \nu_{e'}(c') \left( \max_{\tau \in C_1} f_{e', e'}^v(\tau) - \max_{\tau \in C_2} f_{e', e'}^v(\tau) \right)
\]
and keep all variables and constraints unchanged. We obtain another new optimization problem. Let \( \text{OPT}_2 \) denote the objective function value of the optimal solution to this problem.

It is easy to verify
\[
\text{OPT} \leq \text{OPT}_1 + \text{OPT}_2.
\]

We show that
\begin{align*}
\text{OPT}_1 & \leq \max_{a, b, c \in \mathcal{V}} \mathbb{E}_{c' \sim \nu_c} \left[ \delta_{u, a, b} f_{e, e'}^v \right] \quad (19) \\
\text{OPT}_2 & \leq \max_{a, b, c \in \mathcal{V}} \mathbb{E}_{c' \sim \nu_c} \left[ \delta_{u, a, b} f_{e, e'}^v \right]. \quad (20)
\end{align*}

This proves the lemma.

We prove inequality (19). Inequality (20) can be proved by going through a similar proof.

Consider the new optimization problem with objective function (18). We claim the following result for this problem.

**Claim 12.** There exists an optimal solution \( \text{SOL}^* = (S_1^*, S_2^*, c^*) \) such that \( |S_2^*(u)| = 1 \).

Thus, we have \( |S_1^*(u)| = 2 \) due to the constraint of the problem. Suppose \( S_1^*(u) = \{a, b\} \) and \( S_2^*(u) = \{b\} \). Fix a value \( c' \in [q] \), define
\[
\gamma_1(c') = \min_{\tau \in C_1} f_{e, e'}^v(\tau) = f_{e, e'}^v(\tau')
\]
\[
\gamma_2(c') = \min_{\tau \in C_2} f_{e, e'}^v(\tau) = f_{e, e'}^v(\tau'')
\]
where
\[
C_1^* = \bigotimes_{w \in \mathcal{V} \setminus u} S_1^*(w), \quad C_2^* = \bigotimes_{w \in \mathcal{V} \setminus u} S_2^*(w),
\]
and \( \tau' = \arg \min_{\tau \in C_1^*} f_{e, e'}^v(\tau) \), \( \tau'' = \arg \min_{\tau \in C_2^*} f_{e, e'}^v(\tau) \). It must hold that \( \tau'' = b \) because \( S_2^*(u) = \{b\} \). There are two cases for \( \tau'_u \): \( \tau'_u = a \) or \( \tau'_u = b \), because \( S_1^*(u) = \{a, b\} \).

Suppose \( \tau'_u = b \). Since \( S_1^*(w) = S_2^*(w) \) for all \( w \in \mathcal{V} \setminus \{u\} \), then we must have
\[
\gamma_2(c') - \gamma_1(c') = 0 \leq \delta_{u, a, b} f_{e, e'}^v.
\]

Suppose \( \tau'_u = a \). We define \( \tau'' = \{b\} \) as
\[
\tau''_w = \begin{cases} b & \text{if } w = u, \\ \tau'_w & \text{if } w \neq u. \end{cases}
\]

Note that \( b \in S_2^*(u) \) and \( \tau''_w \in S_2^*(w) \) for all \( w \in \mathcal{V} \setminus \{u\} \) (because \( S_2^*(w) = S_1^*(w) \)). We have \( \tau'' \in C_2^* \), which implies \( f_{e, e'}^v(\tau') \geq f_{e, e'}^v(\tau'') \). Hence
\[
\gamma_2(c') - \gamma_1(c') = f_{e, e'}^v(\tau'') - f_{e, e'}^v(\tau') \leq f_{e, e'}^v(\tau'') - f_{e, e'}^v(\tau'') \leq \delta_{u, a, b} f_{e, e'}^v.
\]
The last inequality is because \( \tau' \) and \( \tau'' \) agree on all nodes except \( u \) and \( \tau'_u = a, \tau''_u = b \).
Combining above two cases together, we have

\[
\text{OPT}_1 = \sum_{c' \in [q]} \nu_v(c') \left( \min_{\tau \in C_2^*} f_{c',c'}(\tau) - \min_{\tau \in C_1^*} f_{c',c'}(\tau) \right)
\]

\[
= \sum_{c' \in [q]} \nu_v(c') \left( \gamma_2(c') - \gamma_1(c') \right)
\]

\[
\leq \sum_{c' \in [q]} \nu_v(c') \delta_{a,a,b} f_{c',c'}^v
\]

\[
= \mathbb{E}_{c' \sim \nu_v} \left[ \delta_{a,a,b} f_{c',c'}^v \right]
\]

\[
\leq \max_{a,b,c \in [q]} \mathbb{E}_{c' \sim \nu_v} \left[ \delta_{a,a,b} f_{c',c'}^v \right].
\]

This proves the inequality \((19)\). \qed

**Proof.** (Proof of Claim 12) Suppose \(\text{SOL}^* = (S_1^*, S_2^*, c^*)\) is an optimal solution with \(S_2^*(u) > 1\). Let \(b \in [q]\) be an arbitrary element in \(S_2^*(u)\). We move the element \(b\) in both \(S_1^*(u)\) and \(S_2^*(u)\) to obtain a new solution \(\text{SOL}^\circ = \{S_1^\circ, S_2^\circ, c^\circ\}\). Namely

\[
S_1^\circ(u) = S_1^*(u) \setminus \{b\}
\]

\[
S_2^\circ(u) = S_2^*(u) \setminus \{b\}
\]

and \(c^\circ = c^*, S_1^\circ(w) = S_1^*(w), S_2^\circ(w) = S_2^*(w)\) for all \(w \in N_v \setminus \{u\}\). It is easy to verify that the new solution \(\text{SOL}^\circ\) also satisfies all the constraints. We will prove that \(\text{SOL}^\circ\) is also an optimal solution. Since \(|S_2^*(u)| = |S_2^*(u)| - 1\), then we can repeat this argument to find the optimal solution \(\text{SOL}^*\) with \(|S_2^*(u)| = 1\).

