DISTRIBUTED METROPOLIS SAMPLER WITH OPTIMAL PARALLELISM

WEIMING FENG, THOMAS P. HAYES, AND 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 faithfully simulates sequential single-site Metropolis chains without introducing any bias. When a natural Lipschitz condition for the the Metropolis filters is satisfied, the algorithm can faithfully 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 graphical models, including proper graph coloring, hardcore model, and Ising model, our condition for linear speedup is much weaker than the uniqueness conditions for the respective models.

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 without introducing any bias.

1. INTRODUCTION

Sampling from joint distributions represented by graphical models is one of the central topics in various fields, including randomized algorithms, statistics, machine learning, and data analysis. The Metropolis-Hastings method is a fundamental Markov chain Monte Carlo (MCMC) method for sampling. Let μ be a joint distribution for a set V of n random variables, each with domain $[q]$. The classic single-site Metropolis chain for sampling from μ is described as follows.

Algorithm 1: single-site Metropolis sampler for μ

```
  Input: initial configuration $X_0 \in [q]^V$
1 for $t = 1$ to $N$ do
2     pick $v \in V$ uniformly at random;
3     sample a random $c' \in [q]$ and construct $X' \in [q]^V$ by modifying $X_t(v)$ to $c'$;
4     with probability $\min\left\{1, \frac{\mu(X')}{\mu(X_t)}\right\}$, set $X_{t+1} \leftarrow X'$; otherwise, set $X_{t+1} \leftarrow X_t$;
```

When μ is described by a graphical model on an undirected graph $G = (V, E)$ (see, for instance, [MM09, KF09, FVY19]), due to the conditional independence property of graphical models, for each node $v \in V$, it holds for the marginal distribution μ_v that $\mu_v\left(\cdot \mid X_{V\setminus\{v\}}\right) = \mu_v\left(\cdot \mid X_{N(v)}\right)$, where $N(v) = \{\, u \in V \mid \{u, v\} \in E \,\}$ denotes the neighborhood of v in G. Therefore, the Metropolis filter $\min\left\{1, \frac{\mu(X')}{\mu(X_t)}\right\}$ in Algorithm 1 can be computed locally at v by accessing its immediate neighbors as:

$$\frac{\mu(X')}{\mu(X_t)} = \frac{\mu_v(X'(v) \mid X_t(N_v))}{\mu_v(X_t(v) \mid X_t(N_v))}.$$

Consider for example a canonical graphical model, the *uniform proper graph colorings*. Here μ is the uniform distribution over all proper q-colorings of G. And Algorithm 1 instantiates to the following

(Weiming Feng, Yitong Yin) STATE KEY LABORATORY FOR NOVEL SOFTWARE TECHNOLOGY, NANJING UNIVERSITY. E-mail: fengwm@smail.nju.edu.cn,yinyt@nju.edu.cn

(Thomas P. Hayes) DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF NEW MEXICO. E-mail: hayes@cs.unm.edu

Weiming Feng and Yitong Yin are supported by the National Key R&D Program of China 2018YFB1003202 and the National Science Foundation of China under Grant Nos. 61722207 and 61672275. Thomas P. Hayes is partially supported by NSF CAREER award CCF-1150281.

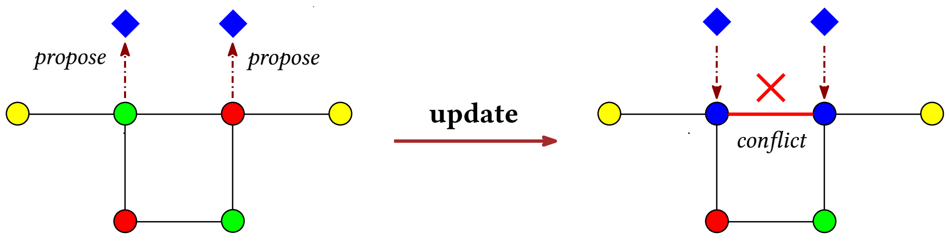

FIGURE 1. Concurrent updates of adjacent nodes resulting in improper coloring.

well known Markov chain on proper q-colorings: at each step, a vertex $v \in V$ is picked uniformly at random, and is recolored to a uniformly random color $c' \in [q]$ if it is proper.

The above single-site Metropolis chain $(X_t)_{t\geq 0}$ is inherently sequential. Meanwhile, it is well known that $(X_t)_{t\geq 0}$ is in fact a discretization of the following continuous-time process $(Y_t)_{t\in\mathbb{R}_{\geq 0}}$, which is parallel in nature:

Algorithm 2: continuous-time Metropolis chain for μ

```
  Input: initial configuration $Y \in [q]^V$
1 each node $v \in V$ is associated with an i.i.d. rate-1 Poisson clock;
2 for $t = 0$ to $T$ continuously do
3     whenever a clock at a node $v$ rings, $Y(v)$ is updated instantly as Line 3–4 in Algorithm 1;
```

It is well known that the two processes $(X_t)_{t\geq 0}$ and $(Y_t)_{t\in\mathbb{R}_{\geq 0}}$ are equivalent up to a speedup factor $n = |V|$. Formally, Y_T is identically distributed as X_N for $N \sim \text{Pois}(nT)$. In fact, the continuous-time chain was introduced much earlier by physicists [Gla63], and has been extensively studied for analyzing the mixing of the discrete-time chain [LP17].

1.1. Distributed sampling from graphical models.

The boom in Big Data applications in contemporary Machine Learning has been drawing increased attention to distributed algorithms for sampling [NASW07, DVMGK09, SWA09, YXQ09, GLGG11, AAG⁺12, DSOR16, DSZOR15, DDJ18, KKSP18].

Although the idealized parallel procedure described in Algorithm 2 has been discovered for over half a century, little has been known for how to execute it correctly and efficiently in distributed systems. Due to the non-atomicity of the update operations to the variables, concurrent accesses to adjacent variables may cause *race conditions* that results in faulty sampling. For example, for uniform proper graph coloring, as illustrated in Figure 1, updating colors of adjacent nodes simultaneously may end up with improper coloring.

Such issue can be resolved by a concurrency control to avoid concurrent updates to adjacent variables. However, this will cause a $\Delta = \max_v |N(v)|$ factor slowdown to the parallel running time since within each neighborhood at most one variable can be updated at a time.

We are concerned with the following fundamental question:

Can we faithfully execute the continuous-time chain by distributed algorithms without significant slowdown?

A positive answer to the question would provide an optimal faithful implementation for the classic idealized process, and moreover, for the first time, would immediately convert all rapid mixing results for the sequential chain to accurate and efficient distributed sampling algorithms. Meanwhile, existing approaches for distributed sampling from graphical models suffer from either slowdown to the parallel running time [GLGG11], or inaccuracy of sampling [SN10, DSOR16, DDJ18], or more restrictive regimes for rapid mixing than the sequential chain [FSY17, FG18, FHY18, FVY19].

1.2. **Our Results.** We give a distributed Metropolis sampler that achieves optimal linear parallel speedup.

The distributed algorithms are described in the *fully-asynchronous* message-passing model [Lyn96]. The communication network is the graph $G = (V, E)$ for the graphical model, with nodes as processors and edges as channels, and all communications are asynchronous, with every message delivered within at most one *time unit* with adversarial delay. For more details, see Section 2.

1.2.1. *Uniform proper q-coloring.* To exemplify our results, we first consider the Metropolis chain for graph coloring. Consider the continuous-time Metropolis chain $(Y_t)_{t\in\mathbb{R}_{\geq 0}}$ (as defined in Algorithm 2) for uniform proper q-coloring on a graph $G = (V, E)$. Let $n = |V|$ and Δ denote the maximum degree of G.

Theorem 1.1. *Assume that $q \geq \alpha\Delta$ for an arbitrary constant $\alpha > 0$. There is fully-asynchronous distributed algorithm that faithfully simulates the continuous-time chain $(Y_t)_{t\in\mathbb{R}_{\geq 0}}$ for proper q-coloring up to any time $T \geq 0$ within $O(T + \log n)$ time units with high probability.*

Specifically, given any initial coloring $Y_0 \in [q]^V$ and time bound $T \geq 0$, where each node $v \in V$ receives $Y_0(v)$ and T as local input, with high probability the distributed algorithm terminates within $O(T + \log n)$ time units and outputs a random coloring that is identically distributed as Y_T, with each node $v \in V$ outputting its own color $Y_T(v)$. Moreover, every message contains $O(\log n + \log\lceil T\rceil + \log q)$ bits.

Recall that the continuous-time chain Y_T is identically distributed as discrete-time single-site chain X_N for $N \sim \mathrm{Pois}(nT)$. As a straightforward corollary, due to concentration of Poisson distribution, the above theorem also gives a fully-asynchronous distributed algorithm that faithfully simulates N steps of the single-site Metropolis chain $(X_t)_{t\geq 0}$ within $O(N/n + \log n)$ time units. Here, the N/n term gives an optimal linear parallel speedup and the $\log n$ term is also necessary due to anti-concentration for Poisson distribution.

Note that the condition $q \geq \alpha\Delta$ for $\alpha > 0$ in Theorem 1.1 is much weaker than the condition for the chain to be irreducible, which is $q \geq \Delta + 2$, or the necessary condition for the tractability of sampling q-coloring, namely the uniqueness condition $q \geq \Delta + 1$ [GŠV15, Jon02]. The simulation of the chain by the distributed algorithm is efficient even when the chain itself is not mixing.

1.2.2. *General Metropolis samplers.* For general graphical models, we use the following abstract formulation of Metropolis samplers that generalizes Algorithm 1. Let $[q]$ be the finite domain and $G = (V, E)$ the underlying graph of the graphical model. For each $v \in V$, denote by $N_v = N(v)$ the neighborhood of v. A single-site Metropolis chain with state space $\Omega = [q]^V$ is specified by a sequence $(\nu_v)_{v\in V}$ of *proposal distributions* and a sequence $(f^v_{c,c'})_{c,c'\in[q],v\in V}$ of *Metropolis filters*, where each ν_v is a distribution over $[q]$ and each $f^v_{c,c'} : [q]^{N_v} \to [0, 1]$ maps local configurations $X_{N_v} \in [q]^{N_v}$ to acceptance probabilities. The single-site Metropolis chain $(X_t)_{t\geq 0}$ is described abstractly as follows.

Algorithm 3: single-site Metropolis chain (abstract version)

```
Input: initial configuration $X_0 \in [q]^V$
1 for $t = 1$ to $N$ do
2     pick $v \in V$ uniformly at random and denote $c = X_t(v)$;
3     sample $c' \in [q]$ according to $\nu_v$ and construct $X' \in [q]^V$ by modifying $X_t(v)$ to $c'$;
4     with probability $f^v_{c,c'}(X_t(N_v))$, set $X_{t+1} \leftarrow X'$; otherwise, set $X_{t+1} \leftarrow X_t$;
```

The Metropolis chain in Algorithm 1 for sampling from μ represented by a graphical model is a special case of the abstract Metropolis sampler with each ν_v being the uniform distribution over $[q]$ and $f^v_{c,c'} : [q]^{N_v} \to [0, 1]$ defined as $\forall\tau \in [q]^{N_v}$, $f^v_{c,c'}(\tau) = \min\left\{1, \frac{\mu_v(c'|\tau)}{\mu_v(c|\tau)}\right\}$.[1]

And the abstract continuous-time Metropolis chain $(Y_t)_{t\in\mathbb{R}_{\geq 0}}$ is accordingly defined as in Algorithm 2, with the transition replaced by Line 3 and Line 4 in Algorithm 3 with abstract filter $f^v_{c,c'}$.

[1]To have $f^v_{c,c'}$ defined everywhere, we may take the conventions that $\mu_v(\cdot \mid \tau) = 0$ for the illegal τ with 0 measure and also $0/0 = 1$. Such extension of $f^v_{c,c'}$ to total functions will not affect the definition of the chain over legal states.

We define a Lipschitz condition for the Metropolis filters.

Condition 1.2. *There is a constant $C > 0$ such that for any $(u, v) \in E$, any $a, b, c \in [q]$,*

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

where $\Delta = \Delta(G)$ denotes the maximum degree of G, and the operator $\delta_{u,a,b}$ is defined as

$$\delta_{u,a,b}\, f^v_{c,c'} \triangleq \max_{(\sigma,\tau)} \left| f^v_{c,c'}(\sigma) - f^v_{c,c'}(\tau)\right|$$

where the maximum is taken over all such $\sigma, \tau \in [q]^{N_v}$ that $\sigma_u = a$, $\tau_u = b$ and $\sigma = \tau$ elsewhere.

Let $(Y_t)_{t\in\mathbb{R}_{\ge 0}}$ denote the abstract continuous-time Metropolis chain. We show that the chain can be faithfully snd efficiently simulated by distributed algorithms assuming Condition 1.2.

Theorem 1.3 (main theorem)**.** *Assume Condition 1.2. There is a fully-asynchronous distributed algorithm that faithfully simulates (as described specifically in Theorem 1.1) the abstract continuous-time Metropolis chain $(Y_t)_{t\in\mathbb{R}_{\ge 0}}$ up to any time $T \ge 0$ within $O(T + \log n)$ time units with high probability.*

The theorem is implied by a more formal and technical version, Theorem 5.1 in Section 5.

As we argued before, due to the equivalence between continuous-time and discrete-time single-site chains, the distributed algorithm also faithfully simulates N steps of the single-site Metropolis chain $(X_t)_{t\ge 0}$ abstractly defined in Algorithm 3 within $O(N/n + \log n)$ time units. Recall that $\Omega(n \log n)$ is a general lower bound for the mixing time of single-site dynamics [HS07]. Therefore the time bound $(N/n + \log n)$ gives the optimal linear speedup for mixing chains.

