Scalable MCMC Sampling for Nonsymmetric Determinantal Point Processes

Insu Han [1] **Mike Gartrell** [2] **Elvis Dohmatob** [3] **Amin Karbasi** [1]

Abstract

A determinantal point process (DPP) is an elegant model that assigns a probability to every subset of a collection of n items. While conventionally a DPP is parameterized by a symmetric kernel matrix, removing this symmetry constraint, resulting in nonsymmetric DPPs (NDPPs), leads to significant improvements in modeling power and predictive performance. Recent work has studied an approximate Markov chain Monte Carlo (MCMC) sampling algorithm for NDPPs restricted to size-k subsets (called k-NDPPs). However, the runtime of this approach is quadratic in n, making it infeasible for large-scale settings. In this work, we develop a scalable MCMC sampling algorithm for k-NDPPs with low-rank kernels, thus enabling runtime that is sublinear in n. Our method is based on a state-of-the-art NDPP rejection sampling algorithm, which we enhance with a novel approach for efficiently constructing the proposal distribution. Furthermore, we extend our scalable k-NDPP sampling algorithm to NDPPs without size constraints. Our resulting sampling method has polynomial time complexity in the rank of the kernel, while the existing approach has runtime that is exponential in the rank. With both a theoretical analysis and experiments on real-world datasets, we verify that our scalable approximate sampling algorithms are orders of magnitude faster than existing sampling approaches for k-NDPPs and NDPPs.

1. Introduction

Determinantal Point Processes (DPPs) are probability distributions defined on the set of all subsets of a collection of n items. They have been applied to a variety of fundamental machine learning problems, including robustness learning (Pang et al., 2019), reinforcement learning (Yang et al., 2020), and bandit optimization (Kathuria et al., 2016), among many others. While conventionally a DPP is parameterized by a symmetric kernel matrix, Gartrell et al. (2019) showed that any nonsymmetric and positive semidefinite matrix can define a valid DPP, which they refer to as a nonsymmetric DPP (NDPP). In addition, they established a number of useful properties of NDPPs. For example, NDPPs are able to capture both positive and negative correlations among items, while symmetric DPPs can only represent negative correlations, leading to significant improvements in modeling power and predictive performance.

Recent works have proposed efficient algorithms for various NDPP tasks, including learning (Gartrell et al., 2021), MAP inference (Anari & Vuong, 2021), and sampling (Han et al., 2022), where the NDPP kernel is given by a low-rank factorization. In this paper we focus on developing an efficient sampling algorithm for NDPPs restricted to size k subsets, called k-NDPPs. Such size-constrained DPPs are often more practical in applications such as video summarization (Sharghi et al., 2018), mini-batch optimization (Zhang et al., 2017), document summarization (Dupuy & Bach, 2018) and coreset sampling (Tremblay et al., 2019). The only existing approach for k-NDPP sampling is an approximate method based on Markov chain Monte Carlo (MCMC) sampling (Alimohammadi et al., 2021; Anari & Vuong, 2021). The algorithm is based on a random walk, where in every iteration a pair of items is exchanged with some probability. These prior works primarily focused on the number of iterations required for convergence, and proved that with time polynomial in k, the sampling algorithm converges to the k-NDPP target distribution. However, each transition step needs time quadratic in n, making this approach infeasible for large-scale settings.

1.1. Contributions

In this work, we develop a scalable MCMC sampling algorithm for k-NDPPs with low-rank kernels. In particular, we accelerate the transition step of the MCMC sampling algorithm so that it runs in sublinear (polynomial-logarithmic) time in n. We first show that this step is equivalent to sampling a subset of size 2 from a conditional NDPP. To achieve fast 2-NDPP sampling, we make use of a state-

[1]Yale University [2]Criteo AI Lab, Paris, France [3]Facebook AI Lab, Paris, France. Correspondence to: Insu Han <insu.han@yale.edu>, Mike Gartrell <m.gartrell@criteo.com>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

Table 1. Summary of recent NDPP sampling algorithms. The sampling time of Han et al. (2022)'s work assumes an orthogonal constraint on the kernel. Here, n is the size of ground set, d is the rank of the kernel, k refers to the size of sampled set ($k \leq d \ll n$) and $\alpha > 0$ is a data-dependent factor. Alimohammadi et al. (2021) showed that $t_{\text{iter}} = \text{poly}(k)$ guarantees the convergence of MCMC sampling, where $\kappa > 0$ is a condition number of the NDPP kernel component (see Proposition 5 for details).

Algorithm	Task	Preprocessing Time	Sampling Time
Cholesky-based Exact (Poulson, 2020)	NDPP	–	$\mathcal{O}(n \cdot d^2)$
Rejection-based Exact (Han et al., 2022)	NDPP	$\mathcal{O}(n \cdot d^2)$	$\mathcal{O}((\log n \cdot k^3 + k^4 + d) \cdot (1+\alpha)^d)$
Naïve MCMC (Alimohammadi et al., 2021)	k-NDPP / NDPP	–	$\mathcal{O}(n^2 \cdot k^3 \cdot t_{\text{iter}})$
Scalable MCMC (**This work**)	k-NDPP / NDPP	$\mathcal{O}(n \cdot d^2)$	$\mathcal{O}((\log n \cdot d^2 + d^3) \cdot (1+\kappa)^2 \cdot t_{\text{iter}})$

of-the-art NDPP rejection sampling algorithm (Han et al., 2022), which we enhance with a novel approach for efficiently constructing a symmetric DPP this is used for the proposal distribution. When the NDPP kernel is given by a rank-d factorization ($d \ll n$), the proposal DPP kernel can be constructed in only $\mathcal{O}(d^3)$ time. This type of proposal kernel is similar to a personalized version of the DPP kernel (Gillenwater et al., 2019; Han & Gillenwater, 2020), which consists of a global features matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and a personalization matrix $\boldsymbol{U} \in \mathbb{R}^{d \times d}$. This proposal distribution changes in every transition of the MCMC sampling, however according to our construction it suffices to update only the matrix $\boldsymbol{U}$. This allows us to utilize a fast tree-based DPP sampling algorithm (Gillenwater et al., 2019) suitable for personalized DPPs. This tree-based algorithm requires us to build a binary tree based on the global features, as a one-time preprocessing step. After preprocessing, the sampling algorithm runs in time that is logarithmic in n. This makes a single iteration of the rejection sampling much faster. We further prove that the number of rejections does not depend on the dimensions of the NDPP kernel, but on some spectral bounds of the kernel. As a consequence, our MCMC sampling algorithm for k-NDPPs runs in logarithmic time in n, and polynomial time in both d and k. To the best of our knowledge, this is the first work on a sublinear time algorithm for k-NDPP sampling. In our experiments, we observe that our proposed algorithm runs orders of magnitude faster than the existing sampling approach, which for kernels learned from some datasets does not terminate within 10 days.

Furthermore, we extend our sampling algorithm to size-unconstrained NDPPs. The resulting algorithm has polynomial time complexity in the rank d of the kernel, while the existing sampling algorithm for NDPPs (Han et al., 2022) has runtime that is exponential in d. Through theoretical analysis and experiments on real-world datasets, we show that our approximate sampling algorithm is orders of magnitude faster than the fastest existing sampling approach for k-NDPPs, and up to an order of magnitude faster for NDPPs. The source code for our NDPP sampling algorithms is publicly available at `https://github.com/insuhan/ndpp-mcmc-sampling`.

1.2. Related Work

Fast sampling algorithms for symmetric DPPs have been extensively studied, including a tree-based algorithm (Gillenwater et al., 2019), and an intermediate sampling method (Derezinski, 2019). These methods commonly require a one-time preprocessing step, with the subsequent sampling procedure running in time that is sublinear in the size of the ground set n. Celis et al. (2017) studied a polynomial time sampling algorithm under partition constraints. For unconstrained-size NDPP sampling, Poulson (2020) developed the Cholesky-based sampling algorithm, which runs in time $\mathcal{O}(n^3)$ for general kernels. Recently, Han et al. (2022) showed that with a rank-d kernel decomposition, the runtime of the Cholesky-based algorithm can be reduced to $\mathcal{O}(nd^2)$. Moreover, they propose a tree-based rejection sampling algorithm for NDPPs that combines previous work for fast sampling of symmetric DPPs with an efficient approach for constructing the proposal distribution. However, although the sampling process has runtime that is sublinear in n, they show that the average number of rejections is exponential in d, which can be problematic in general. For k-NDPPs, to the best of our knowledge, there is no prior work on an efficient algorithm for exact sampling. The only existing approach is an approximate MCMC sampling algorithm (Alimohammadi et al., 2021), which has runtime that is quadratic in n. We summarize these k-NDPP and NDPP sampling algorithms in Table 1.

2. Background

Notation. The set of first n positive integers is denoted by $\{1, \dots, n\} \coloneqq [n]$. For a finite set S, we denote by $\binom{S}{k}$ the collection of all k-element subsets of a set S. We use $\boldsymbol{I}_d$ for the d-by-d identity matrix and drop the subscript when it is clear from the context. For a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ and indices $A \subseteq [m], B \subseteq [n]$, we use $\boldsymbol{X}_{A,B} \in \mathbb{R}^{|A| \times |B|}$ to denote a submatrix of $\boldsymbol{X}$ whose rows and columns are indexed by A and B, respectively. We write $\boldsymbol{X}_{:,B} \coloneqq \boldsymbol{X}_{[m],B}$ to denote all rows of $\boldsymbol{X}$, and similarly $\boldsymbol{X}_{A,:} \coloneqq \boldsymbol{X}_{A,[n]}$ for all columns of $\boldsymbol{X}$. We denote the largest and smallest singular values of $\boldsymbol{X}$ by $\sigma_{\max}(\boldsymbol{X})$ and $\sigma_{\min}(\boldsymbol{X})$, respectively. We use $\succeq$ to denote the Loewner order, i.e., $\boldsymbol{A} \succeq \boldsymbol{B}$ implies $\boldsymbol{A} - \boldsymbol{B}$ is

positive semidefinite (PSD), and Diag to denote the direct sum, i.e., $\mathrm{Diag}(\boldsymbol{A}, \boldsymbol{B}) = \left[\begin{smallmatrix} \boldsymbol{A} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{B} \end{smallmatrix}\right]$.

2.1. Nonsymmetric DPPs

Given a matrix $\boldsymbol{L} \in \mathbb{R}^{n \times n}$, a DPP assigns a probability

$$\mathcal{P}_{\boldsymbol{L}}(S) \propto \det(\boldsymbol{L}_S) \tag{1}$$

to every subset S of $[n]$. Any symmetric and PSD kernel $\boldsymbol{L}$ guarantees that $\det(\boldsymbol{L}_S)$ is nonnegative, and therefore admits a DPP. Gartrell et al. (2019) extended the space of valid DPP kernels and proved that nonsymmetric and PSD $\boldsymbol{L}$ kernels (i.e., $\boldsymbol{L} + \boldsymbol{L}^\top \succeq 0$) can be also used to define a DPP. An important property of NDPPs is that they can capture both positive and negative correlations, while symmetric DPPs only capture the negative correlations, resulting in significant improvements in modeling power and predictive performance for NDPPs.

In particular, Gartrell et al. (2019) proposed a kernel construction for NDPPs that combines a symmetric PSD matrix and a skew-symmetric matrix:

$$\boldsymbol{L} = \boldsymbol{V}\boldsymbol{V}^\top + \boldsymbol{B}(\boldsymbol{D} - \boldsymbol{D}^\top)\boldsymbol{B}^\top, \tag{2}$$

where $\boldsymbol{V} \in \mathbb{R}^{n \times d_1}, \boldsymbol{B} \in \mathbb{R}^{n \times d_2}$, and $\boldsymbol{D} \in \mathbb{R}^{d_2 \times d_2}$. For simplicity, we will write $\boldsymbol{X} := [\boldsymbol{V},\ \boldsymbol{B}] \in \mathbb{R}^{n \times d}$, $\boldsymbol{W} := \mathrm{Diag}(\boldsymbol{I}, \boldsymbol{D} - \boldsymbol{D}^\top) \in \mathbb{R}^{d \times d}$ for $d = d_1 + d_2$, and $\boldsymbol{L} = \boldsymbol{X}\boldsymbol{W}\boldsymbol{X}^\top$.