We denote the objective function value of the solution \(\text{SOL}^*\) as

\[
g(\text{SOL}^*) = \sum_{c' \in [q]} \nu_v(c') \left( \min_{\tau \in C_2^*} f_{c',c'}^v(\tau) - \min_{\tau \in C_1^*} f_{c',c'}^v(\tau) \right),
\]

where

\[
C_1^* = \bigotimes_{w \in N_v} S_1^*(w), \quad C_2^* = \bigotimes_{w \in N_v} S_2^*(w).
\]

Similar, we denote objective function value of the solution \(\text{SOL}^\circ\) as \(g(\text{SOL}^\circ)\). Suppose \(S_1^*(u) \setminus S_2^*(u) = \{a\}\). We define the following set of values \(S_a \subseteq [q]\) as

\[
S_a \triangleq \left\{ c' \in q \mid \min_{\tau \in C_2^\circ} f_{c',c'}^v(\tau) > \min_{\tau \in C_1^\circ} f_{c',c'}^v(\tau) \right\}.
\]

Note that \(C_2^\circ \subset C_1^\circ\). We must have \(\min_{\tau \in C_2^\circ} f_{c',c'}^v(\tau) \geq \min_{\tau \in C_1^\circ} f_{c',c'}^v(\tau)\) for all \(c' \in [q]\). Then \(g(\text{SOL}^\circ)\) can be rewritten as

\[
g(\text{SOL}^\circ) = \sum_{c' \in S_a} \nu_v(c') \left( \min_{\tau \in C_2^\circ} f_{c',c'}^v(\tau) - \min_{\tau \in C_1^\circ} f_{c',c'}^v(\tau) \right).
\] (21)

For each \(c' \in S_a\), suppose \(\min_{\tau \in C_1^\circ} f_{c',c'}^v(\tau) = f_{c',c'}^v(\tau^a)\), then it must hold that \(\tau^a = a\). This is because \(S_1^*(u)\) and \(S_2^*(u)\) differ only at element \(a\) and \(S_1^*(w) = S_2^*(w)\) for all \(w \in N_v \setminus \{u\}\). If \(\tau^a \neq a\), we must have \(\min_{\tau \in C_2^\circ} f_{c',c'}(\tau) = \min_{\tau \in C_1^\circ} f_{c',c'}(\tau)\). This is contradictory to \(c' \in S_a\).
Consider the solution \( \text{SOL}^o \). Note that the set \( S^1_1(u) = S^*_1(u) \setminus \{b\} \) and \( S^2_2(u) = S^*_2(u) \setminus \{b\} \), where \( b \neq a \) because \( b \in S^*_2(u) \). By the definition of \( \text{SOL}^o(u) \), we have

\[
\forall c' \in S_a : \min_{\tau \in \mathcal{C}^2_1} f^v_{c', c'}(\tau) = \min_{\tau \in \mathcal{C}^o_1} f^v_{c', c'}(\tau) \tag{22}
\]

\[
\forall c' \in S_a : \min_{\tau \in \mathcal{C}^2_2} f^v_{c', c'}(\tau) \leq \min_{\tau \in \mathcal{C}^o_2} f^v_{c', c'}(\tau), \tag{23}
\]

where

\[
\mathcal{C}^o_1 = \bigotimes_{w \in N_v} S^o_1(w), \quad \mathcal{C}^o_2 = \bigotimes_{w \in N_v} S^o_2(w).
\]

Recall that \( c^o = c^* \), \( S^1_i(w) = S^*_1(w) \), \( S^2_i(w) = S^*_2(w) \) for all \( w \in N_v \setminus \{u\} \). Thus (22) holds because \( a \in S^1_i(u) \) and (23) holds because \( S^2_2(u) \subset S^*_2(u) \) (hence \( \mathcal{C}^o_2 \subset \mathcal{C}^*_2 \)). Combining (22) and (23) together, we have

\[
g(\text{SOL}^o) = \sum_{c' \in \mathcal{S}_a} \nu_v(c') \left( \min_{\tau \in \mathcal{C}^2_2} f^v_{c', c'}(\tau) - \min_{\tau \in \mathcal{C}^o_1} f^v_{c', c'}(\tau) \right)
\]

\[
\geq \sum_{c' \in \mathcal{S}_a} \nu_v(c') \left( \min_{\tau \in \mathcal{C}^2_2} f^v_{c', c'}(\tau) - \min_{\tau \in \mathcal{C}^o_2} f^v_{c', c'}(\tau) \right)
\]

\[
\geq \sum_{c' \in \mathcal{S}_a} \nu_v(c') \left( \min_{\tau \in \mathcal{C}^2_2} f^v_{c', c'}(\tau) - \min_{\tau \in \mathcal{C}^o_1} f^v_{c', c'}(\tau) \right)
\]

\[
= g(\text{SOL}^*),
\]

where the last equation holds due to (21). Thus \( \text{SOL}^o \) is also an optimal solution. \( \square \)