The Lipschitz property stated in Condition 1.2 is quite moderate. In well-studied graphical models, this condition asks much less than the respective mixing conditions. Therefore, on these well-studied models, our distributed algorithm can faithfully and efficiently simulate the Metropolis chain even when the chain itself is slow mixing:

- **Proper q-coloring:** Theorem 1.1 is in fact a corollary to Theorem 1.3 on proper q-coloring, where Condition 1.2 translates to $q > \alpha\Delta$ for an arbitrary positive constant α.
- **Hardcore model:** The distribution μ 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, $\mu(\sigma) \propto \lambda^{\sum_{v\in V}\sigma(v)}$, where $\lambda \ge 0$ is the *fugacity*. The natural Metropolis chain for this model is: For each $v \in V$, ν_v is defined over $\{0, 1\}$ as $\nu_v(0) = \frac{1}{1+\lambda}$ and $\nu_v(1) = \frac{\lambda}{1+\lambda}$, and $\forall c, c' \in \{0, 1\}, \tau \in \{0, 1\}^{N_v} : f^v_{c,c'}(\tau) = \prod_{u\in N_v} I[\tau_u + c' \le 1]$. Condition 1.2 means:
$$\exists \text{ constant } C, \quad \lambda < \frac{C}{\Delta},$$
while the uniqueness condition is $\lambda < \frac{(\Delta-1)^{\Delta-1}}{(\Delta-2)^{\Delta}} \approx \frac{\mathrm{e}}{\Delta}$ [Wei06, Sly10].
- **Ising model:** The distribution μ is over all configurations in $\{-1, +1\}^V$, such that for each $\sigma \in \{0, 1\}^V$, $\mu(\sigma) \propto \exp\left(\beta \sum_{(u,v)\in E} \sigma_u\sigma_v\right)$, where $\beta \in \mathbb{R}$ is the temperature. The natural Metropolis chain for this model is: For each $v \in V$, ν_v is uniform over $\{-1, 1\}$, and $\forall c, c' \in \{-1, 1\}, \tau \in \{-1, 1\}^{N_v} : f^v_{c,c'}(\tau) = \exp\left(\min\left\{0, \beta(c' - c)\sum_{u\in N_v} \tau_u\right\}\right)$. Condition 1.2 means:
$$\exists \text{ constant } C, \quad 1 - \mathrm{e}^{-2|\beta|} < \frac{C}{\Delta},$$
while the uniqueness condition is $1 - \mathrm{e}^{-2|\beta|} < \frac{2}{\Delta}$ [SST14, SS14].

2. Model of Asynchrony and Related Work

We adopt the *fully-asynchronous message-passing model* in distributed computing [Lyn96] as our model of asynchrony. The network is described by a simple undirected graph $G = (V, E)$, where the nodes represent processors and the edges represent bidirectional channels between them.

For the asynchronous systems, messages sent from a processor to its neighbor arrive within some finite time, determined by an adversarial scheduler who is adaptive to the entire input. We assume *reliable FIFO channels*: within a channel all messages sent from a processor will eventually be delivered in the order in which they are sent by the processor. Given an execution of an asynchronous algorithm, a *time unit* 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 delivered to the receiver. The time complexity of an algorithm is measured by the number of time units before its termination in the worst case.

The synchronous model is a special case of the asynchronous model where the scheduler is benign and synchronized. Any synchronous algorithm can be transformed to a fully-asynchronous algorithm with essentially the same time complexity by an universal synchronizer [Awe85]. We present our algorithms directly in the asynchronous model without assorting to the synchronizer, because the algorithm is in fact simpler to describe asynchronously as event-driven procedure.

Distributed algorithms for sampling from graphical models have to deal with the concurrency issue as illustrated in Figure 1. Existing approaches that circumvents this fundamental issue include:

- A concurrency control that prevents the concurrent updates to adjacent variables, such as [GLGG11, FSY17], which suffers from a slowdown of $\Delta = \max_v |N(v)|$ factor since within each neighborhood at most one variable can be updated at a time.
- Just ignoring the race condition by obliviously executing the chain, as in the *HogWild!* method [SN10], which introduces a bias to the sampling, where the bias is only known to be controllable assuming a benign asynchronous scheduler assuming independently random message delays [DSOR16, DDJ18], but the bias is uncontrollable against a worst-case adversarial asynchronous scheduler.
- Specially designed parallel Markov chains [FSY17, FG18, FHY18] or non-MCMC distributed sampling algorithms [FVY19]. These algorithms so far require stronger mixing conditions than the rapid mixing of the sequential chain. The best known sufficient condition for these sampling algorithms to be efficient with linear speedup is the path-coupling condition for the original sequential chain.

Previously, there is no distributed algorithm that can sample correctly and efficiently (with linear speedup) from graphical models in the same regime as the rapid mixing sequential Markov chains.

3. Technique Overview: *Resolve Updates in Advance*

Consider the continuous-time chain $(Y_t)_{t\in\mathbb{R}_{\geq 0}}$ for the abstract Metropolis chain described in Algorithm 3, where each node $v \in V$ is associated with an i.i.d. rate-1 Poisson clock and whenever a Poisson clock at a node v rings, $Y(v)$ is updated as Line 3–4 of Algorithm 3.

Our distributed algorithm faithfully simulates the continuous-time chain $(Y_t)_{t\in\mathbb{R}_{\geq 0}}$, such that given any initial state $Y_0 \in [q]^V$ and any $T \geq 0$, the algorithm outputs Y_T when terminates.

```
Algorithm at node $v \in V$:

  Phase I:   Locally generate all update times $0 < t_v^1 < \dots < t_v^{m_v} < T$ and all proposals
             $c_v^1, \dots, c_v^{m_v}$ before time $T$, and send them, along with with $Y_0(v)$, to all neighbors.
             Upon receiving above informations from all neighbors, enter Phase II.

  Phase II:  For $i = 1$ to $m_v$ do:
                   Keep listening to the channels from all neighbors.
             ($\star$) As soon as $v$ gets enough information to resolve its $i$-th update:
                         Resolve the $i$-th update of $v$ and send the update decision
                         ("Accept" or "Reject") to all neighbors.
```

We present the algorithm in two *phases*. In the first phase, the algorithm at each node $v \in V$ locally generates its own random choices and shares them with all its neighbors. Therefore in the beginning of the second phase, every node v knows the initial values, all update times and proposals before time T of itself and all its neighbors. [2]

[2]The two-phase paradigm is just for convenience of exposition. What we actually need is that when an update at node v at time t is resolved, v knows the random choices (proposals and update times) of all neighbors before time t, which can be achieved by streamingly exchanging the random choices between neighbors a few steps ahead.

In the second phase, the algorithm at each node $v \in V$ resolves the updates at v one at a time in Line ($\star$). A straightforward way to implement Line ($\star$) is to resolve an update at node v at time t only when v has known the results of all adjacent updates before time t. Therefore node v can infer the current states $Y_t(N_v)$ of the neighborhood when v is resolving its update at time t. This would introduce a Δ slowdown to the running time, because it does not allow concurrent adjacent updates.

To improve parallel speedup, we must be able to resolve updates at adjacent nodes concurrently as in Figure 1, yet still faithfully execute the Markov chain.

For instance, consider the chain for uniform proper q-coloring. For an update at time t in the chain with proposed color $c' \in [q]$, such an update is resolved once one of the following two events occurs:

- $\forall u \in N_v$, $Y_t(u) \neq c'$, in which case the proposed color c' must be accepted;
- $\exists u \in N_v$, $Y_t(u) = c'$, in which case the proposed color c' must be rejected.

Our key observation is: the update can be resolved *in advance*, before the neighborhood coloring $Y_t(N_v)$ fully known to v. This is because even if for some neighbor u, v only knows the precise color of u up to some earlier time $t_u < t$, v can still infer that the possible color $Y_t(u)$ at time t must be among $Y_{t_u}(u)$ and all proposals of u between time t_u and t. Therefore, the above two event can be determined even before $Y_t(N_v)$ is completely known, once the proposed color c' is not blocked by any of these possible colors $Y_t(u)$ for all neighbors u, or c' is blocked by all possible colors $Y_t(u)$ of a neighbor u.

For general continuous-time Metropolis chain $(Y_t)_{t\in\mathbb{R}_{\geq 0}}$, to resolve an update at time t with proposal $c' \in [q]$, the algorithm at node v needs to flip a coin with bias $f^v_{c,c'}(Y_t(N_v))$, where $c \triangleq Y_{t-\epsilon}(v)$ denotes the current value of v right before the update. As before, node v does not necessarily know the exact neighborhood configuration $Y_t(N_v)$ at the moment when trying to resolve its own update at time t. For every neighbor u of v, suppose that node v has received messages from u regarding its update results up to time t_u. If $t_u \geq t$, then v can infer the exact value of $Y_t(u)$; but if $t_u < t$, then v does not know the exact value of $Y_t(u)$, however, as we observed above, even in such pessimistic case, due to the definition of Metropolis chain, v can still infer that $Y_t(u)$ must be among u's current value $Y_{t_u}(u)$ and all proposals of u between time t_u and t. Therefore, v can maintain a (product) space of possible neighborhood configurations $Y_t(N_v)$, and as more messages received by v from its neighbors, this space of possible $Y_t(N_v)$ is getting smaller and smaller, until finally the exact $Y_t(N_v)$ is fully known. Moreover, v can deduce the current upper and lower bounds of the bias $f^v_{c,c'}(Y_t(N_v))$.

Therefore, to resolve the update in advance, our deal is to flip a coin with unknown bias $f^v_{c,c'}(Y_t(N_v))$ as early as possible, while at times only the upper and lower bounds of the bias $f^v_{c,c'}(Y_t(N_v))$ is known (and as more messages received, these upper and lower bounds are getting closer to each other until finally collapse). This can be achieved by sampling a uniform real β_v from interval $[0, 1)$, monitoring the current upper and lower bounds of the bias $f^v_{c,c'}(Y_t(N_v))$, and accepting the update once β_v lies below the current lower bound of $f^v_{c,c'}(Y_t(N_v))$, or rejecting the update once β_v lies above the current upper bound of $f^v_{c,c'}(Y_t(N_v))$. Details of the algorithm are formally described in next section.

4. Distributed Simulation of Continuous-Time Chains

We now formally describe the **main algorithm** outlined in last section that simulates the continuous-time chain $(Y_t)_{t\in\mathbb{R}_{\geq 0}}$ up to time T in the asynchronous communication model. Fix any node $v \in V$. The algorithm at node v consists of two phases. In **Phase I**, node v locally generates these random values:

- the random update times $0 < t^1_v < \cdots < t^{m_v}_v < T$, generated independently by a rate-1 Poisson clock, where $m_v \sim \mathrm{Pois}(T)$ is the number of times the clock rings before time T.
- the random proposals $c^1_v, c^2_v, \ldots, c^{m_v}_v \in [q]$ sampled i.i.d. according to distribution ν_v.

Then node v sends these random values and its initial value $Y_0(v)$ to all its neighbors. The update times t^i_v can be represented with bounded precision as long as being distinct and preserving relative order among adjacent nodes, which is enough for correctly simulating the chain $(Y_t)_{t\in[0,T]}$. This can be done within $O(T+\log n)$ time unites with high probability, with each message of size $O(\log n+\log\lceil T\rceil+\log q)$.[3] Once node v receives all such informations from all its neighbors, it enters its own **Phase II**.

[3]The update times t^i_v are truncated adaptively with bounded precision when sent to neighbors, ensuring that among neighbors the truncated update times are distinct and having the same relative order as before truncation.

With the random update times and proposals $\left(t_v^i, c_v^i\right)$, the continuous-time chain $(Y_t)_{t\in[0,T]}$ can be generated starting from the initial configuration $Y_0 \in [q]^V$, where at each update time $t = t_v^i$ for some $v \in V$ and $1 \le i \le m_v$:

$$Y_t(v) = \begin{cases} c_v^i & \text{with prob. } f^v_{c,c'}(Y_t(N_v)) \\ Y_{t-\epsilon}(v) & \text{with prob. } 1 - f^v_{c,c'}(Y_t(N_v)) \end{cases}, \quad \text{where } c' = c_v^i \text{ and } c = Y_{t-\epsilon}(v). \tag{1}$$

Here we use $Y_{t-\epsilon}(v)$ to denote the value of $Y(v)$ right before the update at time t. And in general for any $0 < t \le T$ and $v \in V$: $Y_t(v) = Y_{t_v^i}(v)$ for the i satisfying $t_v^i \le t < t_v^{i+1}$. Furthermore, for any $v \in V$ and $0 \le i \le m_v$ we denote the state of v right after its i-th update as:

$$Y_v^{(i)} \triangleq Y_{t_v^i}(v). \tag{2}$$

The **Phase II** of the algorithm at node v is described in Algorithm 4.

```
Algorithm 4: Pseudocode for Phase II of the main algorithm at node v
  Assumption: node $v \in V$ knows the initial values $Y_0(u)$ and the lists of update times and
              proposals $\left(t_u^i, c_u^i\right)_{1\le i\le m_u}$ of all $u \in N_v \cup \{v\}$.
1 Initialize $\widehat{Y}_v^{(0)}$, $\mathbf{j} = (j_u)_{u\in N_v}$ and $\mathbf{Y} = \left(\widehat{Y}_u^{(j)}\right)_{u\in N_v, 0\le j\le j_u-1}$ respectively as:
2     $\widehat{Y}_v^{(0)} \leftarrow Y_0(v)$ and for all $u \in N_v$: $j_u \leftarrow 1$ and $\widehat{Y}_u^{(0)} \leftarrow Y_0(u)$;
3 for $i = 1$ to $m_v$ do
4     $\left(\widehat{Y}_v^{(i)}, \mathbf{j}, \mathbf{Y}\right) \leftarrow$ Resolve $\left(i, \widehat{Y}_v^{(i-1)}, \mathbf{j}, \mathbf{Y}\right)$    ▷ resolve the $i$-th update
5 return $\widehat{Y}_v^{(m_v)}$;
```

Remark 4.1 (synchronization between phases). Different nodes may enter Phase II at different time due to asynchrony. There is no message ever sent from a Phase-I node to a Phase-II node. 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 in which they are received once v has entered Phase II.