We refer to a k-NDPP as a NDPP whose support is restricted to size-k subsets of $[n]$.[1] As studied in Kulesza & Taskar (2011, Proposition 5.1), the normalization constant of a k-NDPP can be computed using the eigenvalues of $\boldsymbol{L}$.[2] Formally, when $\{\lambda_i\}_{i=1}^d$ are the nonzero eigenvalues of the rank-d matrix $\boldsymbol{L}$, it holds that

$$\sum_{S \in \binom{[n]}{k}} \det(\boldsymbol{L}_S) = \sum_{S \in \binom{[d]}{k}} \prod_{i \in S} \lambda_i := e_k(\{\lambda_i\}_{i=1}^d), \tag{3}$$

where e_k is known as the k-th *elementary symmetric polynomial*. Note that $\{\lambda_i\}_{i=1}^d$ are also eigenvalues of $\boldsymbol{W}\boldsymbol{X}^\top\boldsymbol{X} \in \mathbb{R}^{d \times d}$, and therefore one can obtain them from matrix-matrix multiplications and the eigendecomposition, resulting in $\mathcal{O}(nd^2)$ runtime. In addition, Equation (3) can be computed in time $\mathcal{O}(dk)$ using the following recursive relation:

$$e_k(\{\lambda_i\}_{i=1}^d) = e_k(\{\lambda_i\}_{i=1}^{d-1}) + \lambda_d \cdot e_{k-1}(\{\lambda_i\}_{i=1}^{d-1}), \tag{4}$$

where $e_0(\{\lambda_i\}_{i=1}^d) = 1$. Since every determinant of a principal submatrix of $\boldsymbol{L}$ is nonnegative, the e_k's for NDPPs are also nonnegative.

[1] Throughout this paper, we assume that $2 \leq k \leq d \ll n$.

[2] This was originally studied for symmetric DPPs, but can be naturally extended to a nonsymmetric PSD matrix $\boldsymbol{L}$.

Algorithm 1 MCMC Sampling for k-NDPP

1: **Input**: $\boldsymbol{L} \in \mathbb{R}^{n \times n}$, $k \in \mathbb{N}$, $t_{\text{iter}} \in \mathbb{N}$
2: Select $S \in \binom{[n]}{k}$ uniformly at random
3: **for** $t = 1, \ldots, t_{\text{iter}}$ **do**
4: Select $A \in \binom{S}{k-2}$ uniformly at random
5: Select $a, b \in [n]$ with probability $\propto \det(\boldsymbol{L}_{A \cup \{a,b\}})$ (▷ Run Algorithm 2)
6: $S \leftarrow A \cup \{a, b\}$
7: **end for**
8: **Return** S

2.2. MCMC Sampling for k-NDPPs

An MCMC sampling algorithm for a k-DPP begins with a subset S selected from $\binom{[n]}{k}$ uniformly at random, and then iteratively updates S with some probability. For symmetric k-DPPs, single-item-exchange Markov chains (i.e., S is replaced with $S \cup \{i\} \setminus \{j\}$ for $i \notin S, j \in S$ in every iteration) can guarantee fast convergence to the approximate target distribution in total variation distance (Li et al., 2016; Anari et al., 2016; Rezaei & Gharan, 2019). However, the single-item-exchange chain does not mix well for k-NDPPs because they are not negatively dependent, which is a key requirement for fast mixing (Anari et al., 2016).

Recent work has shown that when a pair of items S is exchanged, the chain can quickly converge to the target k-NDPP distribution (Anari & Vuong, 2021; Alimohammadi et al., 2021). We provide pseudo-code for this MCMC algorithm in Algorithm 1.

Alimohammadi et al. (2021) proved that the mixing time, i.e., the minimum number of iterations required to approximate the target distribution within ε in terms of total variation distance, is bounded by a polynomial in k.

Proposition (Theorem 11 in (Alimohammadi et al., 2021))**.** *For any $\varepsilon > 0$, a sample S obtained from Algorithm 1 with*

$$t_{\text{iter}} = \mathcal{O}\left(k^2 \cdot \log\left(\frac{1}{\varepsilon \cdot \Pr(S_0)}\right)\right), \tag{5}$$

and randomly chosen subset $S_0 \in \binom{[n]}{k}$, the total variation distance to the target k-NDPP distribution is guaranteed to be less than ε.

Note that each iteration of MCMC sampling (line 5 in Algorithm 1) needs to compute determinants of k-by-k matrices for $\mathcal{O}(n^2)$ candidates, and therefore runs in time $\mathcal{O}(n^2k^3)$. We call this step the "up operator".

3. Scalable MCMC Sampling for k-NDPPs

As mentioned above, the naïve up operator, which involves an exhaustive search over the space of possible candidates,

requires time complexity that is quadratic in the ground set size n. This runtime clearly suffers from scalability issues for large n. In this section, we show how to significantly accelerate the up operator by utilizing the low-rank structure of the kernel matrix.

We first observe that the up operator is equivalent to sampling a size 2 subset from the NDPP conditioned on A. Formally, given a low-rank NDPP kernel $\boldsymbol{L} = \boldsymbol{X}\boldsymbol{W}\boldsymbol{X}^\top$ for $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, $\boldsymbol{W} \in \mathbb{R}^{d\times d}$, and a subset $A \subseteq [n]$, $|A| \leq d$, one can check that $\det\left(\boldsymbol{L}_{A\cup\{a,b\}}\right) \propto \det(\boldsymbol{L}^A_{\{a,b\}})$, where $\boldsymbol{L}^A$ is the kernel of the conditional NDPP on A, given by

$$\boldsymbol{L}^A := \boldsymbol{L} - \boldsymbol{L}_{:,A}\boldsymbol{L}_A^{-1}\boldsymbol{L}_{A,:} = \boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top, \tag{6}$$

where $\boldsymbol{W}^A := \boldsymbol{W} - \boldsymbol{W}\boldsymbol{X}_{A,:}^\top(\boldsymbol{X}_{A,:}\boldsymbol{W}\boldsymbol{X}_{A,:}^\top)^{-1}\boldsymbol{X}_{A,:}\boldsymbol{W}$. Note that computing $\boldsymbol{W}^A$ requires a matrix inversion of dimension $|A| \leq d$ and matrix-matrix multiplications of dimension d, which results in $\mathcal{O}(d^3)$ operations in total. Therefore, the up operator can be seen as sampling a size 2 subset from the NDPP with kernel $\boldsymbol{L}^A$. In the next section, we present an approach for efficiently sampling from this conditional 2-NDPP.

3.1. Up Operator via Rejection Sampling

Our goal is an efficient sampling from a 2-NDPP whose kernel is given by Equation (6). To this end, we utilize recent work on a sublinear-time NDPP rejection sampling algorithm (Han et al., 2022).

Specifically, given a NDPP kernel $\boldsymbol{L}^A$, assume that there exists a matrix $\widehat{\boldsymbol{L}}$ such that

$$\det(\boldsymbol{L}^A_S) \leq \det(\widehat{\boldsymbol{L}}_S) \tag{7}$$

for every $S \subseteq [n]$. The rejection sampling method proceeds as follows: first, draw a sample S from the DPP with kernel $\widehat{\boldsymbol{L}}$ and accept it with probability $\det(\boldsymbol{L}^A_S)/\det(\widehat{\boldsymbol{L}}_S)$, otherwise repeat the draws until S is accepted. The resulting sample S has probability proportional to $\det(\boldsymbol{L}^A_S)$. The distribution from which we actually draw a sample (i.e., the DPP with $\widehat{\boldsymbol{L}}$) is called the *proposal distribution*. Furthermore, if $\widehat{\boldsymbol{L}}$ is symmetric, one can make use of several symmetric DPP sampling algorithms. In particular, we adopt a sublinear-time tree-based method (Gillenwater et al., 2019) for our scalable MCMC sampling algorithm, which we describe in more detail in Section 3.2.

Han et al. (2022) provided a proposal distribution with kernel $\widehat{\boldsymbol{L}}$, based on a spectral decomposition of $\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top$, and shows that it satisfies Equation (7). When $\boldsymbol{L}$ is given by a rank-d factorization, this spectral decomposition has a runtime of $\mathcal{O}(nd^2)$. However, this complexity makes the cost of the preprocessing steps for the sampler dominant when the subsequent sampling from the DPP with $\widehat{\boldsymbol{L}}$ is performed in sublinear-time in n (e.g., using tree-based sampling). Thus, we would not fully utilize the advantages of a scalable DPP sampling algorithm. We resolve this issue by developing a more efficient procedure for constructing the proposal DPP.

Algorithm 2 Up Operator via Rejection Sampling

Input: $A \subseteq [n]$, $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, $\boldsymbol{W} \in \mathbb{R}^{d\times d}$
$\boldsymbol{W}^A \leftarrow \boldsymbol{W} - \boldsymbol{W}\boldsymbol{X}_{A,:}^\top(\boldsymbol{X}_{A,:}\boldsymbol{W}\boldsymbol{X}_{A,:}^\top)^{-1}\boldsymbol{X}_{A,:}\boldsymbol{W}$
$\{(\sigma_i, \boldsymbol{y}_i, \boldsymbol{z}_i)\}_{i=1}^{d/2} \leftarrow$ Youla decomp. of $\frac{\boldsymbol{W}^A - \boldsymbol{W}^{A\top}}{2}$
$\widehat{\boldsymbol{W}}^A \leftarrow \frac{\boldsymbol{W}^A + \boldsymbol{W}^{A\top}}{2} + \sum_{i=1}^{d/2}\sigma_i\left(\boldsymbol{y}_i\boldsymbol{y}_i^\top + \boldsymbol{z}_i\boldsymbol{z}_i^\top\right)$
while true **do**
  Sample $\{a, b\}$ with prob. $\propto \det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_{\{a,b\}})$ (▷ Run Algorithm 3)
  if $\mathcal{U}([0,1]) \leq \frac{\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_{\{a,b\}})}{\det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_{\{a,b\}})}$ **then**
    Return $\{a, b\}$
  end if
end while

Our key idea is to apply a similar spectral decomposition approach for computing the d-by-d matrix $\boldsymbol{W}^A$, which allows us to compute the proposal DPP kernel in time $\mathcal{O}(d^3)$. More specifically, we begin with writing the spectral decomposition of the skew-symmetric matrix $\frac{\boldsymbol{W}^A - \boldsymbol{W}^{A\top}}{2}$ as

$$\frac{\boldsymbol{W}^A - \boldsymbol{W}^{A\top}}{2} = \sum_{i=1}^{d/2}\begin{bmatrix}\boldsymbol{y}_i & \boldsymbol{z}_i\end{bmatrix}\begin{bmatrix}0 & \sigma_i \\ -\sigma_i & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{y}_i^\top \\ \boldsymbol{z}_i^\top\end{bmatrix}, \tag{8}$$

where $\{\boldsymbol{y}_i, \boldsymbol{z}_i\}_{i=1}^{d/2}$ is a set of eigenvectors, and the σ_i's are the nonnegative eigenvalues. The above decomposition is also known as the Youla decomposition (Youla, 1961). Given this, we define a symmetric matrix $\widehat{\boldsymbol{W}}^A$ as follows:

$$\widehat{\boldsymbol{W}}^A := \frac{\boldsymbol{W}^A + \boldsymbol{W}^A}{2} + \sum_{i=1}^{d/2}\begin{bmatrix}\boldsymbol{y}_i & \boldsymbol{z}_i\end{bmatrix}\begin{bmatrix}\sigma_i & 0 \\ 0 & \sigma_i\end{bmatrix}\begin{bmatrix}\boldsymbol{y}_i^\top \\ \boldsymbol{z}_i^\top\end{bmatrix}. \tag{9}$$

An important property is that every determinant of a principal submatrix of $\widehat{\boldsymbol{W}}^A$ is equal to or greater than that of $\boldsymbol{W}^A$, i.e., $\det(\boldsymbol{W}^A_S) \leq \det(\widehat{\boldsymbol{W}}^A_S)$ for all $S \subseteq [d]$, as shown in (Han et al., 2022, Theorem 1). We further prove that this property is preserved under the bilinear transformation $\boldsymbol{W}^A \to \boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top$ for any $\boldsymbol{X} \in \mathbb{R}^{n\times d}$.