The goal of the algorithm at node v is to output the value $Y_v^{(m_v)} = Y_T(v)$. To achieve this goal, it resolves all m_v updates of v one by one in the order in which they are proposed. In the i-th iteration, node v has resolved the first $i-1$ updates and maintains its current state as $\widehat{Y}_v^{(i-1)}$; and within the iteration, it resolves the i-th update and updates its current state $\widehat{Y}_v^{(i-1)}$ to a new $\widehat{Y}_v^{(i)}$. This naturally defines a continuous-time chain $(\widehat{Y}_t)_{t\in[0,T]}$:

$$\forall 0 \le t \le T \text{ and } \forall v \in V: \quad \widehat{Y}_t(v) = \widehat{Y}_v^{(i)} \text{ where } i \text{ satisfies } t_v^i \le t < t_v^{i+1}. \tag{3}$$

In Algorithm 4, the current state $\widehat{Y}_v^{(i)}$ of node v is updated with the assistance of the following auxiliary data structures maintained at node v:

- a tuple $\mathbf{j} = (j_u)_{u\in N_v}$, such that each j_u denotes that from v's perspective, the neighbor u is resolving its j_u-th update, and the outcomes of its first $(j_u - 1)$ updates are known to v;
- a tuple $\mathbf{Y} = \left(\widehat{Y}_u^{(j)}\right)_{u\in N_v, 0\le j\le j_u-1}$, such that each $\widehat{Y}_u^{(j)}$ memorizes for v the correct state of u in the algorithm right after resolving its j-th update.

Given any such $\mathbf{j} = (j_u)_{u\in N_v}$ and $\mathbf{Y} = \left(\widehat{Y}_u^{(j)}\right)_{u\in N_v, 0\le j\le j_u-1}$, for any neighbor $u \in N_v$ and any time $t < t_u^{j_u}$, due to (3), node v can infer the precise value of $\widehat{Y}_t(u)$ as: $\widehat{Y}_t(u) = \widehat{Y}_u^{(k-1)}$ where k satisfies $t_u^{k-1} \le t < t_u^k$. For the time $t \ge t_u^{j_u}$, node v can no longer always infer the exact value of $\widehat{Y}_t(u)$ for a neighbor $u \in N_v$ based on the partial information encoded by $\mathbf{j}$ and $\mathbf{Y}$. However, it can narrow down the possible values of $\widehat{Y}_t(u)$ to a subset of $[q]$.

Definition 4.2 (set of possible states). Fix a node $v \in V$ and given its current $\mathbf{j} = (j_u)_{u \in N(v)}$ and $\mathrm{Y} = \left(\widehat{Y}_u^{(j)}\right)_{u \in N_v, 0 \le j \le j_u - 1}$, for any neighbor $u \in N_v$ and any time $0 \le t < T$, the set of possible states $\mathcal{S}_t(u) \subseteq [q]$ for node u at time t with respect to $\mathbf{j}$ and Y is defined as

$$\mathcal{S}_t(u) \triangleq \begin{cases} \left\{ \widehat{Y}_u^{(k-1)} \right\} \text{ where } t_u^{k-1} \le t < t_u^k & \text{if } 0 \le t < t_u^{j_u} \\ \left\{ \widehat{Y}_u^{(j_u - 1)} \right\} \cup \left\{ c_u^k \mid t_u^{j_u} \le t_u^k \le t \right\} & \text{if } t_u^{j_u} \le t < T. \end{cases} \tag{4}$$

For Metropolis chains, it holds for each transition that $\widehat{Y}_u^{(j)} \in \{c_u^j, \widehat{Y}_u^{(j-1)}\}$, i.e. the state of u is either changed to the proposal or unchanged. It is then easy to verify that $\widehat{Y}_t(u) \in \mathcal{S}_t(u)$ always holds for any $u \in N_v$ any $0 \le t < T$. This guarantees the soundness of the definition. Furthermore, node v can locally compute sets $\mathcal{S}_t(u)$ for all $u \in N_v$ and $0 \le t < T$ based on its current $\mathbf{j}$ and Y.

Next, we show how to use the partial information of $Y_t(u)$ captured by $\mathcal{S}_t(u)$ to resolve an update with filter $f_{c,c'}^v(Y_t(N_v))$ before $Y_t(N_v)$ is fully known.

4.1. **Example: Proper Graph Coloring.** We first give the subroutine Resolve on a special case: the uniform proper q-coloring of graph $G = (V, E)$. To be distinguished from the general case, we use Resolve-Coloring$(i, \widehat{Y}(v), \mathbf{j}, \mathrm{Y})$ to denote the subroutine for this special case, described in Algorithm 5.

```
Algorithm 5: Resolve-Coloring$(i, \widehat{Y}(v), \mathbf{j}, \mathrm{Y})$ at node $v$
  input: index $i$ of the current update; current color $\widehat{Y}(v)$; the tuples $\mathbf{j} = (j_u)_{u \in N_v}$ and
         $\mathrm{Y} = \left(\widehat{Y}_u^{(j)}\right)_{u \in N_v, 0 \le j \le j_u - 1}$ of the current steps and historical colors of $v$'s neighbors;
1  construct $\mathcal{S}(u) = \mathcal{S}_{t_v^i}(u)$ as (4) for all $u \in N_v$;
2  upon $c_v^i \notin \bigcup_{u \in N_v} \mathcal{S}(u)$ do
3      send message "ACCEPT" to all neighbors $u \in N_v$;
4      return $(c_v^i, \mathbf{j}, \mathrm{Y})$;
5  upon $\exists u \in N_v$ s.t. $\mathcal{S}(u) = \{ c_v^i \}$ do
6      send message "REJECT" to all neighbors $u \in N_v$;
7      return $\left(\widehat{Y}(v), \mathbf{j}, \mathrm{Y}\right)$;
8  upon receiving "ACCEPT" from a neighbor $u \in N_v$ do
9      modify Y by setting $\widehat{Y}_u^{(j_u)} = c_u^{j_u}$;
10     $j_u \leftarrow j_u + 1$;
11     recompute $\mathcal{S}(u) = \mathcal{S}_{t_v^i}(u)$ as (4) w.r.t. new $\mathbf{j}$ and Y;
12 upon receiving "REJECT" from a neighbor $u \in N_v$ do
13     modify Y by setting $\widehat{Y}_u^{(j_u)} = \widehat{Y}_u^{(j_u - 1)}$;
14     $j_u \leftarrow j_u + 1$;
15     recompute $\mathcal{S}(u) = \mathcal{S}_{t_v^i}(u)$ as (4) w.r.t. new $\mathbf{j}$ and Y;
```

Algorithm 5 resolves the i-th update $\left(t_v^i, c_v^i\right)$ for proper coloring. Node v maintains for each neighbor $u \in N_v$ a set $\mathcal{S}(u) = \mathcal{S}_{t_v^i}(u)$ of possible colors of u at time t_v^i based on the $\mathbf{j}$ and Y maintained by v. The algorithm is event-driven: if more than one events occur simultaneously, they are responded in the order listed in the algorithm. Node v is constantly monitoring the sets $\mathcal{S}(u)$ for all $u \in N_v$. The update $\left(t_v^i, c_v^i\right)$ is resolved once one of the following two events occurs:

- $c_v^i \notin \bigcup_{u \in N_v} \mathcal{S}(u)$, in which case the proposed color c_v^i is determined to not blocked by any possible color of any neighbor at time t_v^i, thus c_v^i must be accepted;
- $\exists u \in N_v$ s.t. $\mathcal{S}(u) = \left\{c_v^i\right\}$, in which case the proposed color c_v^i is determined to be blocked by neighbor u's color at time t_v^i, thus c_v^i must be rejected.

These two events are mutually exclusive. Eventually, at least one of the above two events must occur so that the algorithm must terminate.

4.2. **General Metropolis Chain.** We then give the subroutine Resolve$(i, \widehat{Y}(v), \mathbf{j}, \mathbf{Y})$ for resolving the i-th update $\left(t_v^i, c_v^i\right)$ at node v in a general Metropolis chain. Throughout, we denote the current and proposed states of v respectively as:

$$c \triangleq \widehat{Y}(v) \text{ and } c' \triangleq c_v^i.$$

The algorithm then simulates a coin flipping with bias $f_{c,c'}^v\left(\widehat{Y}_{t_v^i}(N_v)\right)$, to determine whether the proposal c_v^i is accepted, while the configuration $\widehat{Y}_{t_v^i}(N_v)$ is only partially known to node v.

In particular, given the current $\mathbf{j}$ and $\mathbf{Y}$ (as inputs to the subroutine Resolve), the set $\mathcal{S}_{t_v^i}(u)$ of possible states of each neighbor $u \in N_v$ at the time t_v^i, can be constructed as Definition 4.2. We further define the set of all possible configurations on the neighborhood N_v as

$$C_v^i \triangleq \bigotimes_{u \in N_v} \mathcal{S}_{t_v^i}(u). \tag{5}$$

Note that C_v^i must contain the correct configuration $\widehat{Y}_{t_v^i}(N_v)$ because $\widehat{Y}_{t_v^i}(u) \in \mathcal{S}_{t_v^i}(u)$ for every $u \in N_v$ due to the soundness of Definition 4.2. Besides, C_v^i may contain various other candidate configurations $\tau \in [q]^V$, each corresponding to a coin with bias $f_{c,c'}^v(\tau)$. The subroutine resolves the update in advance and correctly by coupling all these coins.

Specifically, for each $\tau \in C_v^i$, we define the indicator random variable $I_{\mathsf{AC}}(\tau) \in \{0,1\}$ as $\Pr[\, I_{\mathsf{AC}}(\tau) = 1\,] = f_{c,c'}^v(\tau)$. The coins $I_{\mathsf{AC}}(\tau)$ for all $\tau \in C_v^i$ are coupled as follows:

$$I_{\mathsf{AC}}(\tau) = \begin{cases} 1 & \text{if } \beta < f_{c,c'}^v(\tau) \\ 0 & \text{if } \beta \geq f_{c,c'}^v(\tau) \end{cases}, \qquad \beta \text{ is uniformly distributed over } [0,1).$$

Since the true $\widehat{Y}_{t_v^i}(N_v) \in C_v^i$, the update can be resolved once all the indicator random variables $I_{\mathsf{AC}}(\tau)$ for $\tau \in C_{t(v,i)}$ are perfectly coupled, i.e. $\forall \tau_1, \tau_2 \in C_v^i : I_{\mathsf{AC}}(\tau_1) = I_{\mathsf{AC}}(\tau_2)$.

To check whether all the indicator random variables are perfectly coupled, we define the minimum acceptance probability P_{AC} and the minimum rejection probability P_{RE} as:

$$\begin{aligned} P_{\mathsf{AC}} &\triangleq \min_{\tau \in C_v^i} f_{c,c'}^v(\tau); \\ P_{\mathsf{RE}} &\triangleq \min_{\tau \in C_v^i} \left(1 - f_{c,c'}^v(\tau)\right) = 1 - \max_{\tau \in C_v^i} f_{c,c'}^v(\tau). \end{aligned} \tag{6}$$

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

The procedure for Resolve$(i, \widehat{Y}(v), \mathbf{j}, \mathbf{Y})$ at node v is described in Algorithm 6. In the algorithm, a random number $\beta \in [0,1)$ is sampled only once (hence the coupling) in the beginning and used during the entire execution of the subroutine Resolve$(i, \widehat{Y}(v), \mathbf{j}, \mathbf{Y})$ for resolving the i-th update at node v. The algorithm is event-driven: if more than one events occur simultaneously, they are responded in the order listed in the algorithm. The i-th proposed update $\left(t_v^i, c_v^i\right)$ at node v is accepted once $\beta < P_{\mathsf{AC}}$ and is rejected once $\beta \geq 1 - P_{\mathsf{RE}}$. These two events are mutually exclusive because $P_{\mathsf{AC}} + P_{\mathsf{RE}} \leq 1$. The two thresholds $P_{\mathsf{AC}}, P_{\mathsf{RE}} \in [0,1]$ are updated dynamically once $\mathbf{j}$ and $\mathbf{Y}$ updated correctly upon a message "Accept" or "Reject" received from a neighbor $u \in N_v$. Eventually at least one of the two events $\beta < P_{\mathsf{AC}}$ and $\beta \geq 1 - P_{\mathsf{RE}}$ must occur when $|C_v^i| = 1$, i.e. when the correct configuration $\widehat{Y}_{t_v^i}(N_v)$ is fully known to v, because in this case $P_{\mathsf{AC}} + P_{\mathsf{RE}} = 1$.

Remark 4.3 (out-of-order resolution). The algorithm at each node v resolves the updates in the order of the update times $0 < t_v^1 < t_v^2 < \cdots < t_v^{m_v} < T$. Alternatively, the algorithm can resolve any update once one of the conditions for that update in Line 3 and Line 6 in Algorithm 6 is triggered, disregarding the order of the update time. By coupling and a monotone argument, it is easy to verify that such modified algorithm with out-of-order resolution of updates is also correct and is at least as fast as the original algorithm with in-order resolution.