Theorem 1. *Given $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ and $\boldsymbol{W}^A \in \mathbb{R}^{d\times d}$, suppose $\widehat{\boldsymbol{W}}^A$ is obtained from Equation (9) with $\boldsymbol{W}^A$. Then,*

$$\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_S) \leq \det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_S) \tag{10}$$

for every $S \subseteq [n]$. In addition, equality holds when $|S| \geq d$.

We provide the proof of Theorem 1 in Appendix C.1. Theorem 1 allows us to use rejection sampling, with the kernel $\widehat{\boldsymbol{L}} := \boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top$ as the proposal distribution.

Pseudo-code for the up operator computed using rejection sampling is shown in Algorithm 2. Observe that $\widehat{\boldsymbol{W}}^A$ can be computed in time $\mathcal{O}(d^3)$ (lines 2-4 in Algorithm 2), because both matrix operations involve matrices with dimension d, and the Youla decomposition of $\frac{\boldsymbol{W}^A - \boldsymbol{W}^{A\top}}{2}$, have complexities $\mathcal{O}(d^3)$. Therefore, we can build each kernel component for the proposal DPP in time $\mathcal{O}(d^3)$. This improves the previous method with runtime $\mathcal{O}(nd^2)$, since $d \ll n$, and potentially allows us to utilize the sublinear-time sampling algorithm. In the next section, we discuss the tree-based k-DPP sampling algorithm that uses our proposal DPP.

3.2. Sublinear-time Tree-based Sampling

We now focus on sampling the 2-DPP with kernel $\widehat{\boldsymbol{L}} = \boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top$ (line 6 in Algorithm 2). Observe that the matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ remains unchanged, and only the inner matrix $\widehat{\boldsymbol{W}}^A \in \mathbb{R}^{d\times d}$ changes in every iteration of the MCMC sampling algorithm. Fortunately, the sublinear-time tree-based DPP sampling algorithm (Gillenwater et al., 2019) is well suited to this type of kernel structure. We build a binary tree using $\boldsymbol{X}$, which can be used for 2-DPP sampling with the kernel $\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top$, and then the sampling process is equivalent to k tree traversals with a d-by-d query matrix. Consequently, 2-DPP sampling can be done in time $\mathcal{O}(d^2 \log n + d^3)$.

We begin by explaining the workflow for tree-based k-DPP sampling, where we set k to 2. Formally, denote $\boldsymbol{U} := (\widehat{\boldsymbol{W}}^A)^{-\frac{1}{2}}$, and let $\{(\lambda_i, \boldsymbol{v}_i)\}_{i=1}^d$ be the eigendecomposition of $\boldsymbol{U}(\boldsymbol{X}^\top\boldsymbol{X})\boldsymbol{U}$. From (Kulesza & Taskar, 2012, Eq. (187)), the probability of sampling $S \in \binom{[n]}{k}$ from the k-DPP with $\widehat{\boldsymbol{L}}$ can be decomposed into the following

$$\frac{\det(\widehat{\boldsymbol{L}}_S)}{e_k(\{\lambda_i\}_{i=1}^d)} = \sum_{E\in\binom{[d]}{k}} \frac{\prod_{i\in E}\lambda_i}{e_k(\{\lambda_i\}_{i=1}^d)} \cdot \det(\boldsymbol{K}^E_S), \quad (11)$$

where $\boldsymbol{K}^E := \boldsymbol{X}\boldsymbol{U}\left(\sum_{i\in E}\lambda_i^{-1}\boldsymbol{v}_i\boldsymbol{v}_i^\top\right)\boldsymbol{U}\boldsymbol{X}^\top$, and e_k is the elementary symmetric polynomial defined in Equation (3). We observe that $\boldsymbol{K}^E$ is a rank-k projection matrix, because

$$\boldsymbol{K}^E = \sum_{i\in E} \frac{\boldsymbol{X}\boldsymbol{U}\boldsymbol{v}_i}{\sqrt{\lambda_i}}\left(\frac{\boldsymbol{X}\boldsymbol{U}\boldsymbol{v}_i}{\sqrt{\lambda_i}}\right)^\top, \quad (12)$$

and $\frac{\boldsymbol{X}\boldsymbol{U}\boldsymbol{v}_i}{\sqrt{\lambda_i}}$'s are the eigenvectors of $\widehat{\boldsymbol{L}}$ (Kulesza & Taskar, 2012, Proposition 3.1). Any projection matrix can define a DPP with a marginal kernel, called an *elementary* DPP. Equation (11) allows the following two-step k-DPP sampling procedure: 1) select an index set $E \in \binom{[d]}{k}$ with probability $\frac{\prod_{i\in E}\lambda_i}{e_k(\{\lambda_i\}_{i=1}^d)}$, and then 2) sample a subset S from the elementary DPP with kernel $\boldsymbol{K}^E$. As studied in (Kulesza & Taskar, 2012, Algorithm 8), step 1) can be efficiently performed using the recursive property of e_k introduced in Equation (4), resulting in $\mathcal{O}(dk)$ runtime. Notice that step 2) is a computational bottleneck for k-DPP sampling. However, this step can be accelerated using tree-based sampling, which we describe next.

Algorithm 3 Tree-based k-DPP Sampling

Input: $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, $\widehat{\boldsymbol{W}}^A \in \mathbb{R}^{d\times d}$, $\boldsymbol{C} = \boldsymbol{X}^\top\boldsymbol{X} \in \mathbb{R}^{d\times d}$, tree structure $\mathcal{T}$
$\boldsymbol{U} \leftarrow (\widehat{\boldsymbol{W}}^A)^{-\frac{1}{2}}$
$\{(\boldsymbol{v}_i, \lambda_i)\}_{i=1}^d \leftarrow$ Eigendecomp. of $\boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^\top$
Select size k subset E with prob. $\propto \prod_{i\in E}\lambda_i$ (▷ Run Algorithm 8 in (Kulesza & Taskar, 2012))
$\boldsymbol{Q} \leftarrow \boldsymbol{U}\left(\sum_{i\in E}\lambda_i^{-1}\,\boldsymbol{v}_i\boldsymbol{v}_i^\top\right)\boldsymbol{U}^\top$
$Y \leftarrow \emptyset$
for $j = 1, \ldots, k$ **do**
  Sample a with probability $\left\langle \boldsymbol{Q}, \boldsymbol{x}_a\boldsymbol{x}_a^\top\right\rangle$ using the tree structure $\mathcal{T}$ (▷ Run Algorithm 3 in (Han et al., 2022))
  $Y \leftarrow Y \cup \{a\}$
  $\boldsymbol{Q} \leftarrow \boldsymbol{Q} - \boldsymbol{Q}\boldsymbol{X}_{Y,:}^\top\left(\boldsymbol{X}_{Y,:}\boldsymbol{Q}\boldsymbol{X}_{Y,:}^\top\right)^{-1}\boldsymbol{X}_{Y,:}\boldsymbol{Q}$
end for
Return Y

Specifically, let $S \subseteq [n]$ be a subset that we wish to sample. For any $Y \subseteq [n]$ and $a \notin Y$ observe that

$$\mathcal{P}_{\boldsymbol{K}^E}(a \in S | Y \subseteq S) = \frac{\det(\boldsymbol{K}^E_{Y\cup\{a\}})}{\det(\boldsymbol{K}^E_Y)}$$
$$= \boldsymbol{K}^E_{a,a} - \boldsymbol{K}^E_{a,Y}(\boldsymbol{K}^E_Y)^{-1}\boldsymbol{K}^E_{Y,a} = \left\langle \boldsymbol{Q}^Y, \boldsymbol{x}_a^\top\boldsymbol{x}_a\right\rangle, \quad (13)$$

where $\boldsymbol{Q}^Y := \boldsymbol{M} - \boldsymbol{M}\boldsymbol{X}_{Y,:}^\top(\boldsymbol{X}_{Y,:}\boldsymbol{M}\boldsymbol{X}_{Y,:}^\top)^{-1}\boldsymbol{X}_{Y,:}\boldsymbol{M}$, $\boldsymbol{M} := \boldsymbol{U}\left(\sum_{i\in E}\lambda_i^{-1}\boldsymbol{v}_i\boldsymbol{v}_i^\top\right)\boldsymbol{U}$, and $\boldsymbol{x}_a \in \mathbb{R}^d$ is the a-th row vector in $\boldsymbol{X}$. This implies that we can begin with $Y \leftarrow \emptyset$ and iteratively append a to Y, where a is selected with the probability described in Equation (13). The process of selecting a single element can be done in a divide-and-conquer manner by leveraging a binary tree structure.

We construct a binary tree where the root contains $[n]$ and assigns a partition A_ℓ, A_r of $[n]$ to its left and right nodes. The branching proceeds until n leaf nodes are created. In addition, every non-leaf node contains a d-by-d matrix $\sum_{a\in A}\boldsymbol{x}_a^\top\boldsymbol{x}_a$, where A is the stored subset. Sampling a single element can be done by traversing the tree with the query matrix $\boldsymbol{Q}^Y$. In every non-leaf node containing a subset A, we move down to the left branch with probability

$$\frac{\left\langle \boldsymbol{Q}^Y, \sum_{a\in A_\ell}\boldsymbol{x}_a^\top\boldsymbol{x}_a\right\rangle}{\left\langle \boldsymbol{Q}^Y, \sum_{a\in A}\boldsymbol{x}_a^\top\boldsymbol{x}_a\right\rangle} \quad (14)$$

or otherwise to the right branch, until we reach a leaf node. The tree traversal process is repeated for k iterations, because every subset sampled from the elementary DPP has

exactly k elements. If we construct a binary tree of depth $\mathcal{O}(\log n)$, which requires time $\mathcal{O}(nd^2)$, then step 2) can run in time $\mathcal{O}(kd^2 \log n + k^2 d^2)$. We summarize the tree-based k-DPP sampling in Algorithm 3 and provide the overall runtime in Theorem 2.

Theorem 2. *Given $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, and symmetric and PSD $\widehat{\boldsymbol{W}}^A \in \mathbb{R}^{d \times d}$, Algorithm 3 samples a subset from the k-DPP with kernel $\boldsymbol{L} = \boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top$, and runs in time $\mathcal{O}(kd^2 \log n + k^2 d^2 + d^3)$, after a one-time preprocessing step that runs in time $\mathcal{O}(nd^2)$.*

We provide the proof of Theorem 2 in Appendix C.2. The runtime of our tree-based sampling algorithm improves that of previous work (Gillenwater et al., 2019), which is $\mathcal{O}(k^2 d^2 \log n + d^3)$. We also remark that the binary tree used in (Han et al., 2022) is slightly different from ours. They build a tree using the eigenvectors of the kernel, while our tree structure is based on the non-orthogonal features $\boldsymbol{X}$. This allows our tree to be used for sublinear-time sampling for any DPP with kernel $\boldsymbol{X}\boldsymbol{A}\boldsymbol{X}^\top$, with an arbitrarily symmetric and PSD matrix $\boldsymbol{A} \in \mathbb{R}^{d \times d}$, as is the case for our MCMC-based k-NDPP sampling approach.