Algorithm 6: Resolve($i, \widehat{Y}(v), \mathbf{j}, \mathbf{Y}$) at node v

```
input: index $i$ of the current update; current value $\widehat{Y}(v)$; the tuples $\mathbf{j} = (j_u)_{u \in N_v}$ and
       $\mathbf{Y} = \left(\widehat{Y}_u^{(j)}\right)_{u \in N_v, 0 \le j \le j_u - 1}$ of the current steps and historical values of $v$'s neighbors;
1  sample $\beta \in [0, 1)$ uniformly at random;
2  compute $P_{\mathsf{AC}}$ and $P_{\mathsf{RE}}$ as (6), where $\mathcal{S}_{t_v^i}(u)$ for each $u \in N_v$ is constructed as (4);
3  upon $\beta < P_{\mathsf{AC}}$ do
4      send message "Accept" to all neighbors $u \in N_v$;
5      return $(c_v^i, \mathbf{j}, \mathbf{Y})$;
6  upon $\beta \ge 1 - P_{\mathsf{RE}}$ do
7      send message "Reject" to all neighbors $u \in N_v$;
8      return $\left(\widehat{Y}(v), \mathbf{j}, \mathbf{Y}\right)$;
9  upon receiving "Accept" from a neighbor $u \in N_v$ do
10     modify $\mathbf{Y}$ by setting $\widehat{Y}_u^{(j_u)} = c_u^{j_u}$;
11     $j_u \leftarrow j_u + 1$;
12     recompute $P_{\mathsf{AC}}$ and $P_{\mathsf{RE}}$ as (6) with $\mathcal{S}_{t_v^i}(u)$ reconstructed as (4) w.r.t. new $\mathbf{j}$ and $\mathbf{Y}$;
13 upon receiving "Reject" from a neighbor $u \in N_v$ do
14     modify $\mathbf{Y}$ by setting $\widehat{Y}_u^{(j_u)} = \widehat{Y}_u^{(j_u-1)}$;
15     $j_u \leftarrow j_u + 1$;
16     recompute $P_{\mathsf{AC}}$ and $P_{\mathsf{RE}}$ as (6) with $\mathcal{S}_{t_v^i}(u)$ reconstructed as (4) w.r.t. new $\mathbf{j}$ and $\mathbf{Y}$;
```

Computation costs. The cost for local computation is dominated by the cost 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 1.2. For such graphical models, the Metropolis filter $f_{c,c'}^v(\tau)$ can be written as:

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

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

$$P_{\mathsf{AC}} = \min\left\{1, \prod_{u \in N_v}\left(\min_{b \in \mathcal{S}(u)} f_{c,c'}^{v,u}(b)\right)\right\}, \quad P_{\mathsf{RE}} = 1 - \min\left\{1, \prod_{u \in N_v}\left(\max_{b \in \mathcal{S}(u)} f_{c,c'}^{v,u}(b)\right)\right\},$$

where $c = \widehat{Y}(v)$ and $c' = c_v^i$ as before and for each neighbor $u \in N_v$, the set $\mathcal{S}(u) = \mathcal{S}_{t_v^i}(u)$ is easy to compute locally based on the current $\mathbf{j}$ and $\mathbf{Y}$ as in (4).

5. Analysis of the Algorithm

Recall the original continuous-time chain $(Y_t)_{t \in [0,T]}$ defined in (1), and the continuous-time chain $(\widehat{Y}_t)_{t \in [0,T]}$ generated by the algorithm, defined in (3).

Theorem 5.1 (restatement of Theorem 1.3). *Let $\zeta \ge 1$ be an arbitrary constant and $n = |V|$. The followings hold for the main algorithm.*

(1) (correctness) Upon termination, the algorithm outputs a random $\widehat{Y}_T \in [q]^V$, with each $v \in V$ outputting $\widehat{Y}_T(v)$, such that $\widehat{Y}_T$ is identically distributed as Y_T.

(2) (complexity) With probability at least $1 - n^{-\zeta}$, the algorithm terminates within $20\zeta(\Delta T + \log n)$ time units, where Δ denotes the maximum degree of G.

If Condition 1.2 holds with constant C, then with probability at least $1 - n^{-\zeta}$, the algorithm terminates within $20(\zeta + C)(T + \log n)$ time units.

Moreover, each message contains $10\zeta(\log n + \log\lceil T\rceil + \log q)$ bits.

Theorem 1.1 is implied by Theorem 1.3 as a special case on proper graph coloring by setting every proposal distribution ν_v as the uniform distribution over $[q]$ and every Metropolis filter $f^v_{c,c'} : [q]^{N_v} \to [0,1]$ as $\forall \tau \in [q]^{N_v} : f^v_{c,c'}(\tau) = \prod_{u \in N(v)} I[\tau_u \neq c']$.

5.1. Proof Outline.

The correctness part of Theorem 5.1 is proved by a perfect coupling between the continuous-time chain $(\widehat{Y}_t)_{t\in[0,T]}$ generated by the algorithm and the original continuous-time chain $(Y_t)_{t\in[0,T]}$. Both chains are determined by the following randomness: the random update times $\{t^i_v\}$, the random proposals $\{c^i_v\}$, and the random $\{\beta^i_v\}$ which are sampled independently at Line 1 in Algorithm 6 when resolving the i-th update at node v. The two chains $(\widehat{Y}_t)_{t\in[0,T]}$ and $(Y_t)_{t\in[0,T]}$ are coupled by using the same randomness. The identity between $(\widehat{Y}_t)_{t\in[0,T]}$ and $(Y_t)_{t\in[0,T]}$ can then be verified by induction. The proof is in Section 5.2.

Note that each message of the main algorithm is of at most $O(\log n + \log\lceil T\rceil + \log q)$ bits, and due to the concentration for Poisson distribution with high probability **Phase I** lasts for at most $O(T + \log n)$ time units. The complexity of the algorithm is determined by the length of **Phase II**.

The $O(\Delta T + \log n)$ suboptimal time upper bound is easy to obtain. Observe that if a node v resolving an update at time t does not terminate after $\ell \geq 1$ time units, then there must exist an update at adjacent node $u \in N(v) \cup \{v\}$ at some time $t' < t$, otherwise v can instantly resolve its update without waiting. Therefore, if a node v does not terminate after $\ell \geq 1$ time units, then there exists a path $v_1, v_2, \ldots, v_\ell$ with $v_\ell = v$ and $v_i \in N(v_{i-1}) \cup \{v_{i-1}\}$ for all $1 < i \leq \ell$ and a sequence of update times $0 < t_1 < t_2 < \cdots < t_\ell < T$ such that t_i is the time for an update at node v_i for all $1 \leq i \leq \ell$. Fix any such path $v_1, v_2, \ldots, v_\ell$. Note that $0 < t_1 < t_2 < \cdots < t_\ell < T$ are generated by independent Poisson clock. The probability that a Poisson clock rings ℓ times before time T is bounded by $(eT/\ell)^\ell$. Taking the union bound over at most $n(\Delta+1)^\ell$ such paths $v_1, v_2, \ldots, v_\ell$ gives the $n(\Delta+1)^\ell(eT/\ell)^\ell$ upper bound on the probability that the algorithm does not terminate after ℓ time units. Therefore, $\ell = O(\Delta T + \log n)$ with high probability. Similar analysis has been applied for analyzing distributed algorithms, such as in [NO08].

The analysis is getting substantially more sophisticated for the $O(T + \log n)$ optimal time bound.

We use the pair (v, i) to identify the i-th update of node v. In Algorithm 6, the resolution of each update (v, i) is triggered by one of the two *resolution conditions* ($\beta < P_{\mathsf{AC}}$ and $\beta \geq 1 - P_{\mathsf{RE}}$) respectively at Line 3 and Line 6. This can be further divided into to cases:

- **self-triggered resolution:** a resolution condition is triggered by computing P_{AC} and P_{RE} for the first time in Line 2 of Algorithm 6;
- **resolution triggered by an adjacent update (u, j):** a resolution condition is triggered by the recomputing of P_{AC} and P_{RE} upon processing the message "Accept" or "Reject" from a neighbor $u \in N_v$ that indicates the result for u resolving its j-th update (u, j).

Fixed all randomnesses $\{t^i_v\}$, $\{c^i_v\}$ and $\{\beta^i_v\}$, given an update (v, i), we use $\mathcal{D}_{(v,i)}$ to denote the *dependency chain that ends at* (v, i). The dependency chain $\mathcal{D}_{(v,i)}$ is a sequence of updates $(v_1, i_1), (v_2, i_2), \ldots, (v_\ell, i_\ell)$ where $(v_\ell, i_\ell) = (v, i)$, which is recursively constructed as:

$$\mathcal{D}_{(v,i)} \triangleq \begin{cases} (v,i) & \text{if resolution of } (v,i) \text{ is self-triggered and } i = 1, \\ \mathcal{D}_{(v,i-1)}, (v,i) & \text{if resolution of } (v,i) \text{ is self-triggered and } i > 1, \\ \mathcal{D}_{(u,j)}, (v,i) & \text{if resolution of } (v,i) \text{ is triggered by adjacent update } (u,j); \end{cases} \tag{7}$$

Clearly such a dependency chain $\mathcal{D}_{(v,i)}$ is uniquely constructed once all randomnesses are fixed. A key observation is that the number of time units **Phase II** costs is always bounded by the length of the longest dependency chain. And assuming Condition 1.2, with high probability the length of the longest dependency chain is at most $O(T + \log n)$. The detailed analysis is in Section 5.3.

5.2. Proof of Correctness.

Recall that the algorithm defines a chain $(\widehat{Y}_t)_{t\in[0,T]}$ (formally defined in (3)) and outputs $\widehat{Y}_T$ when terminates. Now we show that this chain produced by the algorithm perfectly simulates the original continuous-time Metropolis chain $(Y_t)_{t\in[0,T]}$ (defined in (1)).

Lemma 5.2. *With probability 1 the algorithm terminates and $\widehat{Y}_T$ is identically distributed as Y_T.*

Both processes $(Y_t)_{t\in[0,T]}$ and $(\widehat{Y}_t)_{t\in[0,T]}$ are fully determined by the following randomnesses:

$$\begin{aligned} &m_v \text{ for each } v \in V, \\ &(t_v^i, c_v^i, \beta_v^i) \text{ for each } v \in V \text{ and every } 1 \le i \le m_v, \end{aligned} \tag{8}$$

where m_v denotes the number of times the Poisson clock at node v rings, t_v^i and c_v^i denote respectively the time and proposal of the i-th update at node v, and $\beta_v^i \in [0,1]$ denotes the randomness used for the coin flipping in the i-th update of node v: in particular, in $(Y_t)_{t\in[0,T]}$, the random value $\beta_v^i \in [0,1]$ is used in (1) to determine the experiment of the Metropolis filter of the i-th update at node v, so that (1) is implemented as

$$Y_t(v) = \begin{cases} c_v^i & \beta_v^i \le f_{c,c'}^v(Y_t(N_v)) \\ Y_{t-\epsilon}(v) & \beta_v^i > f_{c,c'}^v(Y_t(N_v)) \end{cases}, \quad \text{where } c' = c_v^i \text{ and } c = Y_{t-\epsilon}(v);$$

and in $(\widehat{Y}_t)_{t\in[0,T]}$, the random value β_v^i is just the $\beta \in [0,1]$ in Resolve$(i, \widehat{Y}(v), \mathbf{j}, \mathbf{Y})$ (Algorithm 6) for resolving the i-th update at node v.

The perfect coupling: We couple the two processes $(Y_t)_{t\in[0,T]}$ and $(\widehat{Y}_t)_{t\in[0,T]}$ by using the same randomnesses $(m_v)_{v\in V}$ and $(t_v^i, c_v^i, \beta_v^i)_{v\in V, 1\le i\le m_v}$.

We use the pair (v,i), where $v \in V$ and $1 \le i \le m_v$, to identify the i-th update at node v. Fixed any collection of randomnesses as (8) such that all update times are distinct, a partial order $\prec$ on updates (v,i) can be naturally defined as:

$$\forall v \in V, u \in N_v \cup \{v\}, 1 \le i \le m_u, 1 \le j \le m_v: \quad (u,i) \prec (v,j) \iff t_u^i < t_v^j. \tag{9}$$

Recall that $Y_v^{(i)}$ (defined in (2)) denotes the state of v in the chain $(Y_t)_{t\in[0,T]}$ right after the update (v,i), and $\widehat{Y}_v^{(i)}$ (constructed in Algorithm 4) denotes the state of v in the algorithm after resolving the update (v,i).

Assume that all m_v's are finite and any pair of update times t_u^i and t_v^j for $(u,i) \ne (v,j)$ are distinct by a finite gap. We then apply a structural induction to prove the following hypothesis:

Claim 5.3. *For every $v \in V$ and every $1 \le i \le m_v$, $\widehat{Y}_v^{(i)}$ is well-defined and $\widehat{Y}_v^{(i)} = Y_v^{(i)}$.*

Note that this proves Lemma 5.2, because it implies that as long as all m_v's are finite and all update times are distinct (which occurs with probability 1), $\widehat{Y}_T$ is well-defined, where $\widehat{Y}_T(v) = \widehat{Y}_v^{(m_v)}$ for every $v \in V$ by (3), thus the algorithm terminates; and $\widehat{Y}_T$ is identically distributed as Y_T, where Y_T is constructed as $Y_T(v) = Y_v^{(m_v)}$ for every $v \in V$ by (2).

It then remains to verify Claim 5.3 by a structural induction on the partial order in (9).

In the **main algorithm**, each update time is represented with bounded precision. After the truncation of precision, suppose each update time t_u^i becomes $\widetilde{t}_u^i$. The following property holds for truncated update times.

$$\forall v \in V, u \in N_v \cup \{v\}, 1 \le i \le m_u, 1 \le j \le m_v: \widetilde{t}_u^i < \widetilde{t}_v^j \iff t_u^i < t_v^j. \tag{10}$$

Hence, the partial order $\prec$ defined in (9) is preserved after the truncation:

$$(u,i) \prec (v,j) \iff \widetilde{t}_u^i < \widetilde{t}_v^j. \tag{11}$$

Induction basis: We say the update (v,j) is a minimal element with respect to partial order $\prec$ if and only if there is no update (u,i) such that $(u,i) \prec (v,j)$. We prove that for each minimal element (v,j), it holds that $\widehat{Y}_v^{(j)}$ is well-defined and $\widehat{Y}_v^{(j)} = Y_v^{(j)}$. Since (v,j) is a minimal element, it must hold that $j = 1$, $\widetilde{t}_u^i > \widetilde{t}_v^j$ and $t_u^i > t_v^j$ for all $u \in N_v$ and $1 \le i \le m_u$. In **Phase II**, node v first tries to resolve the update (v,j) and constructs the set $\mathcal{S}_{\widetilde{t}_v^j}(u) = \{\widehat{Y}_u^{(0)}\}$ for each $u \in N_v$ by (4). Hence, once v receives all $\widehat{Y}_u^{(0)}$ for $u \in N_v$, the update (v,j) must be resolved. This implies $\widehat{Y}_v^{(j)}$ is well-defined. Note that (v,j) is a minimal element with respect to partial order $\prec$. This implies $j = 1$, $Y_{t_v^j}(N_v) = Y_0(N_v)$ and

$\widehat{Y}_{\widetilde{t}_v^j}(N_v) = \widehat{Y}_0(N_v)$. In continuous-time Metropolis chain, it holds that $Y_v^{(j)} = c_v^j$ if $\beta_v^j < f_{c,c'}^v(Y_0(N_v))$; $Y_v^{(j)} = Y_v^{(0)}$ if $\beta_v^j \geq f_{c,c'}^v(Y_0(N_v))$, where $c = Y_v^{(0)}$ and $c' = c_v^j$. In **main algorithm**, the set C_v^j defined in (5) is $\{\widehat{Y}_0(N_v)\}$ and it holds that $\widehat{Y}_v^{(j)} = c_v^j$ if $\beta_v^j < f_{c,c'}^v(\widehat{Y}_0(N_v))$; $\widehat{Y}_v^{(j)} = \widehat{Y}_v^{(0)}$ if $\beta_v^j \geq f_{c,c'}^v(\widehat{Y}_0(N_v))$, where $c = \widehat{Y}_v^{(0)}$ and $c' = c_v^j$. Since $\widehat{Y}_u^{(0)} = Y_u^{(0)}$ for all $u \in V$, then $\widehat{Y}_v^{(j)} = Y_v^{(j)}$.

Induction step: Fix an update (v,j). By induction hypothesis, for all updates (u,i) with $(u,i) \prec (v,j)$, it holds that $\widehat{Y}_u^{(i)}$ is well defined and $\widehat{Y}_u^{(i)} = Y_u^{(i)}$. For all $u \in N_v$ and $1 \leq i \leq m_u$ with $(u,i) \prec (v,j)$, since $\widehat{Y}_u^{(i)}$ is well-defined, then in **main algorithm**, node u must resolve the update (u,i) and sends the message "Accept" or "Reject" to node v. Hence, node v will eventually resolve the update (v,j) no later than the moment at which all these messages are delivered, because in that moment, node v knows the full information of $Y_{\widetilde{t}_v^j}(N_v)$ and $P_{\mathsf{AC}} + P_{\mathsf{RE}} = 1$. This proves that $\widehat{Y}_v^{(j)}$ is well defined.

Consider the continuous-time Metropolis chain. It holds that

$$Y_v^{(j)} = \begin{cases} c_v^j & \text{if } \beta_v^j < f_{c,c'}^v(Y_{t_v^j}(N_v)) \text{ where } c = Y_v^{(j-1)}, c' = c_v^j \\ Y_v^{(j-1)} & \text{if } \beta_v^j \geq f_{c,c'}^v(Y_{t_v^j}(N_v)) \text{ where } c = Y_v^{(j-1)}, c' = c_v^j. \end{cases} \tag{12}$$

Consider the moment at which v resolves the update (v,j). In Algorithm 6, node v computes the set $\mathcal{S}_{\widetilde{t}_v^j}(u)$ by (4), formally,

$$\forall u \in N_v : \quad \mathcal{S}_{\widetilde{t}_v^j}(u) \triangleq \begin{cases} \left\{ \widehat{Y}_u^{(k-1)} \right\} \text{ where } \widetilde{t}_u^{k-1} \leq \widetilde{t}_v^j < \widetilde{t}_u^k & \text{if } 0 \leq \widetilde{t}_v^j < \widetilde{t}_u^{j_u} \\ \left\{ \widehat{Y}_u^{(j_u-1)} \right\} \cup \left\{ c_u^k \mid \widetilde{t}_u^{j_u} \leq \widetilde{t}_u^k \leq \widetilde{t}_v^j \right\} & \text{if } \widetilde{t}_u^{j_u} \leq \widetilde{t}_v^j < T, \end{cases} \tag{13}$$

where j_u denotes that from v's perspective, the neighbor u is resolving its j_u-th update, and the outcomes of its first $(j_u - 1)$ updates are known to v. In (13), if $0 \leq \widetilde{t}_v^j < \widetilde{t}_u^{j_u}$, let $\ell = k-1$, where $\widetilde{t}_u^{k-1} \leq \widetilde{t}_v^j < \widetilde{t}_u^k$; if $\widetilde{t}_u^{j_u} \leq \widetilde{t}_v^j < T$, let $\ell = j_u - 1$. By (11), it must hold that $(u,\ell) \prec (v,j)$. By induction hypothesis, we have $\widehat{Y}_u^{(\ell)} = Y_u^{(\ell)}$. By (10), the truncated update times preserve the order among all update times of node u and node v. Combining with the update rule of the Metropolis chain, it holds that

$$\forall u \in N_v : \quad Y_{t_v^j}(u) \in \mathcal{S}_{\widetilde{t}_v^j}(u).$$

Recall $C_v^j \triangleq \bigotimes_{u \in N_v} \mathcal{S}_{\widetilde{t}_v^j}(u)$. Then, we have

$$Y_{t_v^j}(N_v) \in C_v^j.$$

Recall that $P_{\mathsf{AC}} \triangleq \min_{\tau \in C_v^j} f_{\widehat{c},c'}^v(\tau)$ and $P_{\mathsf{RE}} \triangleq \min_{\tau \in C_v^j} \left(1 - f_{\widehat{c},c'}^v(\tau)\right)$ where $\widehat{c} = \widehat{Y}_v^{(j-1)}$ and $c' = c_v^j$. Since $(v, j-1) \prec (v,j)$, then by induction hypothesis, we have $\widehat{Y}_v^{(j-1)} = Y_v^{(j-1)}$. Hence, the threshold $f_{c,c'}^v(Y_{t_v^j}(N_v))$ in (12) where $c = Y_v^{(j-1)} = \widehat{Y}_v^{(j-1)}$, satisfies

$$P_{\mathsf{AC}} \leq f_{c,c'}^v(Y_{t_v^j}(N_v)) \quad \text{and} \quad P_{\mathsf{RE}} \leq 1 - f_{c,c'}^v(Y_{t_v^j}(N_v)). \tag{14}$$

Finally, there are two cases when v resolves the update (v,j).

- Case $\beta_v^j < P_{\mathsf{AC}}$: In **main algorithm**, it holds that $\widehat{Y}_v^{(j)} = c_v^j$. In continuous-time Metropolis chain, by (14), it holds that $\beta_v^j < f_{c,c'}^v(Y_{t_v^j}(N_v))$, thus $Y_v^{(j)} = c_v^j$. This implies $\widehat{Y}_v^{(j)} = Y_v^{(j)}$.
- Case $\beta_v^j \geq 1 - P_{\mathsf{RE}}$: In **main algorithm**, it holds that $\widehat{Y}_v^{(j)} = \widehat{Y}_v^{(j-1)}$. In continuous-time Metropolis chain, by (14), it holds that $\beta_v^j \geq f_{c,c'}^v(Y_{t_v^j}(N_v))$, thus $Y_v^{(j)} = Y_v^{(j-1)}$. This implies $\widehat{Y}_v^{(j)} = Y_v^{(j)}$, because $\widehat{Y}_v^{(j-1)} = Y_v^{(j-1)}$ by induction hypothesis.

Combining two cases proves that $\widehat{Y}_v^{(j)} = Y_v^{(j)}$.

5.3. **Analysis of Running Time.** We then upper bound the running time of the main algorithm.

Lemma 5.4. *Let $\zeta \geq 1$ be an arbitrary constant. With probability at least $1 - n^{-\zeta}$, the main algorithm terminates within $20\zeta(T\Delta + \log n)$ time units. If Condition 1.2 holds with constant C, with probability at least $1 - n^{-\zeta}$, the algorithm terminates within $20(\zeta + C)(T + \log n)$ time units.*

Note that each message of the main algorithm is of at most $10\zeta(\log n + \log\lceil T \rceil + \log q)$ bits (in fact, only 1 bit in **Phase II**). Theorem 5.1 is then implied by Lemma 5.2 and Lemma 5.4.

We then proceed to prove Lemma 5.4. Given an execution of the algorithm, we define the **Phase I** of the execution as the time duration between the beginning of the algorithm and the moment at which the last node enters its **Phase II**, and define the **Phase II** of the execution as the time duration between this moment and the termination of the algorithm.

Recall that for every node $v \in V$, the Poisson clock at v rings m_v times, where $m_v \sim \text{Pois}(T)$. By the concentration of Poisson distribution, it holds that

$$\Pr[\,\forall v \in V, m_v \leq 5T + (\zeta + 2)\log n\,] \geq 1 - \frac{1}{n^{2+\zeta}}.$$

It also holds with probability at least $1 - n^{-\zeta-1}$ that no two update times are too close to each other (e.g. within $< (n\lceil T \rceil)^{-5\zeta}$ difference) since they are generated by rate-1 Poisson clocks, thus every update (t_v^i, c_v^i) can be encoded in one message of $10\zeta(\log n + \log\lceil T \rceil + \log q)$ bits. Therefore with probability at least $1 - n^{-\zeta-1}$, the **Phase I** of the execution ends within $\max_{v \in V} m_v \leq 5T + (\zeta + 2)\log n$ time units.

It then remains to upper bound the length of the **Phase II** of the execution. We treat it as a standalone distributed algorithm, with different nodes starting asynchronously, while the complexity is measured starting from the moment the latest node starts.

Definition 5.5 (time complexity of a node in Phase II). For each node $v \in V$, we define the time for v *staying in* ***Phase II*** as the time duration between the earliest moment at which all nodes have entered their respective **Phase II** and the moment v terminates.

Lemma 5.6. *Let $\zeta \geq 1$ be an arbitrary constant. Fix any node $v \in V$. With probability at least $1 - n^{-\zeta-2}$, node v stays in* ***Phase II*** *for at most $12\zeta(T\Delta + \log n)$ time units. And if Condition 1.2 holds, then with probability at least $1 - n^{-\zeta-2}$, node v stays in* ***Phase II*** *for at most $12(\zeta + C)(T + \log n)$ time units, where C is the constant in Condition 1.2.*

Combined with the above analysis of **Phase I**, Lemma 5.6 is sufficient to imply Lemma 5.4.

5.3.1. *Proof of Lemma 5.6.* Fix all randomnesses of the main algorithm as defined in (8): m_v for each $v \in V$ and $(t_v^i, c_v^i, \beta_v^i)$ for each $v \in V$ and every $1 \leq i \leq m_v$, where all update times t_v^i are distinct.

Recall that we use the pair (v, i) to identify the i-th update of node v. Recall the *dependency chain* $\mathcal{D}_{(v,i)}$ that ends at (v, i), formally defined in (7).

The dependency chain satisfies the following monotonicity.

Proposition 5.7. *For any two consecutive updates $(u, j), (w, k)$ in a dependency chain $\mathcal{D}_{(v,i)}$, it holds that $(u, j) \prec (w, k)$ where the partial order $\prec$ is as defined in (9).*

Proof. First, if the resolution of (w, k) is self-triggered, then $u = w$ and $j = k - 1$, and hence $(u, j) \prec (w, k)$ clearly holds. Otherwise, w and u are neighbors, and the resolution of (w, k) is triggered by the resolution of (u, j). Then it must hold that $t_u^j < t_w^k$, since if otherwise, by definition (4), the construction of $\mathcal{S}_{t_w^k}(u)$ is unaffected by the resolution of the update (u, j) at a later time $t_u^j > t_w^k$, thus will not trigger any resolution condition of the update (w, k) if the condition has not been satisfied already. Therefore, it always holds that $(u, j) \prec (w, k)$. □

A direct consequence to the above proposition is that every dependency chain has finite length.

A key application of the dependency chain is that it can be used to upper bound the time complexity of Algorithm 4.

Proposition 5.8. *Fix any node $v \in V$. The number of time units for node v staying in* ***Phase II*** *is always upper bounded by the length of the dependency chain $\mathcal{D}_{(v,m_v)}$, where m_v denotes the total number of updates at node v.*

Proof. Consider the main algorithm. Let $\mathcal{T}$ be the earliest moment at which all nodes have entered their respective **Phase II** and $\mathcal{T}'$ the moment at which v terminates. The last update in dependency chain $\mathcal{D}_{(v,m_v)}$ is (v, m_v), which must be resolved at the moment $\mathcal{T}'$. Let (u, i) denote the first update in dependency chain $\mathcal{D}_{(v,m_v)}$. Then $i = 1$. Suppose (u, i) is resolved at the moment $\mathcal{T}''$. It must hold that $\mathcal{T}'' \le \mathcal{T}$, because (u, i) must be resolved once u enters **Phase-II**. By definition 5.5, the time for v staying in **Phase-II** is $\mathcal{T}' - \mathcal{T}$, which is upper bounded by $\mathcal{T}' - \mathcal{T}''$. By the definition of the dependency chain $\mathcal{D}_{(v,m_v)}$, it is easy to see that the number of time units in the time duration between $\mathcal{T}''$ and $\mathcal{T}'$ is at most the length of $\mathcal{D}_{(v,m_v)}$. □

We then proceed to prove Lemma 5.6.