We remind the reader that the rejection-based up operator requires sampling from a 2-DPP (line 6 in Algorithm 2). From Theorem 2, sampling from the proposal distribution runs in $\mathcal{O}(d^2 \log n + d^3)$ time. However, as discussed in Section 3.1, this process is repeated until the sample is accepted. In the next section, we examine the average number of rejections in Algorithm 2.

4. Runtime Analysis

We first define the ratio of the largest and smallest singular values of the conditional kernel components, which will affect the average number of rejections.

Definition 3. *Given $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{W} \in \mathbb{R}^{d \times d}$, such that $\boldsymbol{W} + \boldsymbol{W}^\top \succeq 0$ and $A \in \binom{[n]}{k-2}$ for $k \geq 2$, consider $\boldsymbol{W}^A$ as defined in Equation (6). Define*

$$\kappa_A := \frac{\sigma_{\max}(\boldsymbol{W}^A - \boldsymbol{W}^{A^\top})}{\min_{Y \in \binom{[n] \setminus A}{2}} \sigma_{\min}([\boldsymbol{X}(\boldsymbol{W}^A + \boldsymbol{W}^{A^\top})\boldsymbol{X}^\top]_Y)}.$$

and $\kappa := \max_{A \subseteq [n], |A| \leq d-2} \kappa_A$.

We now provide an upper bound on the average number of rejections in Algorithm 2.

Theorem 4. *Given $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{W} \in \mathbb{R}^{d \times d}$, such that $\boldsymbol{W} + \boldsymbol{W}^\top \succeq 0$ and $A \in \binom{[n]}{k-2}$ for $k \geq 2$, consider κ_A as in Definition 3. Then, the average number of rejections of the rejecion-based up operator (Algorithm 2) is no greater than $(1 + \sigma_{\max}(\boldsymbol{X})^2\ \kappa_A)^2$.*

Proof Sketch. First, we observe that the average number of rejections can be expressed as

$$\frac{\sum_{Y \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_Y)}{\sum_{Y \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_Y)}. \tag{15}$$

Instead of bounding the above directly, we consider $\max_{Y \in \binom{[n] \setminus A}{2}} \frac{\det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_Y)}{\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_Y)}$, which upper bounds Equation (15). In addition, observing that the denominator is no less than $\sum_{Y \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}(\frac{\boldsymbol{W}^A + \boldsymbol{W}^{A^\top}}{2})\boldsymbol{X}^\top]_Y)$, we can derive the bound as a determinant of a 2-by-2 symmetric and PSD matrix. This can be bounded by the singular values of the kernel. A full proof is provided in Appendix C.3. □

We observe that the matrices in the numerator and denominator of the factor κ_A in Definition 3 are bounded by the largest and smallest eigenvalues among some 2-by-2 matrices (see Equation (39) in Appendix C.3). There is no dependency on d here, and therefore the number of rejections does not depend on either n or d. In Section 6.3, we empirically verify that the actual rejection numbers are very small compared to n both for synthetic and real-world datasets. For example, for the Book recommendation dataset with $n \simeq 10^6$ (Wan & McAuley, 2018), we see only 3 rejections on average. This makes our rejection-based MCMC sampling algorithm practical for NDPPs with large n.

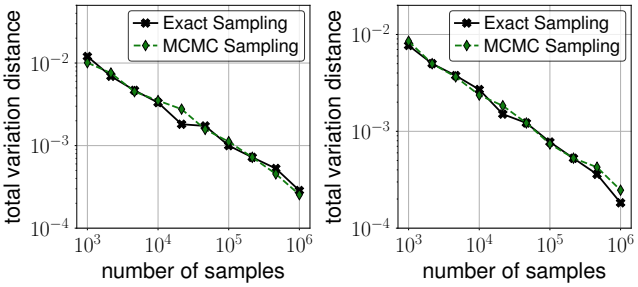
Putting all of the above together, we provide the overall runtime for our MCMC sampling algorithm for k-NDPPs in the following proposition.

Proposition 5. *Given $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{W} \in \mathbb{R}^{d \times d}$, such that $\boldsymbol{W} + \boldsymbol{W}^\top \succeq 0$ and $k \geq 2$, consider κ as defined in Definition 3. With a preprocessing step that runs in time $\mathcal{O}(nd^2)$, Algorithm 1 runs in time $\mathcal{O}(t_{\text{iter}}\ (1 + \sigma_{\max}(\boldsymbol{X})^2\ \kappa)^2\ (d^2 \log n + d^3))$ in expectation.*

The proof of Proposition 5 can be found in Appendix C.4. Note that the size k only affects the number of MCMC iterations t_{iter}, since each transition of the MCMC algorithm requires sampling from a 2-NDPP. Moreover, as mentioned in Section 2.2, $t_{\text{iter}} = \mathcal{O}(k^2 \log \frac{1}{\varepsilon \Pr(S_0)})$ guarantees convergence. Therefore, our MCMC algorithm runs in time that is sublinear in n and polynomial in both k and d. In Section 6.1, we compare the MCMC sampling algorithm (Algorithm 1) to the exact sampler by empirically evaluating the total variation (TV) distance to the ground-truth distribution. We observe that the TV distance of the MCMC sampler with $t_{\text{iter}} = k^2$ decreases as fast as the exact sampler when the number of samples increases.

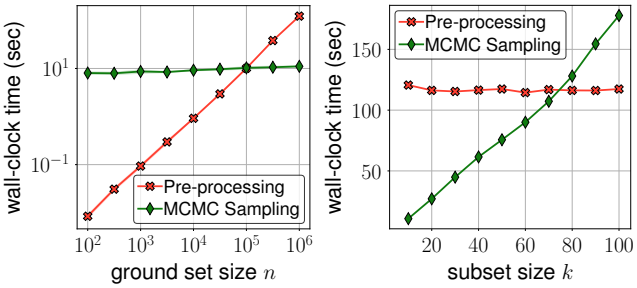
5. Extension from k-NDPPs to Unconstrained NDPPs

In this section we show that any k-NDPP sampling algorithm can be transformed into an unconstrained-size NDPP



(a) k-NDPP (b) NDPP

Figure 1. Total variation distance between the exact sampler and our proposed MCMC sampler for (a) a k-NDPP and (b) a unconstrained-size NDPP. We use synthetically-generated NDPP kernels with $n = 10, d = 8, k = 5$, and set $t_{\text{iter}} = 25$ for our MCMC algorithm.



(a) (b)

Figure 2. Wall-clock runtime for the preprocessing and sampling steps of our scalable MCMC algorithm, for k-NDPPs with synthetic kernels. In (a) we vary $n \in \{10^2, \ldots, 10^6\}$ and set $k = 10$, and in (b) vary $k \in \{10, \ldots, 100\}$, and set $n = 10^6$.

Algorithm 4 MCMC Sampling for NDPP

Input: $\boldsymbol{X} \in \mathbb{R}^{n \times d}, \boldsymbol{W} \in \mathbb{R}^{d \times d}$
$\{(\lambda_i, \boldsymbol{v}_i)\}_{i=1}^d \leftarrow$ Eigendecomp. of $\boldsymbol{W}\boldsymbol{X}^\top\boldsymbol{X}$
Compute elementary symmetric polynomials $\{e_k\}_{k=0}^d$ of $\{\lambda_i\}_{i=1}^d$ (▷ Run Algorithm 7 in (Kulesza & Taskar, 2012))
Sample $k \in \{0, 1, \ldots, d\}$ with prob. $\propto e_k$
Compute t_{iter} with the chosen k (e.g., $t_{\text{iter}} = k^2$)
Construct a binary tree $\mathcal{T}$ with $\boldsymbol{X}$
$S \leftarrow$ Run Algorithm 1 with $\mathcal{T}, \boldsymbol{X}, \boldsymbol{W}, k, t_{\text{iter}}$
Return S

sampling algorithm, with a marginal cost for preprocessing. A simple approach for using a k-NDPP sampler to perform NDPP sampling consists of two steps: 1) first, sample a random variable $k \in \{0, 1, \ldots, d\}$ with probability proportional to the normalization constant of the k-NDPP, and then 2) run any k-NDPP sampling algorithm with the chosen k. From Equation (3), the normalization constant of a k-NDPP is equal to the k-th elementary symmetric polynomial $e_k(\{\lambda_i\}_{i=1}^d)$, where $\{\lambda_i\}_{i=1}^d$ are the nonzero eigenvalues of the rank-d kernel. Once we obtain the eigenvalues in $\mathcal{O}(nd^2)$ time, the corresponding e_k's can be computed in $\mathcal{O}(dk)$ time using Equation (4). The MCMC sampling algorithm for NDPPs is outlined in Algorithm 4.

We consider the computation of the e_k's as a preprocessing step, because we re-use these values for drawing subsequent NDPP samples. The runtime complexity of this preprocessing step is $\mathcal{O}(nd^2)$, which is equivalent to the runtime complexity of preprocessing for our sublinear-time MCMC sampling algorithm for k-NDPPs.

We describe the overall runtime of Algorithm 4 in the following proposition.

Proposition 6. *Given $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{W} \in \mathbb{R}^{d \times d}$, such that $\boldsymbol{W} + \boldsymbol{W}^\top \succeq 0$, consider κ as in Definition 3. With a preprocessing step that runs in time $\mathcal{O}(nd^2)$, Algorithm 4 runs in time $\mathcal{O}(t_{\text{iter}}\,(1+\sigma_{\max}(\boldsymbol{X})^2\,\kappa)^2\,(d^2 \log n + d^3))$ in expectation.*

Previous work on exact NDPP sampling (Han et al., 2022) also has runtime that is sublinear in n. However, their algorithm has runtime that is exponential in the rank of kernel d (see Theorem 2 therein). In contrast, our MCMC-based approximate sampling algorithm runs in time polynomial in d, because of $t_{\text{iter}} = \widetilde{\mathcal{O}}(d^2)$. Such a gap makes our approximate MCMC sampler feasible to run in cases where the exact sampler does not terminate for several days in some real-world settings; see Section 6.3 for details.

6. Experiments

In this section, we report empirical results for our experiments involving several NDPP sampling algorithms, for NDPPs with and without size constraints.

6.1. Convergence of MCMC Sampling

We first benchmark our MCMC sampling algorithm and compare it to the exact sampler for both k-NDPPs and unconstrained-size NDPPs. We randomly generate $\boldsymbol{V}, \boldsymbol{B} \in \mathbb{R}^{n \times d/2}$, where each entry is sampled from $\mathcal{N}(0, \sqrt{2/d})$; $\boldsymbol{D} \in \mathbb{R}^{d/2 \times d/2}$, where each entry is sampled from $\mathcal{N}(0, 1)$; and then construct the NDPP kernel as $\boldsymbol{L} = \boldsymbol{V}\boldsymbol{V}^\top + \boldsymbol{B}(\boldsymbol{D} - \boldsymbol{D}^\top)\boldsymbol{B}^\top$. We collect samples from each sampling algorithm and evaluate the empirical total variation (TV) distance, i.e., $\max_S |p(S) - q(S)|$, where p and q correspond to the ground-truth and empirical distributions from the samplers, respectively. We set $n = 10, d = 8, k = 5$, and draw up to 10^6 random samples from each sampler. For our MCMC algorithm, we set $t_{\text{iter}} = k^2$. The results are shown in

Table 2. Number of rejections and runtime (in seconds), for sampling and preprocessing, for k-NDPP and unconstrained-size NDPP sampling algorithms. Runtimes in the top three rows report sampling times, and the bottom row shows the preprocessing times of our MCMC algorithm. Bold values indicate the fastest runtimes, and $(*)$ indicates the expected results for those cases where the sampling algorithm does not terminate within a feasible timeframe.