Fix a node $v \in V$. Let R_v denote the number of times units that node v stays in **Phase II**.

Fix an integer $\ell > 0$. We bound the probability that $R_v \ge \ell$. By Proposition 5.8 and Proposition 5.7, if $R_v \ge \ell$, then there exist a sequence of nodes $v_1, v_1, \ldots, v_\ell \in V$ and a sequence of times $0 < t_1 < t_2 < \cdots < t_\ell < T$ such that

(i) $v_\ell = v$ and $v_{j+1} \in N_{v_j} \cup \{v_j\}$ for every $1 \le j \le \ell - 1$;
(ii) for every $1 \le j \le \ell$, t_j is an update time of v_j, and we denote by k_j the order of this update time at v_j, i.e. $t_j = t_{v_j}^{k_j}$;
(iii) for every $2 \le j \le \ell$, $(v_{j-1}, k_{j-1}) \prec (v_j, k_j)$, and if $v_{j-1} \ne v_j$, the resolution of the update (v_j, k_j) is triggered by the resolution of the adjacent update (v_{j-1}, k_{j-1}).

We call such a sequence of nodes $v_1, v_1, \ldots, v_\ell \in V$ a *dependency path* if there exists a sequence of times $0 < t_1 < t_2 < \cdots < t_\ell < T$ together with $v_1, v_1, \ldots, v_\ell$ satisfying the above conditions.

We first prove the simple $12\zeta(T\Delta + \log n)$ upper bound. Fix a sequence of nodes $v_1, v_2, \ldots, v_\ell$ satisfying the condition in (i). Consider the event that there exists a sequence of times $0 < t_1 < t_2 < \cdots < t_\ell < T$ such that the Poisson clock at node v_i rings at time t_i. By [HS07, Observation 3.2], the probability of this event is at most $\left(\frac{eT}{\ell}\right)^\ell$. A union bound over at most $(\Delta+1)^\ell$ possible paths satisfying the condition (i) implies

$$\Pr[\, R_v \ge \ell \,] \le (\Delta + 1)^\ell \left(\frac{eT}{\ell}\right)^\ell \le \left(\frac{2eT\Delta}{\ell}\right)^\ell .$$

Choosing $\ell = 4eT\Delta + (\zeta + 2)\log n \le 12\zeta(T\Delta + \log n)$, we have

$$\Pr[R_v \ge 4eT\Delta + (\zeta + 2)\log n] \le \left(\frac{1}{2}\right)^{(\zeta+2)\log n} = \frac{1}{n^{\zeta+2}}.$$

Hence, node v stays in **Phase II** for at most $12\zeta(T\Delta + \log n)$ time units with probability at least $1 - n^{-\zeta-2}$. Remark that this proof does not need to use Condition 1.2.

We now prove the $12(\zeta + C)(T + \log n)$ upper bound under Condition 1.2. Let $0 \le s < \ell$ be an integer. We use $\mathcal{P}(\ell, s)$ to denote the set of all sequences $v_1, v_2, \ldots, v_\ell \in V$ satisfying that $v = v_\ell$, $v_{j+1} \in N(v_j) \cup \{v_j\}$ for every $1 \le j < \ell$, and $s = |\{1 \le j \le \ell - 1 \mid v_j \ne v_{j+1}\}|$.

By the above argument, we have

$$\Pr[\, R_v \ge \ell \,] \le \sum_{s=0}^{\ell-1} \sum_{P \in \mathcal{P}(\ell,s)} \Pr[\, P \text{ is a dependency path} \,]. \tag{15}$$

We then bound the probability that a sequence P is a dependency path. Let random variable $\mathcal{N} \in \mathbb{Z}_{\ge 0}$ denote the total number of updates in the entire network before time T. Apparently, $\mathcal{N}$ is given by the total number of times that n rate-1 Poisson clocks ring up to time T and follows the Poisson distribution with mean nT. We have

$$\Pr[\, P \text{ is a dependency path}\,] = \sum_{m\geq 0} \Pr[\, \mathcal{N} = m\,] \Pr[\, P \text{ is a dependency path} \mid \mathcal{N} = m\,]$$

$$= \mathrm{e}^{-nT} \sum_{m\geq 0} \frac{(nT)^m}{m!} \Pr[\, P \text{ is a dependency path} \mid \mathcal{N} = m\,]. \tag{16}$$

Conditioning on $\mathcal{N} = m$ for a fixed integer $m \geq 0$, we define the following random variables:

$$(U_1, T_1, C_1, \beta_1), (U_2, T_2, C_2, \beta_2), \ldots, (U_m, T_m, C_m, \beta_m), \tag{17}$$

where 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 proposal distributed as ν_{U_i}, and each $\beta_i \in [0, 1)$ is uniformly distributed over $[0, 1)$, such that:

- the Poisson clock at node U_i rings at time T_i;
- node U_i proposes C_i for its update at time T_i;
- node U_i samples the random real number $\beta_i \in [0, 1)$ at Line 1 of Algorithm 6 to resolve its update at time T_i.

We now use UD_k for $1 \leq k \leq m$ to identify the update of node U_k at time T_k with proposal C_k.

Conditioning on $\mathcal{N} = m$, if $P = v_1, v_2, \ldots, v_\ell$ is a dependency path, then from the above discussion we know that there exist ℓ 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 $1 \leq j \leq \ell$, $U_{p(j)} = v_j$;
- **event** $\mathcal{A}_2^{(j)}$, where $2 \leq j \leq \ell$: either $U_{p(j-1)} = U_{p(j)}$ or the resolution of $\mathsf{UD}_{p(j)}$ is triggered by the resolution of the adjacent update $\mathsf{UD}_{p(j-1)}$.

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

$$\Pr\left[\mathcal{A}_1 \wedge \left(\bigwedge_{j=2}^{\ell} \mathcal{A}_2^{(j)} \right) \mid \mathcal{N} = m \right]$$

$$= \Pr[\, \mathcal{A}_1 \mid \mathcal{N} = m\,] \prod_{j=2}^{\ell} \Pr\left[\mathcal{A}_2^{(j)} \mid \mathcal{N} = m \wedge \mathcal{A}_1 \wedge \left(\bigwedge_{k=2}^{j-1} \mathcal{A}_2^{(k)} \right) \right]. \tag{18}$$

Note that conditioning on $\mathcal{N} = m$, each U_i is uniformly and independently distributed in V. This can be proved by an alternative equivalent process: there is a single rate-n Poisson clock, and once the clock rings, a node is picked uniformly at random. Then conditioning on the rate-n Poisson clock rings for m times, each U_i is uniformly and independently distributed in V. Therefore,

$$\Pr[\, \mathcal{A}_1 \mid \mathcal{N} = m\,] = \left(\frac{1}{n} \right)^{\ell}. \tag{19}$$

We further make the following claim on events $\mathcal{A}_2^{(j)}$.

Claim 5.9. *Assume that Condition 1.2 holds. For any $2 \leq j \leq \ell$ where $v_j \neq v_{j-1}$, it holds that*

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

where C is the constant in Condition 1.2.

Since $P \in \mathcal{P}(\ell, s)$, there are exactly s indices j such that $v_j \neq v_{j+1}$. Combining(18), (19) and Claim 5.9 yields

$$\Pr\left[\mathcal{A}_1 \wedge \left(\bigwedge_{j=2}^{\ell} \mathcal{A}_2^{(j)} \right) \mid \mathcal{N} = m \right] \leq \left(\frac{1}{n} \right)^{\ell} \left(\frac{2C}{\Delta} \right)^{s}.$$

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

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

Finally, note that $|\mathcal{P}(\ell, s)| \le \binom{\ell-1}{s}\Delta^s$. Together with (15) and (16) it gives that

$$\begin{aligned}\Pr[R_v \ge \ell] &\le \sum_{s=0}^{\ell-1} \sum_{P \in \mathcal{P}(\ell,s)} \mathrm{e}^{-nT} \sum_{m=0}^{\infty} \frac{(nT)^m}{m!} \binom{m}{\ell} \left(\frac{1}{n}\right)^{\ell} \left(\frac{2C}{\Delta}\right)^{s} \\ &\le \sum_{s=0}^{\ell-1} \binom{\ell-1}{s} \Delta^s \mathrm{e}^{-nT} \sum_{m=\ell}^{\infty} \frac{(nT)^m}{m!} \binom{m}{\ell} \left(\frac{1}{n}\right)^{\ell} \left(\frac{2C}{\Delta}\right)^{s} \\ &\le \frac{T^{\ell}}{\ell!} (1+2C)^{\ell} \\ &\le \left(\frac{T\mathrm{e}(1+2C)}{\ell}\right)^{\ell}.\end{aligned}$$

Choosing $\ell = 2\mathrm{e}\,(1 + 2C)\,T + (\zeta + 2)\log n \le 12(\zeta + C)(T + \log n)$, we have

$$\Pr[R_v \ge 2\mathrm{e}\,(1+2C)\,T + (\zeta+2)\log n] \le \left(\frac{1}{2}\right)^{(\zeta+2)\log n} = \frac{1}{n^{\zeta+2}}.$$

Hence, node v stays in **Phase II** for at most $12(\zeta + C)(T + \log n)$ time units with probability at least $1 - n^{-\zeta-2}$. This proves Lemma 5.6.

Proof. (Proof of Claim 5.9) By the definition of fully-asynchronous message-passing model in Section 2, all the message delays are determined by an adversarial scheduler who is adaptive to the entire input. Given the input of each node, we can fix all the message delays in **main algorithm**. Denote the delays of all messages as $\mathcal{F}_D$. Note that $\mathcal{F}_D$ satisfies the constraint that each unidirectional channel is a reliable FIFO channel.

Recall that $\mathcal{N}$ is the total number of updates in the entire network before time T. Consider the random variables defined in (17):

$$(U_1, T_1, C_1, \beta_1), (U_2, T_2, C_2, \beta_2), \ldots, (U_{\mathcal{N}}, T_{\mathcal{N}}, C_{\mathcal{N}}, \beta_{\mathcal{N}}), \tag{20}$$

where $T_1 < T_2 < \ldots < T_{\mathcal{N}}$. We first fix the randomness of all Poisson clocks:

- $\mathcal{F}_1$: fix $\mathcal{N} = m$ and fix the values of all $U_1, U_2, \ldots, U_m$ and $T_1, T_2, \ldots, T_m$.

Recall $P = v_1, v_2, \ldots, v_\ell$ is the fixed path and $1 \le p(1) < p(2) < \ldots < p(\ell) \le m$ is the ℓ fixed indices. Fix an integer $2 \le j \le \ell$ such that $v_{j-1} \ne v_j$. We then fix the randomness of the first $p(j) - 1$ updates in sequence (20):

- $\mathcal{F}_2$: fix the values of all C_k, β_k for $1 \le k \le p(j) - 1$.

We claim that the following two results hold.

(R1) Given any $\mathcal{F}_D$, $\mathcal{F}_1$ and $\mathcal{F}_2$, it holds that $\mathcal{N} = m$ and the occurrences of events $\mathcal{A}_1$ and $\mathcal{A}_2^{(k)}$ for all $2 \le k \le j-1$ are fully determined.

(R2) For any $\mathcal{F}_D$, $\mathcal{F}_1$ and $\mathcal{F}_2$ under which $\mathcal{A}_1$ and all $\mathcal{A}_2^{(k)}$ for $2 \le k \le j-1$ occur, if Condition 1.2 is satisfied, then:

$$\Pr[\, \mathcal{A}_2^{(j)} \mid \mathcal{F}_D \wedge \mathcal{F}_1 \wedge \mathcal{F}_2 \,] \le \frac{2C}{\Delta},$$

where the probability takes over the randomness of unfixed variables $C_{p(j)}$ and $\beta_{p(j)}$.

The Claim 5.9 is proved by combining above two results.

We first prove result (R1). It is easy to see that $\mathcal{F}_1$ determines whether the event $\mathcal{A}_1$ occurs. We prove that the occurrences of events $\mathcal{A}_2^{(k)}$ for all $2 \le k \le j-1$ are determined. Given $\mathcal{F}_1$ and $\mathcal{F}_2$, the evolution of the continuous-time Metropolis chain Y_t from $t = 0$ to $t = T_{p(j)} - \epsilon$ is fully determined. Given $\mathcal{F}_D$, $\mathcal{F}_1$ and $\mathcal{F}_2$, define algorithm $\mathcal{A}$ as a modified version of the **main algorithm** such that for each node $v \in V$:

- in **Phase I**, node v generates all the update times and random proposals conditioning on $\mathcal{F}_1$ and $\mathcal{F}_2$, then exchanges this information with neighbors as the **main algorithm**.
- in **Phase II**, node v resolves each update as the **main algorithm** and v samples β in Line 1 of Algorithm 6 conditioning on $\mathcal{F}_2$; once node v needs to use any variable whose value is not fixed by $\mathcal{F}_1$ and $\mathcal{F}_2$, then the algorithm $\mathcal{A}$ at node v terminates immediately.

By a similar induction argument in Section 5.2, it can be verified that algorithm $\mathcal{A}$ simulates the continuous-time Metropolis chain up to time $T_{p(j)} - \epsilon$ and generates $(\widehat{Y}_t)_{t \in [0, T_{p(j)})}$. The algorithm $\mathcal{A}$ resolves all the updates (U_k, T_k, C_k, β_k) for $1 \le k \le p(j) - 1$ in (20). In algorithm $\mathcal{A}$, for each node $v \in V$, the **Phase II** of node v is fully determined by $\mathcal{F}_D, \mathcal{F}_1$ and $\mathcal{F}_2$. For each $2 \le k \le j-1$, by the definition of event $\mathcal{A}_2^{(k)}$, its occurrence is fully determined given $\mathcal{F}_D, \mathcal{F}_1$ and $\mathcal{F}_2$.