Task	Metric	Algorithm	**UK Retail** $n = 3{,}941$	**Recipe** $n = 7{,}993$	**Instacart** $n = 49{,}677$	**Million Song** $n = 371{,}410$	**Book** $n = 1{,}059{,}437$
$k = 10$	Runtime	Exact (Rejection)	406	**2.1**	93.7	**0.13**	**0.46**
		MCMC (Ours)	**25.4**	14.5	**21.0**	9.5	23.7
	# of Rejections	Exact (Rejection)	20880	79.2	3102	2.2	8.5
		MCMC (Ours)	7.8	3.5	6.0	0.8	6.8
$k = 50$	Runtime	Exact (Rejection)	$^{(*)}5.11\times10^{12}$	$^{(*)}9.55\times10^{5}$	$^{(*)}9.50\times10^{5}$	$^{(*)}1.45\times10^{12}$	$^{(*)}4.06\times10^{6}$
		MCMC (Ours)	**334**	**229**	**242**	**488**	**374**
	# of Rejections	Exact (Rejection)	$^{(*)}2.83\times10^{13}$	$^{(*)}4.94\times10^{6}$	$^{(*)}4.63\times10^{6}$	$^{(*)}4.66\times10^{12}$	$^{(*)}1.65\times10^{7}$
		MCMC (Ours)	3.8	1.3	1.6	5.4	3.2
Unconstrained	Runtime	Exact (Cholesky)	**5.6**	11.5	71.1	537	1540
		Exact (Rejection)	$^{(*)}1.34\times10^{8}$	**1.0**	1351.6	$^{(*)}1.89\times10^{10}$	1022
		MCMC (Ours)	75.3	11.8	**21**	**281**	**80**
	# of Rejections	Exact (Rejection)	$^{(*)}1.50\times10^{9}$	45.3	27941.7	$^{(*)}6.91\times10^{10}$	13924.5
		MCMC (Ours)	6.0	3.6	5.7	7.2	9.8
Preprocessing	Runtime	MCMC (Ours)	1.0	2.2	14.0	30.8	74.3

Figure 1. We observe that the TV distance of MCMC sampling decays as fast as that of the exact sampler for both k-NDPPs and NDPPs. This indicates that setting the number of MCMC iterations to k^2 is sufficient for convergence to the target distribution. Therefore, we use $t_{\text{iter}} = k^2$ for all of our experiments. In Appendix A.5, we additionally validate our choice for t_{iter} by evaluating the Potential Scale Reduction Factor (PSRF), commonly used to measure the convergence of the Markov chains (Gelman & Rubin, 1992).

6.2. Runtimes for Synthetic Datasets

Next, we report the runtimes of both the preprocessing and sampling steps of our proposed MCMC algorithm. We generate random NDPP kernels using the same approach described above, and measure the actual runtime in seconds. In Figure 2(a), we vary the size of ground set n from 10^2 to 10^6 while fixing $d = 100, k = 10$. In Figure 2(b), we vary k from 10 to 100 while $n = 10^6, d = 100$ are fixed. As discussed in Proposition 5, we verify that the preprocessing time increases linearly with respect to n, and that the sampling time tends to grow sublinearly in n. Interestingly, we notice that the sampling times for both $n = 10^2$ and 10^6 are almost identical, at about 10 seconds. This indicates that our algorithm scales well with respect to n, and is suitable for large-scale settings. We also see that our sampling algorithm scales superlinearly with k, because the number of MCMC iterations is set to $t_{\text{iter}} = k^2$.

6.3. Runtimes for Recommendation Datasets

To investigate the practical performance of our proposed sampling algorithms, we apply them to NDPP kernels learned from five real-world recommendation datasets, used in (Han et al., 2022). The ground set size n varies from 3,941 to 1 million, while the rank of the kernel is generally set to $d = 200$ for all datasets. More details on these datasets can be found in Appendices A.2 and A.3. We learn the low-rank components of the NDPP kernels, $\boldsymbol{V}, \boldsymbol{B}, \boldsymbol{D}$, using gradient-based maximum likelihood estimation, as described in (Gartrell et al., 2021).[3] We run our algorithms for k-NDPPs with sizes $k = 10$ and 50, and unconstrained-size NDPPs, and compare our MCMC algorithms to the exact rejection-based sampling algorithm (Han et al., 2022). We omit the naïve MCMC algorithm (Alimohammadi et al., 2021), which runs in quadratic time in n, from our experiments, because it is over 1,000 times slower than our sampling method for synthetic NDPP kernels with $n = 1{,}000$. For NDPP sampling, we also test the Cholesky-based sampling algorithm (Poulson, 2020), which has linear runtime in n. In Table 2, we report the runtimes of each sampling algorithm, as well as the number of rejections if the algorithm is based on rejection sampling.

We observe that for the 10-NDPP, the exact sampling algo-

[3]We use the code from https://github.com/insuhan/nonsymmetric-dpp-sampling for data preprocessing and NDPP kernel learning.

rithm often runs faster than our MCMC method, e.g., for the Recipe, Million Song, and Book datasets. However, for the 50-NDPP, the exact sampling algorithm results in a very large number of rejections on average, and thus is infeasible for all datasets. On the other hand, our MCMC sampler always terminates within a few minutes, running orders of magnitude faster than the exact sampling algorithm. For NDPP sampling, our algorithm is also orders of magnitude faster for the UK Retail and Million Song datasets. In Appendix A.4, we also apply those sampling algorithms to NDPP kernels learned with an orthogonality constraint (known as ONDPPs), which is tailored to ensure a small number of rejections for NDPP sampling (Han et al., 2022). These results show that reducing the runtime complexity from exponential to polynomial time can be very important in practice. Additionally, for NDPP sampling, we see up to a 13 times speedup for our method compared to the linear-time Cholesky-based sampling algorithm.

7. Conclusion

We have shown in this work how to accelerate MCMC sampling for k-NDPPs by leveraging a tree-based rejection sampling algorithm. Our proposed sampling algorithm achieves runtime that is sublinear in n, and polynomial in d and k. We have also extended our scalable k-NDPP MCMC sampling approach to NDPP sampling, while preserving the same efficient runtime. Compared to the fastest state-of-the-art exact sampling algorithms for k-NDPPs and NDPPs, which have runtime that is quadratic in n or exponential in d, respectively, our method makes sampling feasible for large-scale real-world settings by showing significantly faster and more scalable runtimes.

Acknowledgement

Insu Han was supported by TATA DATA Analysis (grant no. 105676). Amin Karbasi acknowledges funding in direct support of this work from NSF (IIS-1845032), ONR (N00014-19-1-2406), and the AI Institute for Learning-Enabled Optimization at Scale (TILOS).

References

Alimohammadi, Y., Anari, N., Shiragur, K., and Vuong, T.-D. Fractionally log-concave and sector-stable polynomials: counting planar matchings and more. In *Symposium on the Theory of Computing (STOC)*, 2021.

Anari, N. and Vuong, T.-D. From Sampling to Optimization on Discrete Domains with Applications to Determinant Maximization. *arXiv preprint arXiv:2102.05347*, 2021.

Anari, N., Gharan, S. O., and Rezaei, A. Monte Carlo Markov Chain Algorithms for Sampling Strongly Rayleigh Distributions and Determinantal Point Processes. In *Conference on Learning Theory (COLT)*, 2016.

Celis, L. E., Deshpande, A., Kathuria, T., Straszak, D., and Vishnoi, N. K. On the Complexity of Constrained Determinantal Point Processes. In *APPROX/RANDOM*, 2017.

Chen, D., Sain, S. L., and Guo, K. Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining. *Journal of Database Marketing & Customer Strategy Management*, 2012.

Derezinski, M. Fast determinantal point processes via distortion-free intermediate sampling. In *Conference on Learning Theory (COLT)*, 2019.

Dupuy, C. and Bach, F. Learning determinantal point processes in sublinear time. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

Gartrell, M., Brunel, V.-E., Dohmatob, E., and Krichene, S. Learning Nonsymmetric Determinantal Point Processes. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Gartrell, M., Han, I., Dohmatob, E., Gillenwater, J., and Brunel, V.-E. Scalable Learning and MAP Inference for Nonsymmetric Determinantal Point Processes. In *International Conference on Learning Representations (ICLR)*, 2021.

Gelman, A. and Rubin, D. B. Inference from iterative simulation using multiple sequences. *Statistical science*, 1992.

Gillenwater, J., Kulesza, A., Mariet, Z., and Vassilvtiskii, S. A Tree-Based Method for Fast Repeated Sampling of Determinantal Point Processes. In *International Conference on Machine Learning (ICML)*, 2019.

Han, I. and Gillenwater, J. MAP Inference for Customized Determinantal Point Processes via Maximum Inner Product Search. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

Han, I., Gartrell, M., Gillenwater, J., Dohmatob, E., and Karbasi, A. Scalable Sampling for Nonsymmetric Determinantal Point Processes. In *International Conference on Learning Representations (ICLR)*, 2022.

Instacart. The Instacart Online Grocery Shopping Dataset, 2017. URL `https://www.instacart.com/datasets/grocery-shopping-2017`. Accessed May 2020.

Kathuria, T., Deshpande, A., and Kohli, P. Batched gaussian process bandit optimization via determinantal point processes. *Neural Information Processing Systems (NIPS)*, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Kulesza, A. and Taskar, B. k-DPPs: Fixed-size Determinantal Point Processes. In *International Conference on Machine Learning (ICML)*, 2011.

Kulesza, A. and Taskar, B. Determinantal Point Processes for Machine Learning. *Foundations and Trends® in Machine Learning*, 2012.

Li, C., Jegelka, S., and Sra, S. Fast DPP Sampling for Nystrom with Application to Kernel Methods. In *International Conference on Machine Learning (ICML)*, 2016.

Majumder, B. P., Li, S., Ni, J., and McAuley, J. J. Generating Personalized Recipes from Historical User Preferences. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

McFee, B., Bertin-Mahieux, T., Ellis, D. P., and Lanckriet, G. R. The million song dataset challenge. In *International Conference on the World Wide Web (WWW)*, 2012.

Pang, T., Xu, K., Du, C., Chen, N., and Zhu, J. Improving adversarial robustness via promoting ensemble diversity. In *International Conference on Machine Learning (ICML)*. PMLR, 2019.

Poulson, J. High-performance sampling of generic Determinantal Point Processes. *Philosophical Transactions of the Royal Society A*, 2020.

Rezaei, A. and Gharan, S. O. A polynomial time MCMC method for sampling from continuous determinantal point processes. In *International Conference on Machine Learning (ICML)*, 2019.

Sharghi, A., Borji, A., Li, C., Yang, T., and Gong, B. Improving Sequential Determinantal Point Processes for Supervised Video Summarization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

Tremblay, N., Barthelmé, S., and Amblard, P.-O. Determinantal Point Processes for Coresets. *Journal of Machine Learning Research (JMLR)*, 2019.

Wan, M. and McAuley, J. Item recommendation on monotonic behavior chains. In *Conference on Recommender Systems (RecSys)*, 2018.

Yang, Y., Wen, Y., Wang, J., Chen, L., Shao, K., Mguni, D., and Zhang, W. Multi-agent determinantal q-learning. In *International Conference on Machine Learning (ICML)*, 2020.

Youla, D. A normal form for a matrix under the unitary congruence group. *Canadian Journal of Mathematics*, 1961.

Zhang, C., Kjellström, H., and Mandt, S. Determinantal Point Processes for Mini-Batch Diversification. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

A. Additional Details on Experimental Results

A.1. Efficient Tree Construction

Although our MCMC sampler can be very fast for large-scale settings, we do note that consideration of the preprocessing cost is important. Notably, preprocessing requires construction of a binary tree with $\mathcal{O}(nd^2)$ memory space, which can be problematic in practice. To alleviate this, we suggest a *fat-leaf tree structure*, where each leaf node contains $B > 1$ elements. This reduces the number of nodes in the tree to $\mathcal{O}(\frac{n}{B})$, and thus memory space can be reduced to $\mathcal{O}(d^2\frac{n}{B})$. However, since tree-based sampling returns a leaf node with some probability, according to Line 8 in Algorithm 3, an additional cost for computing the probabilities required for selecting a single item is required, with runtime $\mathcal{O}(d^2B)$. Therefore, with this change, the tree-based 2-DPP sampling runtime becomes $\mathcal{O}\left(d^2\left(\frac{\log n}{B}+B\right)+d^3\right)$. We set $B=8$ for datasets with $n \geq 10^5$ elements, and observe that the additional runtime overhead is very marginal, while memory consumption is reduced by a factor of 8.

A.2. Full Details on Datasets

We perform experiments on the following real-world public datasets:

- **UK Retail:** This dataset (Chen et al., 2012) contains baskets representing transactions from an online retail company that sells all-occasion gifts. We omit baskets with more than 100 items, leaving us with a dataset containing 19,762 baskets drawn from a catalog of $n = 3{,}941$ products. Baskets containing more than 100 items are in the long tail of the basket-size distribution.
- **Recipe:** This dataset (Majumder et al., 2019) contains recipes and food reviews from Food.com (formerly Genius Kitchen)[4]. Each recipe ("basket") is composed of a collection of ingredients, resulting in 178,265 recipes and a catalog of 7,993 ingredients.
- **Instacart:** This dataset (Instacart, 2017) contains baskets purchased by Instacart users[5]. We omit baskets with more than 100 items, resulting in 3.2 million baskets and a catalog of 49,677 products.
- **Million Song:** This dataset (McFee et al., 2012) contains playlists ("baskets") of songs from Echo Nest users[6]. We trim playlists with more than 100 items, leaving 968,674 playlists and a catalog of 371,410 songs.
- **Book:** This dataset (Wan & McAuley, 2018) contains reviews from the Goodreads book review website, including a variety of attributes describing the items[7]. For each user we build a subset ("basket") containing the books reviewed by that user. We trim subsets with more than 100 books, resulting in 430,563 subsets and a catalog of 1,059,437 books.

A.3. Full Details on Experimental Setup

NDPP kernel learning. We use the learning algorithm described in (Gartrell et al., 2021), where we learn the kernel components $\boldsymbol{V}, \boldsymbol{B} \in \mathbb{R}^{n\times d/2}, \boldsymbol{D} \in \mathbb{R}^{d/2\times d/2}$ by minimizing the regularized negative log-likelihood using training example subsets $\{Y_1, \ldots, Y_m\}$:

$$\begin{aligned}\min_{\boldsymbol{V},\boldsymbol{B},\boldsymbol{D}} \quad & -\frac{1}{m}\sum_{i=1}^{m}\log\det\left(\boldsymbol{V}_{Y_i}\boldsymbol{V}_{Y_i}^\top + \boldsymbol{B}_{Y_i}(\boldsymbol{D}-\boldsymbol{D}^\top)\boldsymbol{B}_{Y_i}^\top\right) \\ & + \log\det\left(\boldsymbol{V}\boldsymbol{V}^\top + \boldsymbol{B}(\boldsymbol{D}-\boldsymbol{D}^\top)\boldsymbol{B}^\top + \boldsymbol{I}\right) + \alpha\sum_{i=1}^{n}\frac{\|\boldsymbol{v}_i\|_2^2}{\mu_i} + \beta\sum_{i=1}^{n}\frac{\|\boldsymbol{b}_i\|_2^2}{\mu_i}, \end{aligned} \tag{16}$$

where $\boldsymbol{v}_i$ and $\boldsymbol{b}_i$ are the i-th row vectors of $\boldsymbol{V}$ and $\boldsymbol{B}$, respectively. We also use the training scheme from (Han et al., 2022), where 300 randomly-selected baskets are held-out as a validation set for tracking convergence during training,

[4]See https://www.kaggle.com/shuyangli94/food-com-recipes-and-user-interactions for the license for this public dataset.

[5]This public dataset is available for non-commercial use; see https://www.instacart.com/datasets/grocery-shopping-2017 for the license.

[6]See http://millionsongdataset.com/faq/ for the license for this public dataset.

[7]This public dataset is available for academic use only; see https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home for the license.

Table 3. Number of rejections and runtime (in seconds) for k-NDPP and unconstrained-size NDPP sampling algorithms, for ONDPP kernels learned with regularization on the number of NDPP sampling rejections. Bold values indicate the fastest runtimes, and $(*)$ indicates the expected results for those cases where the sampling algorithm does not terminate within a feasible timeframe.

Task	Metric	Algorithm	**UK Retail** n=3,941	**Recipe** n=7,993	**Instacart** n=49,677	**Million Song** n=371,410	**Book** n=1,059,437
$k = 10$	Runtime	Exact (Rejection)	**0.04**	**0.6**	**1.9**	**0.2**	**0.8**
		MCMC (Ours)	6.5	9.0	11.8	8.3	10.7
	# of Rejections	Exact (Rejection)	0	12.4	36.6	2.1	13.4
		MCMC (Ours)	0	0.9	1.3	0.3	0.9
$k = 50$	Runtime	Exact (Rejection)	**0.4**	$^{(*)}7.33\times10^{9}$	$^{(*)}1.92\times10^{8}$	**99.1**	$^{(*)}7.95\times10^{6}$
		MCMC (Ours)	140.7	**450.8**	**307.9**	182.9	**285.2**
	# of Rejections	Exact (Rejection)	0.1	$^{(*)}2.08\times10^{8}$	$^{(*)}4.96\times10^{8}$	239.7	$^{(*)}1.61\times10^{7}$
		MCMC (Ours)	0.1	6.2	2.6	0.5	2.1
Unconstrained	Runtime	Exact (Rejection)	**0.1**	**0.7**	**6.0**	**7.5**	**2.8**
		MCMC (Ours)	23.1	8.3	11.7	81.3	17.1
	# of Rejections	Exact (Rejection)	0.1	15.0	91.6	27.5	34.0
		MCMC (Ours)	0.0	1.1	1.3	0.4	0.9

another 2000 random subsets are used for testing, and the remaining baskets are used for training. Convergence is reached during training when the relative change in validation log-likelihood is below a predetermined threshold. We use the Adam optimizer (Kingma & Ba, 2015); we initialize $\boldsymbol{D}$ from $\mathcal{N}(0,1)$, and $\boldsymbol{V}$ and $\boldsymbol{B}$ are initialized from the $\mathcal{U}([0,1])$. We set $\alpha = \beta = 0.01$ for all datasets.

ONDPP kernel learning. Unlike the NDPP kernel, the orthogonal NDPP (ONDPP) kernel (Han et al., 2022) is parameterized as $\boldsymbol{L} = \boldsymbol{V}\boldsymbol{V}^\top + \boldsymbol{B}(\boldsymbol{D} - \boldsymbol{D}^\top)\boldsymbol{B}^\top$, where

$$\boldsymbol{D} = \mathrm{Diag}\left(\begin{bmatrix} 0 & \sigma_1 \\ 0 & 0 \end{bmatrix}, \ldots, \begin{bmatrix} 0 & \sigma_{d/2} \\ 0 & 0 \end{bmatrix}\right) \in \mathbb{R}^{d/2 \times d/2}$$

and $\sigma_j > 0$. The training objective is

$$\min_{\boldsymbol{V},\boldsymbol{B},\{\sigma_j\}_{j=1}^{d/2}} -\frac{1}{m}\sum_{i=1}^{m}\log\left(\frac{\det(\boldsymbol{L}_{Y_i})}{\det(\boldsymbol{L}+\boldsymbol{I})}\right) + \alpha\sum_{i=1}^{n}\frac{\|\boldsymbol{v}_i\|_2^2}{\mu_i} + \beta\sum_{i=1}^{n}\frac{\|\boldsymbol{b}_i\|_2^2}{\mu_i} + \gamma\sum_{j=1}^{d/2}\log\left(1+\frac{2\sigma_j}{\sigma_j^2+1}\right), \tag{17}$$

with constraints $\boldsymbol{B}^\top\boldsymbol{B} = \boldsymbol{I}$ and $\boldsymbol{V}^\top\boldsymbol{B} = 0$. To satisfy the first constraint, Han et al. (2022) applies QR decomposition on $\boldsymbol{B}$; for the second constraint, we project $\boldsymbol{V}$ to the column space of $\boldsymbol{B}$ by updating $\boldsymbol{V} \leftarrow \boldsymbol{V} - \boldsymbol{B}(\boldsymbol{B}^\top\boldsymbol{B})^{-1}(\boldsymbol{B}^\top\boldsymbol{V})$. We use the regularizer settings from Han et al. (2022): $\alpha = \beta = 0.01, \gamma = 0.5$ for the UK Retail dataset, $\alpha = \beta = 0.01, \gamma = 0.1$ for Recipe, $\alpha = \beta = 0.001, \gamma = 0.001$ for Instacart, $\alpha = \beta = 0.01, \gamma = 0.2$ for Million Song, and $\alpha = \beta = 0.01, \gamma = 0.1$ for Book.

A.4. Additional Experiments with ONDPPs

We apply NDPP sampling algorithms in Section 6.3 to NDPP kernels learned with an orthogonality constraint (known as ONDPPs), studied in (Han et al., 2022). In particular, these kernels are learned using a regularization mechanism that guarantees a small number of NDPP sampling rejections. Therefore, we expect exact sampling with ONDPP kernels to run very quickly. Table 3 shows the results with real-world datasets and ONDPP kernels learned on these datasets. As expected, exact ONDPP sampling runs faster than our MCMC approach for unconstrained-size NDPPs. It also runs faster for 10-NDPP sampling. However, we see that for 50-NDPP the expected exact sampling runtimes are over 92 days for three datasets, while our MCMC approach always terminates within a few minutes. This substantial slowdown for 50-NDPP sampling results from the runtime being exponential in k for exact sampling, which we are unable to mitigate using regularization during training. This suggests that for 50-NDPP sampling, our scalable MCMC algorithm is the best and only viable choice in practice.

A.5. Empirical Mixing Time with Potential Scale Reduction Factor (PSRF)

We additionally validate the mixing times of our MCMC sampling algorithm (Algorithm 1) using the Potential Scale Reduction Factor (PSRF). PSRF computes the ratio of within-chain and between-chain variances and is frequently used for measuring the empirical mixing times of MCMC samplers. We used the synthetic dataset described in Section 6.1, and the PSRF implementation in `tensorflow.probability.mcmc`, with 100 independent chains for $n = \{100, 200, \ldots, 3200\}, k = \{2, 3, \ldots, 30\}$, and a fixed $d = 20$. Interestingly, as shown in Figure 3, we observe that empirical mixing times increase linearly in k for all choices of n. We leave the problem of further improving the mixing time of our NDPP MCMC sampling algorithm for future work.

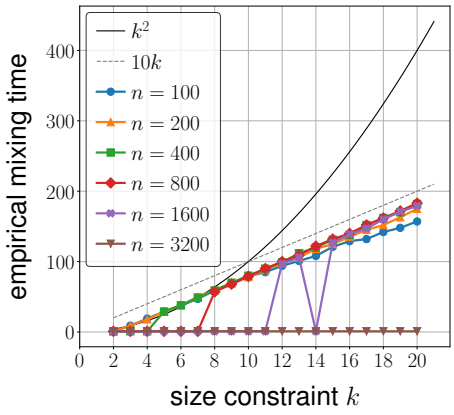


Figure 3. Empirical mixing time, computed using Potential Scale Reduction Factor (PSRF), for our proposed NDPP MCMC sampling algorithm.