We then prove result (R2). Now, we only consider $\mathcal{F}_D$, $\mathcal{F}_1$ and $\mathcal{F}_2$ under which $\mathcal{A}_1$ and all $\mathcal{A}_2^{(k)}$ for $2 \le k \le j-1$ occur. Hence, we assume $U_{p(j-1)} = v_{j-1}$ and $U_{p(j)} = v_j$. Recall that $v_j \neq v_{j-1}$ and UD_k denotes the k-th update (U_k, T_k, C_k, β_k) in (20). The event $\mathcal{A}_2^{(j)}$ occurs if and only if the update $\mathsf{UD}_{p(j)}$ is triggered by the resolution of the adjacent update $\mathsf{UD}_{p(j-1)}$. In Algorithm 6, to resolve the update $\mathsf{UD}_{p(j)}$, node v_j needs to compute the set $\mathcal{S}_{T_{p(j)}}(w)$ defined in (4) for all $w \in N(v_j)$. For any $1 \le k \le m$ and $w \in N(U_k)$, let $\mathsf{Msg}_k^{\to w}$ denote the message sent from U_k to w that indicates whether the update UD_k is accepted. When computing the set $\mathcal{S}_{T_{p(j)}}(w)$ for $w \in N(v_j)$, node v_j only uses the following information:

(I1) $\{T_k \mid 1 \le k \le m \wedge U_k = w\} \cup \{T_{p(j)}\}$;
(I2) $\{C_k \mid k \le p(j) - 1 \wedge U_k = w\} \cup \{\widehat{Y}_0(w)\}$;
(I3) all messages $\mathsf{Msg}_k^{\to v_j}$ received by v_j satisfying $k \le p(j) - 1$ and $U_k = w$.

By the definition in (4), node v_j computes $\mathcal{S}_{T_{p(j)}}(w)$ based on the current j_w and $(\widehat{Y}_w^{(j)})_{0 \le j \le j_w}$, where j_w and $(\widehat{Y}_w^{(j)})_{0 \le j \le j_w}$ are computed according to the messages received from w. Note that computing $\mathcal{S}_{T_{p(j)}}(w)$ only needs to use the information about the updates at node w whose update times are before $T_{p(j)}$, because $\mathcal{S}_{T_{p(j)}}(w)$ defined in (4) is the set of possible states for w at time $T_{p(j)}$. Also note that in **main algorithm**, all update times are represented with bounded precision satisfying (10), the partial order $\prec$ in (9) is preserved. Thus, node v_j only uses information in (I1), (I2), and (I3) to compute $\mathcal{S}_{T_{p(j)}}(w)$.

Given $\mathcal{F}_D$, $\mathcal{F}_1$ and $\mathcal{F}_2$, the sets in (I1) and (I2) are fixed and node v_j knows these sets once v_j enters the **Phase-II**. Define the set of messages

$$\mathcal{M} = \left\{ \mathsf{Msg}_k^{\to v_j} \mid k \le p(j) - 1 \wedge U_k \in N(v_j) \right\}.$$

According to the proof of result (R1), we know that for each message $M \in \mathcal{M}$, the content of M and the moment at which node v_j processes M is fully determined given $\mathcal{F}_D$, $\mathcal{F}_1$ and $\mathcal{F}_2$. Consider the moment $\mathcal{T}$ at which node v_j processes the message $\mathsf{Msg}_{p(j-1)}^{\to v_j} \in \mathcal{M}$ to update the current $(Y, \mathbf{j})$. Suppose after the update, the pair $(Y, \mathbf{j})$ becomes $(Y', \mathbf{j}')$. For any $w \in N(v_j)$, let $\mathcal{S}(w)$ be the set $\mathcal{S}_{T_{p(j)}}(w)$ computed by v_j according to (4) based on $(Y, \mathbf{j})$; and let $\mathcal{S}'(w)$ be the set $\mathcal{S}_{T_{p(j)}}(w)$ computed by v_j according to (4) based on $(Y', \mathbf{j}')$. Given $\mathcal{F}_D$, $\mathcal{F}_1$ and $\mathcal{F}_2$, it holds that

- for all $w \in N(v_j)$, both $\mathcal{S}(w)$ and $\mathcal{S}'(w)$ are fixed,

furthermore, it holds that $U_{p(j-1)} = v_{j-1}$ and

(P1) for all $w \in N(v_j) \setminus \{v_{j-1}\}$, $\mathcal{S}(w) = \mathcal{S}'(w)$;
(P2) $\mathcal{S}'(v_{j-1}) \subseteq \mathcal{S}(v_{j-1})$ and $|\mathcal{S}(v_{j-1})| - |\mathcal{S}'(v_{j-1})| \le 1$;
(P3) for all $w \in N(v_j)$, $|\mathcal{S}'(w)| \ge 1$.

Properties (P1) and (P2) hold because v_j only processes one the message $\mathsf{Msg}_{p(j-1)}^{\to v_j}$ sent from v_{j-1} at moment $\mathcal{T}$. Property (P3) holds because the set $\mathcal{S}_{T_{p(j)}}(w)$ contains at least one element due to the definition in (4).

Let c denote $\widehat{Y}_t(v_j)$ where $t = T_{p(j)} - \epsilon$. Note that c is fixed given $\mathcal{F}_D$, $\mathcal{F}_1$ and $\mathcal{F}_2$. Let c' denote the proposal $C_{p(j)}$. Let $P_{\mathsf{AC}}, P_{\mathsf{RE}}$ be the acceptance and rejection thresholds for update $\mathsf{UD}_{p(j)}$ obtained from $(\mathcal{S}(w))_{w\in N(v_j)}$ and $P'_{\mathsf{AC}}, P'_{\mathsf{RE}}$ the thresholds obtained from $(\mathcal{S}'(w))_{w\in N(v_j)}$, formally

$$P_{\mathsf{AC}} = \min_{\tau\in C} f^{v_j}_{c,c'}(\tau) \qquad P_{\mathsf{RE}} = 1 - \max_{\tau\in C} f^{v_j}_{c,c'}(\tau)$$
$$P'_{\mathsf{AC}} = \min_{\tau\in C'} f^{v_j}_{c,c'}(\tau) \qquad P'_{\mathsf{RE}} = 1 - \max_{\tau\in C'} f^{v_j}_{c,c'}(\tau),$$

where $C = \bigotimes_{w\in N(v_j)} \mathcal{S}(w)$ and $C' = \bigotimes_{w\in N(v_j)} \mathcal{S}'(w)$.

If the event $\mathcal{A}_2^{(j)}$ occurs, then the following event $\mathcal{B}_j$ must occur.

- **event** $\mathcal{B}_j$: $(P_{\mathsf{AC}} \le \beta_{p(j)} < 1 - P_{\mathsf{RE}}) \wedge (\beta_{p(j)} < P'_{\mathsf{AC}} \vee \beta_{p(j)} \ge 1 - P'_{\mathsf{RE}})$.

Suppose the event $\mathcal{A}_2^{(j)}$ occurs. Node v_j resolves the update $\mathsf{UD}_{p(j)}$ right after moment $\mathcal{T}$, which implies $\beta_{p(j)} < P'_{\mathsf{AC}} \vee \beta_{p(j)} \ge 1 - P'_{\mathsf{RE}}$. And node v_j cannot resolve the update $\mathsf{UD}_{p(j)}$ right before moment $\mathcal{T}$, which implies $P_{\mathsf{AC}} \le \beta_{p(j)} < 1 - P_{\mathsf{RE}}$.

Given $\mathcal{F}_D$, $\mathcal{F}_1$ and $\mathcal{F}_2$, note that $c' = C_{p(j)}$ is an independent random proposal from distribution ν_{v_j} and $\beta_{p(j)}$ is an independent random real number uniformly distributed over $[0,1)$. Then

$$\begin{aligned}\Pr[\,\mathcal{A}_2^{(j)} \mid \mathcal{F}_D \wedge \mathcal{F}_1 \wedge \mathcal{F}_2\,] &\le \Pr[\,\mathcal{B}_j \mid \mathcal{F}_D \wedge \mathcal{F}_1 \wedge \mathcal{F}_2\,]\\ &\le \mathbb{E}_{c'\sim\nu_{v_j}}\left[(P'_{\mathsf{AC}} - P_{\mathsf{AC}}) + (P'_{\mathsf{RE}} - P_{\mathsf{RE}}) \mid \mathcal{F}_D \wedge \mathcal{F}_1 \wedge \mathcal{F}_2\,\right],\end{aligned} \tag{21}$$

where the last inequality holds because $\beta_{p(j)}$ is uniformly distributed over $[0,1)$, $P'_{\mathsf{AC}} \ge P_{\mathsf{AC}}$ and $P'_{\mathsf{RE}} \ge P_{\mathsf{RE}}$ (because $\mathcal{S}'(w) \subseteq \mathcal{S}(w)$ for all $w \in N(v_j)$).

Finally, we bound the expectation in (21). Given $\mathcal{F}_D$, $\mathcal{F}_1$ and $\mathcal{F}_2$, the value $c = \widehat{Y}_{T_{p(j)}-\epsilon}(v_j)$ and all the sets $\mathcal{S}(w)$ and $\mathcal{S}'(w)$ are determined, and the Properties (P1), (P2) and (P3) hold. Recall

$$(P'_{\mathsf{AC}} - P_{\mathsf{AC}}) + (P'_{\mathsf{RE}} - P_{\mathsf{RE}}) = \min_{\tau\in C'} f^{v_j}_{c,c'}(\tau) - \min_{\tau\in C} f^{v_j}_{c,c'}(\tau) + \max_{\tau\in C} f^{v_j}_{c,c'}(\tau) - \max_{\tau\in C'} f^{v_j}_{c,c'}(\tau).$$

We introduce the following optimization problem to find the maximum value of the expectation in (21) under the worst case of $\mathcal{F}_D$, $\mathcal{F}_1$ and $\mathcal{F}_2$. Fix two nodes $\{v,u\} \in E$, define the optimization problem $\mathfrak{P}(v,u)$ as follows.

$$\boxed{\begin{array}{lll}
\text{variables} & S_1(w) \subseteq [q], S_2(w) \subseteq [q] & \forall w \in N_v\\
 & c \in [q] & \\
\text{maximize} & \displaystyle\sum_{c'\in[q]} \nu_v(c')\left(\min_{\tau\in C_2} f^v_{c,c'}(\tau) - \min_{\tau\in C_1} f^v_{c,c'}(\tau) + \max_{\tau\in C_1} f^v_{c,c'}(\tau) - \max_{\tau\in C_2} f^v_{c,c'}(\tau)\right) & \\
\text{subject to} & \displaystyle C_1 = \bigotimes_{w\in N_v} S_1(w), \quad C_2 = \bigotimes_{w\in N_v} S_2(w) & \\
 & S_2(u) \subset S_1(u) & \\
 & |S_1(u)| - |S_2(u)| = 1 & \\
 & |S_2(w)| \ge 1 & \forall w \in N_v\\
 & S_2(w) = S_1(w) & \forall w \in N_v \setminus \{u\}
\end{array}} \tag{22}$$

Remark that we use constraint $|S_1(u)| - |S_2(u)| = 1$ rather than $|S_1(u)| - |S_2(u)| \le 1$ as Property (P2) 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 $\mathfrak{P}(v,u)$ is at most

$$2 \max_{a,b,c\in[q]} \mathbb{E}_{c'\sim\nu_v}\left[\delta_{u,a,b} f^v_{c,c'}\right].$$

This result is proved in Section 5.3.2. The expectation in (21) is upper bounded by the optimal value of the objective function in problem $\mathfrak{P}(v_j, v_{j-1})$. By Condition 1.2, we have

$$\Pr[\,\mathcal{A}_2^{(j)} \mid \mathcal{F}_D \wedge \mathcal{F}_1 \wedge \mathcal{F}_2\,] \le 2 \max_{a,b,c\in[q]} \mathbb{E}_{c'\sim\nu_{v_j}}\left[\delta_{v_{j-1},a,b} f^{v_j}_{c,c'}\right] \le \frac{2C}{\Delta}.$$

□

5.3.2. *Analysis of the Optimization Problem.*

Lemma 5.10. *Fix an edge $\{v,u\} \in E$. Let OPT denote the objective function value of the optimal solution to problem $\mathfrak{P}(v,u)$ defined in (22). It holds that*

$$OPT \le 2 \max_{a,b,c\in[q]} \mathbb{E}_{c'\sim\nu_v}\left[\delta_{u,a,b} f^v_{c,c'}\right].$$

Proof. Suppose we replace the objective function in problem $\mathfrak{P}(v,u)$ with

$$\text{maximize} \quad \sum_{c'\in[q]} \nu_v(c')\left(\min_{\tau\in C_2} f^v_{c,c'}(\tau) - \min_{\tau\in C_1} f^v_{c,c'}(\tau)\right) \tag{23}$$

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.

Similarly, suppose we replace the objective function in problem $\mathfrak{P}(v,u)$ with

$$\text{maximize} \quad \sum_{c'\in[q]} \nu_v(c')\left(\max_{\tau\in C_1} f^v_{c,c'}(\tau) - \max_{\tau\in C_2} f^v_{c,c'}(\tau)\right)$$

and keep all variables and constraints unchanged. We obtain another new optimization problem. Let OPT_2 denote the objective function value of the optimal solution to this problem.