B. MAP Inference for Initialization

We observe that the mixing time in Equation (5) also depends on the initial subset S_0. It is desirable to find a size-k subset S_0 where $\det(\boldsymbol{L}_{S_0})$ is as large as possible, and then use S_0 as the initial subset in Algorithm 1. This is known as the MAP inference problem for a DPP; that is,

$$\operatorname*{argmax}_{S \in \binom{[n]}{k}} \det(\boldsymbol{L}_S).$$

MAP inference for a NDPP is generally known to be NP-hard, and a greedy algorithm is typically used as a heuristic (Gartrell et al., 2021). In particular, Gartrell et al. (2021) showed that with a rank-d NDPP kernel, greedy MAP inference runs in $\mathcal{O}(nd^2)$ time. Once we find a proper size-k subset S_0, we can re-use S_0 for drawing subsequent k-NDPP samples. Therefore, for a faster mixing time, we utilize greedy MAP inference as a preprocessing step, while preserving the total preprocessing runtime described previously.

Furthermore, this MAP-based initialization approach can also be used for NDPP sampling without size constraints. We note that the greedy algorithm finds elements in the output subset in a sequential way. In other words, if $\{s_1, \ldots, s_d\}$ is the output of the greedy algorithm with size constraint d, then the algorithm with size constraint $k \leq d$ returns $\{s_1, \ldots, s_k\}$. Therefore, for NDPP sampling, we run the greedy algorithm to find a sequence of d items that maximize the determinant of each principal submatrix of size d. While running our MCMC NDPP sampler (Algorithm 4), if the size random variable k is selected, then we run the MCMC k-NDPP sampling (Algorithm 1) with the chosen k and a subset containing the first $k \leq d$ elements in the sequence obtained from the greedy algorithm. In practice, for our experiments in Section 6.1 we observe that our MCMC sampler without greedy initialization shows promising convergence, and thus we omit this procedure in our experiments.

C. Proofs

C.1. Proof of Theorem 1

Theorem 1. *Given $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ and $\boldsymbol{W}^A \in \mathbb{R}^{d\times d}$, suppose $\widehat{\boldsymbol{W}}^A$ is obtained from Equation (9) with $\boldsymbol{W}^A$. Then,*

$$\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_S) \leq \det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_S) \tag{10}$$

for every $S \subseteq [n]$. In addition, equality holds when $|S| \geq d$.

Proof. For simplicity, we write that $\boldsymbol{G} := \boldsymbol{X}(\frac{\boldsymbol{W}^A+\boldsymbol{W}^{A\top}}{2})\boldsymbol{X}^\top$ and $\boldsymbol{A} := \boldsymbol{X}(\frac{\boldsymbol{W}^A-\boldsymbol{W}^{A\top}}{2})\boldsymbol{X}^\top$, so that $\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top = \boldsymbol{G}+\boldsymbol{A}$. Also, denote $\boldsymbol{B} := \boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top - \boldsymbol{G}$. Since $\boldsymbol{G}$ is positive semi-definite, for any $S \subseteq [n]$ such that $|S| \leq d$, we have

$$\begin{aligned}
\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_S) &= \det\left(\boldsymbol{G}_S + \boldsymbol{A}_S\right) \\
&= \det(\boldsymbol{G}_S^{1/2}(\boldsymbol{I} + \boldsymbol{G}_S^{-1/2}\boldsymbol{A}_S\boldsymbol{G}_S^{-1/2})\boldsymbol{G}_S^{1/2}) \\
&= \det(\boldsymbol{G}_S^{1/2}) \cdot \det(\boldsymbol{I} + \boldsymbol{G}_S^{-1/2}\boldsymbol{A}_S\boldsymbol{G}_S^{-1/2}) \cdot \det(\boldsymbol{G}_S^{1/2})
\end{aligned} \tag{18}$$

where $\boldsymbol{I}$ is the $|S|$-by-$|S|$ identity matrix. Similarly,

$$\det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_S) = \det(\boldsymbol{G}_S^{1/2}) \cdot \det(\boldsymbol{I} + \boldsymbol{G}_S^{-1/2}\boldsymbol{B}_S\boldsymbol{G}_S^{-1/2}) \cdot \det(\boldsymbol{G}_S^{1/2}). \tag{19}$$

From Theorem 2.1 in (Kulesza & Taskar, 2012), we have

$$\det(\boldsymbol{I} + \boldsymbol{G}_S^{-1/2}\boldsymbol{A}_S\boldsymbol{G}_S^{-1/2}) = \sum_{T\subseteq[|S|]} \det([\boldsymbol{G}_S^{-1/2}\boldsymbol{A}_S\boldsymbol{G}_S^{-1/2}]_T), \tag{20}$$

$$\det(\boldsymbol{I} + \boldsymbol{G}_S^{-1/2}\boldsymbol{B}_S\boldsymbol{G}_S^{-1/2}) = \sum_{T\subseteq[|S|]} \det([\boldsymbol{G}_S^{-1/2}\boldsymbol{B}_S\boldsymbol{G}_S^{-1/2}]_T). \tag{21}$$

Therefore, it is enough to prove that for every $T \subseteq [|S|]$

$$\det([\boldsymbol{G}_S^{-1/2}\boldsymbol{A}_S\boldsymbol{G}_S^{-1/2}]_T) \leq \det([\boldsymbol{G}_S^{-1/2}\boldsymbol{B}_S\boldsymbol{G}_S^{-1/2}]_T). \tag{22}$$

Now consider the Youla decomposition on $\frac{\boldsymbol{W}^A-\boldsymbol{W}^{A\top}}{2}$ as in Equation (8), i.e.,

$$\frac{\boldsymbol{W}^A - \boldsymbol{W}^{A\top}}{2} = \sum_{i=1}^{d/2} \sigma_i \left(\boldsymbol{y}_i\boldsymbol{z}_i^\top - \boldsymbol{z}_i\boldsymbol{y}_i^\top\right) = \boldsymbol{V}\ \mathrm{Diag}\left(\begin{bmatrix} 0 & \sigma_1 \\ -\sigma_1 & 0 \end{bmatrix}, \cdots, \begin{bmatrix} 0 & \sigma_{\frac{d}{2}} \\ -\sigma_{\frac{d}{2}} & 0 \end{bmatrix}\right) \boldsymbol{V}^\top, \tag{23}$$

where $\boldsymbol{V} := [\boldsymbol{y}_1, \boldsymbol{z}_1 \ldots, \boldsymbol{y}_{\frac{d}{2}}, \boldsymbol{z}_{\frac{d}{2}}]$. Then, it can be written

$$\boldsymbol{G}_S^{-1/2}\boldsymbol{A}_S\boldsymbol{G}_S^{-1/2} = \boldsymbol{G}_S^{-1/2}\boldsymbol{X}_{S,:}\boldsymbol{V} \cdot \mathrm{Diag}\left(\begin{bmatrix} 0 & \sigma_1 \\ -\sigma_1 & 0 \end{bmatrix}, \cdots, \begin{bmatrix} 0 & \sigma_{\frac{d}{2}} \\ -\sigma_{\frac{d}{2}} & 0 \end{bmatrix}\right) \cdot \boldsymbol{V}^\top\boldsymbol{X}_{S,:}^\top\boldsymbol{G}_S^{-1/2} := \boldsymbol{R}, \tag{24}$$

$$\boldsymbol{G}_S^{-1/2}\boldsymbol{B}_S\boldsymbol{G}_S^{-1/2} = \boldsymbol{G}_S^{-1/2}\boldsymbol{X}_{S,:}\boldsymbol{V} \cdot \mathrm{Diag}\left(\sigma_1, \sigma_1, \ldots, \sigma_{\frac{d}{2}}, \sigma_{\frac{d}{2}}\right) \cdot \boldsymbol{V}^\top\boldsymbol{X}_{S,:}^\top\boldsymbol{G}_S^{-1/2} := \widehat{\boldsymbol{R}}. \tag{25}$$

From Theorem 1 in (Han et al., 2022), it holds that $\det(\boldsymbol{R}_T) \leq \det(\widehat{\boldsymbol{R}}_T)$ for all $T \subseteq [|S|]$. This completes the proof of Theorem 1. □

C.2. Proof of Theorem 2

Theorem 2. *Given $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, and symmetric and PSD $\widehat{\boldsymbol{W}}^A \in \mathbb{R}^{d\times d}$, Algorithm 3 samples a subset from the k-DPP with kernel $\boldsymbol{L} = \boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top$, and runs in time $\mathcal{O}(kd^2\log n + k^2d^2 + d^3)$, after a one-time preprocessing step that runs in time $\mathcal{O}(nd^2)$.*

Proof. The preprocessing for the k-DPP sampler includes (1) a binary tree construction based on $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and (2) computing $\boldsymbol{C} = \boldsymbol{X}^\top \boldsymbol{X} \in \mathbb{R}^{d \times d}$. Both can be done in $\mathcal{O}(nd^2)$ time. Given this preprocessing, Algorithm 3 first performs the eigendecomposition of $\boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^\top$, which requires $\mathcal{O}(d^3)$ time. Then, a subset $E \subseteq [d]$ is sampled with probability proportional to $\prod_{i \in E} \lambda_i$ where the λ_i's are the eigenvalues of $\boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^\top$. With (Kulesza & Taskar, 2012, Algorithm 8), this can be done in $\mathcal{O}(dk)$ time. Next, we need to perform tree-based sampling and query matrix updates for k iterations. Since the tree has depth $\mathcal{O}(\log n)$, and computing the required probability for moving down the tree takes $\mathcal{O}(d^2)$ time, the tree-based sampler requires $\mathcal{O}(d^2 \log n)$ time. In addition, computation of the query matrix runs in $\mathcal{O}(d^2 k)$. Therefore, the overall runtime of Algorithm 3 (after preprocessing) is $\mathcal{O}(d^3 + kd^2 \log n + k^2 d^2)$. This improves the runtime of $\mathcal{O}(d^3 + k^2 d^2 \log n)$ from previous work (Gillenwater et al., 2019), which uses an alternative probability formulation for the tree traversal in Equation (14) that needs several matrix multiplications in every tree node, resulting in $\mathcal{O}(k^2 d^2 \log n)$ runtime. In our algorithm, these matrix multiplications are computed with a query matrix, outside of the tree traversal. □

C.3. Proof of Theorem 4

Theorem 4. *Given $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{W} \in \mathbb{R}^{d \times d}$, such that $\boldsymbol{W} + \boldsymbol{W}^\top \succeq 0$ and $A \in \binom{[n]}{k-2}$ for $k \geq 2$, consider κ_A as in Definition 3. Then, the average number of rejections of the rejecion-based up operator (Algorithm 2) is no greater than $(1 + \sigma_{\max}(\boldsymbol{X})^2 \, \kappa_A)^2$.*

Proof. Let p be the probability distribution of the target 2-NDPP with kernel $\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top$, and q be that of the proposal 2-DPP with kernel $\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top$. For every $S \in \binom{[n] \setminus A}{2}$, it holds that

$$
\begin{aligned}
p(S) &= \frac{\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_S)}{\sum_{\{a,b\} \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_{\{a,b\}})} \\
&\leq \frac{\det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_S)}{\sum_{\{a,b\} \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_{\{a,b\}})} \\
&= \frac{\sum_{\{a,b\} \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_{\{a,b\}})}{\sum_{\{a,b\} \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_{\{a,b\}})} \cdot \frac{\det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_S)}{\sum_{\{a,b\} \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_{\{a,b\}})} \\
&= \frac{\sum_{\{a,b\} \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_{\{a,b\}})}{\sum_{\{a,b\} \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_{\{a,b\}})} \cdot q(S),
\end{aligned}
$$

where the inequality comes from Theorem 1. This tells us that the average number of rejections is equal to

$$
\frac{\sum_{\{a,b\} \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_{\{a,b\}})}{\sum_{\{a,b\} \in \binom{[n] \setminus A}{2}} \det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_{\{a,b\}})}. \tag{26}
$$

Instead of finding an upper bound on the above directly, we consider the following

$$
\max_{\{a,b\} \in \binom{[n] \setminus A}{2}} \frac{\det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_{\{a,b\}})}{\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_{\{a,b\}})},
$$

which is greater than or equal to expression (26).