It is easy to verify

$$\mathrm{OPT} \le \mathrm{OPT}_1 + \mathrm{OPT}_2.$$

We show that

$$\mathrm{OPT}_1 \le \max_{a,b,c\in[q]} \mathbb{E}_{c'\sim\nu_v}\left[\delta_{u,a,b} f^v_{c,c'}\right] \tag{24}$$

$$\mathrm{OPT}_2 \le \max_{a,b,c\in[q]} \mathbb{E}_{c'\sim\nu_v}\left[\delta_{u,a,b} f^v_{c,c'}\right]. \tag{25}$$

This proves the lemma.

We prove inequality (24). Inequality (25) can be proved by going through a similar proof.

Consider the new optimization problem with objective function (23). We claim the following result for this problem.

Claim 5.11. *There exists an optimal solution $SOL^\star = (S_1^\star, S_2^\star, c^\star)$ such that $|S_2^\star(u)| = 1$, where $S_1^\star = (S_1^\star(w))_{w\in N(v)}$ and $S_2^\star = (S_2^\star(w))_{w\in N(v)}$.*

Thus, we have $|S_1^\star(u)| = 2$ due to the constraint of the problem. Suppose $S_1^\star(u) = \{a,b\}$ and $S_2^\star(u) = \{b\}$. Fix a value $c' \in [q]$, define

$$\gamma_1(c') = \min_{\tau\in C_1^\star} f^v_{c^\star,c'}(\tau) = f^v_{c^\star,c'}(\tau')$$
$$\gamma_2(c') = \min_{\tau\in C_2^\star} f^v_{c^\star,c'}(\tau) = f^v_{c^\star,c'}(\tau''),$$

where

$$C_1^\star = \bigotimes_{w\in N_v} S_1^\star(w), \quad C_2^\star = \bigotimes_{w\in N_v} S_2^\star(w),$$

and $\tau' = \arg\min_{\tau\in C_1^\star} f^v_{c^\star,c'}(\tau)$, $\tau'' = \arg\min_{\tau\in C_2^\star} f^v_{c^\star,c'}(\tau)$. It must hold that $\tau''_u = b$ because $S_2^\star(u) = \{b\}$. There are two cases for τ'_u: $\tau'_u = a$ or $\tau'_u = b$, because $S_1^\star(u) = \{a,b\}$.

Suppose $\tau'_u = b$. Since $S_1^\star(w) = S_2^\star(w)$ for all $w \in N_v \setminus \{u\}$, then we must have

$$\gamma_2(c') - \gamma_1(c') = 0 \le \delta_{u,a,b} f^v_{c^\star,c'}.$$

Suppose $\tau'_u = a$. We define $\tau''' \in [q]^{N_v}$ as

$$\tau'''_w = \begin{cases} b & \text{if } w = u \\ \tau'_w & \text{if } w \ne u. \end{cases}$$

Note that $b \in S_2^\star(u)$ and $\tau'_w \in S_2^\star(w)$ for all $w \in N_v \setminus \{u\}$ (because $S_2^\star(w) = S_1^\star(w)$). We have $\tau''' \in C_2^\star$, which implies $f^v_{c^\star,c'}(\tau''') \geq f^v_{c^\star,c'}(\tau'')$. Hence

$$\gamma_2(c') - \gamma_1(c') = f^v_{c^\star,c'}(\tau'') - f^v_{c^\star,c'}(\tau') \leq f^v_{c^\star,c'}(\tau''') - f^v_{c^\star,c'}(\tau') \leq \delta_{u,a,b} f^v_{c^\star,c'}.$$

The last inequality is because τ' and τ''' agree on all nodes except u and $\tau'_u = a, \tau'''_u = b$.

Combining above two cases together, we have

$$\begin{aligned}
\mathrm{OPT}_1 &= \sum_{c' \in [q]} \nu_v(c') \left(\min_{\tau \in C_2^\star} f^v_{c^\star,c'}(\tau) - \min_{\tau \in C_1^\star} f^v_{c^\star,c'}(\tau) \right) \\
&= \sum_{c' \in [q]} \nu_v(c') \left(\gamma_2(c') - \gamma_1(c') \right) \\
&\leq \sum_{c' \in [q]} \nu_v(c') \delta_{u,a,b} f^v_{c^\star,c'} \\
&= \mathbb{E}_{c' \sim \nu_v} \left[\delta_{u,a,b} f^v_{c^\star,c'} \right] \\
&\leq \max_{a,b,c \in [q]} \mathbb{E}_{c' \sim \nu_v} \left[\delta_{u,a,b} f^v_{c,c'} \right].
\end{aligned}$$

This proves the inequality (24). □

Proof. (Proof of Claim 5.11) Suppose $\mathrm{SOL}^* = (S_1^*, S_2^*, c^*)$ is an optimal solution with $|S_2^*(u)| > 1$. Note that $S_2^*(u) \subset S_1^*(u)$. Let $b \in [q]$ be an arbitrary element in $S_2^*(u)$. We remove the element b from both $S_1^*(u)$ and $S_2^*(u)$ to obtain a new solution $\mathrm{SOL}^\circ = \{S_1^\circ, S_2^\circ, c^\circ\}$. Namely

$$\begin{aligned}
S_1^\circ(u) &= S_1^*(u) \setminus \{b\} \\
S_2^\circ(u) &= S_2^*(u) \setminus \{b\}
\end{aligned}$$

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 SOL° also satisfies all the constraints. We will prove that SOL° is also an optimal solution. Since $|S_2^\circ(u)| = |S_2^*(u)| - 1$, then we can repeat this argument to find the optimal solution $\mathrm{SOL}^\star$ with $|S_2^\star(u)| = 1$.

We denote the objective function value of the solution SOL^* as

$$g(\mathrm{SOL}^*) = \sum_{c' \in [q]} \nu_v(c') \left(\min_{\tau \in C_2^*} f^v_{c^*,c'}(\tau) - \min_{\tau \in C_1^*} f^v_{c^*,c'}(\tau) \right),$$

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 SOL° as $g(\mathrm{SOL}^\circ)$. Suppose $S_1^*(u) \setminus S_2^*(u) = \{a\}$. We define the following set of values $S_a \subseteq [q]$ as

$$S_a \triangleq \left\{ c' \in [q] \mid \min_{\tau \in C_2^*} f^v_{c^*,c'}(\tau) > \min_{\tau \in C_1^*} f^v_{c^*,c'}(\tau) \right\}.$$

Note that $C_2^* \subset C_1^*$. We must have $\min_{\tau \in C_2^*} f^v_{c^*,c'}(\tau) \geq \min_{\tau \in C_1^*} f^v_{c^*,c'}(\tau)$ for all $c' \in [q]$. Then $g(\mathrm{SOL}^*)$ can be rewritten as

$$g(\mathrm{SOL}^*) = \sum_{c' \in S_a} \nu_v(c') \left(\min_{\tau \in C_2^*} f^v_{c^*,c'}(\tau) - \min_{\tau \in C_1^*} f^v_{c^*,c'}(\tau) \right). \tag{26}$$

For each $c' \in S_a$, suppose $\min_{\tau \in C_1^*} f^v_{c^*,c'}(\tau) = f^v_{c^*,c'}(\tau^*)$, then it must hold that $\tau^*_u = a$. 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 $\tau^*_u \neq a$, we must have $\min_{\tau \in C_2^*} f^v_{c^*,c'}(\tau) = \min_{\tau \in C_1^*} f^v_{c^*,c'}(\tau)$. This is contradictory to $c' \in S_a$.

Consider the solution SOL$^\circ$. Note that the set $S_1^\circ(u) = S_1^*(u) \setminus \{b\}$ and $S_2^\circ(u) = S_2^*(u) \setminus \{b\}$, where $b \neq a$ because $b \in S_2^*(u)$. By the definition of SOL$^\circ(u)$, we have

$$\forall c' \in S_a : \quad \min_{\tau \in C_1^*} f^v_{c^*,c'}(\tau) = \min_{\tau \in C_1^\circ} f^v_{c^\circ,c'}(\tau) \tag{27}$$

$$\forall c' \in S_a : \quad \min_{\tau \in C_2^*} f^v_{c^*,c'}(\tau) \leq \min_{\tau \in C_2^\circ} f^v_{c^\circ,c'}(\tau), \tag{28}$$

where

$$C_1^\circ = \bigotimes_{w \in N_v} S_1^\circ(w), \quad C_2^\circ = \bigotimes_{w \in N_v} S_2^\circ(w).$$

Recall that $c^\circ = c^*$, $S_1^\circ(w) = S_1^*(w)$, $S_2^\circ(w) = S_2^*(w)$ for all $w \in N_v \setminus \{u\}$. Thus (27) holds because $a \in S_1^\circ(u)$ and (28) holds because $S_2^\circ(u) \subset S_2^*(u)$ (hence $C_2^\circ \subset C_2^*$). Combining (27) and (28) together, we have

$$\begin{aligned} g(\mathrm{SOL}^\circ) &= \sum_{c' \in [q]} \nu_v(c') \left(\min_{\tau \in C_2^\circ} f^v_{c^\circ,c'}(\tau) - \min_{\tau \in C_1^\circ} f^v_{c^\circ,c'}(\tau) \right) \\ &\geq \sum_{c' \in S_a} \nu_v(c') \left(\min_{\tau \in C_2^\circ} f^v_{c^\circ,c'}(\tau) - \min_{\tau \in C_1^\circ} f^v_{c^\circ,c'}(\tau) \right) \\ &\geq \sum_{c' \in S_a} \nu_v(c') \left(\min_{\tau \in C_2^*} f^v_{c^*,c'}(\tau) - \min_{\tau \in C_1^*} f^v_{c^*,c'}(\tau) \right) \\ &= g(\mathrm{SOL}^*), \end{aligned}$$

where the last equation holds due to (26). Thus SOL$^\circ$ is also an optimal solution. □

6. Conclusion and Future Work

Markov chains are basic tools for Gibbs sampling. It is well known that the classic single-site Markov chain is in fact a sequential discretization of the idealized continuous-time parallel chain, with continuous time T corresponding to $nT \pm O(\log n)$ sequential steps. Although having been studied for decades, a basic question remains to be open: How to give a parallel discretization of the idealized continuous-time chain, without significant slowdown? Here a fundamental barrier is that each single-site update of the chain depends on the current states of the neighbors, and hence concurrent execution may be faulty. Meanwhile, avoiding concurrent adjacent updates slows down the parallel execution.

In this work, we show that for all Metropolis chains, assuming some moderate Lipschitz condition for the Metropolis filter, any time-T continuous chain can be faithfully simulated by a distributed algorithm within $O(T + \log n)$ rounds. Consequently, N steps of the sequential single-site Metropolis chain can be faithfully simulated within $O(N/n + \log n)$ rounds. Our key idea for circumventing the aforementioned barrier for concurrent execution of single-site Markov chains, is to "resolve update in advance": to efficiently execute the Metropolis chain in parallel, we basically implement coin flipping at each node (for evaluating Metropolis filters) before the biases of the coins are fully known.

An important future direction is to have distributed algorithms that can correctly and efficiently simulate general single-site dynamics, beyond Metropolis chains. Our current approach crucially relies on the Metropolis chain, where each update operation is determined by a coin flipping.

For example, consider the heat-bath Glauber dynamics for proper q-coloring of graph $G = (V, E)$. In its continuous-time version, each node $v \in V$ is associated with an i.i.d. rate 1 Poisson clock, such that when a clock at a node v rings:

- $Y(v)$ is updated instantly to a uniform random available color in $[q] \setminus \{Y(u) \mid \{u, v\} \in E\}$.

A concrete open question is to have a distributed algorithm that correctly simulate this continuous chain up to time T within $O(T + \log n)$ or even just $O(\mathrm{polylog}(n) \cdot T)$ rounds, when $q > \Delta$.

More generally, a fundamental problem is to give parallel and distributed simulation of general single-site dynamics that is efficient in the above sense.

References

[AAG+12] Amr Ahmed, Moahmed Aly, Joseph E. 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.

[Awe85] Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM (JACM)*, 32(4):804–823, 1985.

[DDJ18] 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.

[DSOR16] 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.

[DSZOR15] 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.

[DVMGK09] 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.

[FG18] 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.

[FHY18] Weiming Feng, Thomas P. Hayes, and Yitong Yin. Distributed symmetry breaking in sampling (optimal distributed randomly coloring with fewer colors). *arXiv preprint arXiv:1802.06953*, 2018.

[FSY17] 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.

[FVY19] Weiming Feng, Nisheeth Vishnoi, and Yitong Yin. Dynamic sampling from graphical models. In *Proceedings of the 51st ACM Symposium on Theory of Computing (STOC)*, 2019.

[Gla63] Roy J. Glauber. Time-dependent statistics of the Ising model. *Journal of mathematical physics*, 4(2):294–307, 1963.

[GLGG11] 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.

[GŠV15] 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.

[HS07] 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.

[Jon02] Johan Jonasson. Uniqueness of uniform random colorings of regular trees. *Statistics & Probability Letters*, 57(3):243–248, 2002.

[KF09] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques.* MIT press, 2009.

[KKSP18] 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.

[LP17] David A. Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017.

[Lyn96] Nancy A. Lynch. *Distributed algorithms.* Elsevier, 1996.

[MM09] Marc Mezard and Andrea Montanari. *Information, physics, and computation.* Oxford University Press, 2009.

[NASW07] 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.

[NO08] Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *Proceedings of the 49th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 327–336. IEEE, 2008.

[Sly10] 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.

[SN10] Alexander Smola and Shravan Narayanamurthy. An architecture for parallel topic models. *Proceedings of the VLDB Endowment*, 3(1-2):703–710, 2010.

[SS14] Allan Sly and Nike Sun. Counting in two-spin models on d-regular graphs. *The Annals of Probability*, 42(6):2383–2416, 2014.

[SST14] 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.

[SWA09] 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.

[Wei06] 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.

[YXQ09] 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.