Now, for any symmetric and positive semidefinite (SPSD) matrix $\boldsymbol{M}$, we denote by $\lambda_{\max}(\boldsymbol{M})$ and $\lambda_{\min}(\boldsymbol{M})$ the largest and smallest nonzero eigenvalues of $\boldsymbol{M}$, respectively. Let $\boldsymbol{S} \coloneqq \frac{\boldsymbol{W}^A + \boldsymbol{W}^{A\top}}{2}$ and $\boldsymbol{R} \coloneqq \widehat{\boldsymbol{W}}^A - \boldsymbol{S}$. From the construction of $\widehat{\boldsymbol{W}}^A$ in Equation (9), it is easy to check that both $\boldsymbol{S}$ and $\boldsymbol{R}$ are SPSD. First we claim that for any $Y \subseteq [n] \setminus A$, it holds that

$$
\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_Y) \geq \det([\boldsymbol{X}\boldsymbol{S}\boldsymbol{X}^\top]_Y). \tag{27}
$$

This comes from the following. If $\det([\boldsymbol{X}\boldsymbol{S}\boldsymbol{X}^\top]_Y) = 0$, the result is trivial due to $\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_Y) \geq 0$ for all Y.

Assume that $\det([\boldsymbol{X}\boldsymbol{S}\boldsymbol{X}^\top]_Y) \neq 0$, then

$$\frac{\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_Y)}{\det([\boldsymbol{X}\boldsymbol{S}\boldsymbol{X}^\top]_Y)} = \frac{\det([\boldsymbol{X}\boldsymbol{S}\boldsymbol{X}^\top]_Y + [\boldsymbol{X}(\boldsymbol{W}^A - \boldsymbol{S})\boldsymbol{X}^\top]_Y)}{\det([\boldsymbol{X}\boldsymbol{S}\boldsymbol{X}^\top]_Y)} \tag{28}$$

$$= \det\left(\boldsymbol{I}_{|Y|} + \underbrace{[\boldsymbol{X}\boldsymbol{S}\boldsymbol{X}^\top]_Y^{-\frac{1}{2}}\,\boldsymbol{X}_{Y,:}}_{:=\boldsymbol{X}'}\,(\boldsymbol{W}^A - \boldsymbol{S})\,\boldsymbol{X}_{Y,:}^\top\,[\boldsymbol{X}\boldsymbol{S}\boldsymbol{X}^\top]_Y^{-\frac{1}{2}}\right) \tag{29}$$

$$= \det\left(\boldsymbol{I}_{|Y|} + \boldsymbol{X}'\left(\boldsymbol{W}^A - \boldsymbol{S}\right)\boldsymbol{X}'^\top\right) \tag{30}$$

$$= \sum_{T\subseteq[|Y|]} \det\left(\left[\boldsymbol{X}'\left(\boldsymbol{W}^A - \boldsymbol{S}\right)\boldsymbol{X}'^\top\right]_T\right) \tag{31}$$

$$\geq \det(\left[\boldsymbol{X}'\left(\boldsymbol{W}^A - \boldsymbol{S}\right)\boldsymbol{X}'^\top\right]_\emptyset) = 1, \tag{32}$$

where the fourth line comes from (Kulesza & Taskar, 2012, Theorem 2.1), and the last line follows from the observation that $\boldsymbol{W}^A - \boldsymbol{S} = \frac{\boldsymbol{W}^A - \boldsymbol{W}^{A\top}}{2}$ is a skew-symmetric matrix, so that every principal submatrix has a nonnegative determinant. Now we fix some $\{a, b\} \in \binom{[n]\setminus A}{2}$ and denote $\boldsymbol{Q} := \boldsymbol{X}_{\{a,b\},:} \in \mathbb{R}^{2\times d}$. Then we have

$$\frac{\det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_{\{a,b\}})}{\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_{\{a,b\}})} \leq \frac{\det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_{\{a,b\}}}{\det([\boldsymbol{X}\boldsymbol{S}\boldsymbol{X}^\top]_{\{a,b\}})} \tag{33}$$

$$= \frac{\det(\boldsymbol{Q}(\boldsymbol{S} + \boldsymbol{R})\boldsymbol{Q}^\top)}{\det(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^\top)} \tag{34}$$

$$= \det\left(\boldsymbol{I}_2 + \left(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^\top\right)^{-1/2}\boldsymbol{Q}\boldsymbol{R}\boldsymbol{Q}^\top\left(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^\top\right)^{-1/2}\right) \tag{35}$$

$$\leq \left(\frac{1}{2}\cdot\operatorname{tr}\left(\boldsymbol{I}_2 + \left(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^\top\right)^{-1/2}\boldsymbol{Q}\boldsymbol{R}\boldsymbol{Q}^\top\left(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^\top\right)^{-1/2}\right)\right)^2 \tag{36}$$

$$= \left(1 + \frac{1}{2}\cdot\operatorname{tr}\left(\boldsymbol{Q}\boldsymbol{R}\boldsymbol{Q}^\top\left(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^\top\right)^{-1}\right)\right)^2 \tag{37}$$

$$\leq \left(1 + \frac{1}{2}\cdot\operatorname{tr}\left(\boldsymbol{Q}\boldsymbol{R}\boldsymbol{Q}^\top\right)\cdot\lambda_{\max}\left(\left(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^\top\right)^{-1}\right)\right)^2 \tag{38}$$

$$\leq \left(1 + \frac{\lambda_{\max}\left(\boldsymbol{Q}\boldsymbol{R}\boldsymbol{Q}^\top\right)}{\lambda_{\min}\left(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^\top\right)}\right)^2, \tag{39}$$

where the first line follows from Equation (27), the fourth line is due to the fact that $\det(\boldsymbol{M}) \leq (\operatorname{tr}(\boldsymbol{M})/d)^d$ for a SPSD matrix $\boldsymbol{M} \in \mathbb{R}^{d\times d}$ (thanks to the *AM-GM inequality*), the fifth line comes from the cyclic property of a trace, and the sixth line is from the fact that $\operatorname{tr}(\boldsymbol{M}\boldsymbol{N}) \leq \operatorname{tr}(\boldsymbol{M})\cdot\lambda_{\max}(\boldsymbol{N})$ for SPSD matrices $\boldsymbol{M}, \boldsymbol{N}$. For an arbitrary vector $\boldsymbol{v} \in \mathbb{R}^2$, we observe that

$$\boldsymbol{v}^\top\left(\boldsymbol{Q}\boldsymbol{R}\boldsymbol{Q}^\top\right)\boldsymbol{v} \leq \lambda_{\max}(\boldsymbol{R})\cdot\boldsymbol{v}^\top\boldsymbol{Q}\boldsymbol{Q}^\top\boldsymbol{v} \leq \lambda_{\max}(\boldsymbol{R})\cdot\lambda_{\max}(\boldsymbol{Q}\boldsymbol{Q}^\top)\cdot\|\boldsymbol{v}\|_2^2.$$

Since $\boldsymbol{Q}\boldsymbol{Q}^\top = \boldsymbol{X}_{\{a,b\},:}\boldsymbol{X}_{\{a,b\},:}^\top = [\boldsymbol{X}\boldsymbol{X}^\top]_{\{a,b\}} \in \mathbb{R}^{2\times 2}$ is a principal submatrix of $\boldsymbol{X}\boldsymbol{X}^\top$, by Cauchy's interlace theorem, all eigenvalues of $\boldsymbol{Q}\boldsymbol{Q}^\top$ interlace those of $\boldsymbol{X}\boldsymbol{X}^\top$, and thus $\lambda_{\max}(\boldsymbol{Q}\boldsymbol{Q}^\top) \leq \lambda_{\max}(\boldsymbol{X}\boldsymbol{X}^\top) = \sigma_{\max}(\boldsymbol{X})^2$. Furthermore, since the matrix $\boldsymbol{R}$ is obtained from the spectral symmetrization of $\frac{\boldsymbol{W}^A - \boldsymbol{W}^{A\top}}{2}$, their spectra are identical, i.e., $\lambda_{\max}(\boldsymbol{R}) = \frac{\sigma_{\max}(\boldsymbol{W}^A - \boldsymbol{W}^{A\top})}{2}$. Therefore,

$$\lambda_{\max}\left(\boldsymbol{Q}\boldsymbol{R}\boldsymbol{Q}^\top\right) \leq \frac{\sigma_{\max}(\boldsymbol{W}^A - \boldsymbol{W}^{A\top})}{2}\cdot\sigma_{\max}(\boldsymbol{X})^2. \tag{40}$$

In addition, we have[8]

$$\lambda_{\min}\left(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^\top\right) = \frac{\sigma_{\min}\left(\left[\boldsymbol{X}\left(\boldsymbol{W}^A + \boldsymbol{W}^{A\top}\right)\boldsymbol{X}^\top\right]_{\{a,b\}}\right)}{2} \geq \frac{\min_{Y\in\binom{[n]\setminus A}{2}}\sigma_{\min}([\boldsymbol{X}(\boldsymbol{W}^A + \boldsymbol{W}^{A\top})\boldsymbol{X}^\top]_Y)}{2}. \tag{41}$$

Putting Equations (40) and (41) into Equation (39) gives

$$\frac{\det([\boldsymbol{X}\widehat{\boldsymbol{W}}^A\boldsymbol{X}^\top]_{\{a,b\}})}{\det([\boldsymbol{X}\boldsymbol{W}^A\boldsymbol{X}^\top]_{\{a,b\}})} \leq \left(1 + \sigma_{\max}(\boldsymbol{X})^2 \cdot \kappa_A\right)^2, \tag{42}$$

where, in Definition 3, κ_A is defined as

$$\kappa_A := \frac{\sigma_{\max}(\boldsymbol{W}^A - \boldsymbol{W}^{A^\top})}{\min_{Y\in\binom{[n]\setminus A}{2}}\sigma_{\min}([\boldsymbol{X}(\boldsymbol{W}^A + \boldsymbol{W}^{A\top})\boldsymbol{X}^\top]_Y)}.$$

This completes the proof of Theorem 4. □

C.4. Proof of Proposition 5

Proposition 5. *Given $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ and $\boldsymbol{W} \in \mathbb{R}^{d\times d}$, such that $\boldsymbol{W} + \boldsymbol{W}^\top \succeq 0$ and $k \geq 2$, consider κ as defined in Definition 3. With a preprocessing step that runs in time $\mathcal{O}(nd^2)$, Algorithm 1 runs in time $\mathcal{O}(t_{\text{iter}}\,(1 + \sigma_{\max}(\boldsymbol{X})^2\,\kappa)^2\,(d^2\log n + d^3))$ in expectation.*

Proof. We remind the reader that our MCMC sampler (Algorithm 1) repeatedly runs tree-based rejection sampling for t_{iter} iterations. From Theorem 4, each iteration requires 2-DPP sampling for at most $(1 + \sigma_{\max}(\boldsymbol{X})^2\kappa)^2$ times on average. From Theorem 2, sampling from the 2-DPP can be done in time $\mathcal{O}(d^2\log n + d^3)$. Combining all of these runtimes gives the result. □

[8] One can similarly show that $\lambda_{\min}\left(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^\top\right) \geq \lambda_{\min}\left(\boldsymbol{S}\right) \cdot \lambda_{\min}\left(\boldsymbol{Q}\boldsymbol{Q}^\top\right)$. However, the matrix $\boldsymbol{S}$ can be rank-deficient, because $\boldsymbol{W}^A$ is computed by projecting $\boldsymbol{W}$ onto some subspace with dimension $d - |A|$. Thus, this approach gives us a trivial lower bound of zero.