Exact sampling of determinantal point processes with sublinear time preprocessing

Michał Dereziński*  
Department of Statistics  
University of California, Berkeley  
mderezin@berkeley.edu

Daniele Calandriello*  
LCSL  
Istituto Italiano di Tecnologia, Italy  
daniele.calandriello@iit.it

Michał Valko  
DeepMind Paris  
valkom@deepmind.com

Abstract

We study the complexity of sampling from a distribution over all index subsets of the set \{1, ..., n\} with the probability of a subset \(S\) proportional to the determinant of the submatrix \(L_S\) of some \(n \times n\) positive semidefinite matrix \(L\), where \(L_S\) corresponds to the entries of \(L\) indexed by \(S\). Known as a determinantal point process (DPP), this distribution is used in machine learning to induce diversity in subset selection. When sampling from DDPs, we often wish to sample multiple subsets \(S\) with small expected size \(k \triangleq \mathbb{E}[|S|] \ll n\) from a very large matrix \(L\), so it is important to minimize the preprocessing cost of the procedure (performed once) as well as the sampling cost (performed repeatedly). For this purpose we provide DPP-VFX, a new algorithm which, given access only to \(L\), samples exactly from a determinantal point process while satisfying the following two properties: (1) its preprocessing cost is \(n \cdot \text{poly}(k)\), i.e., sublinear in the size of \(L\), and (2) its sampling cost is \(\text{poly}(k)\), i.e., independent of the size of \(L\). Prior to our results, state-of-the-art exact samplers required \(O(n^3)\) preprocessing time and sampling time linear in \(n\) or dependent on the spectral properties of \(L\). We furthermore give a reduction which allows using our algorithm for exact sampling from cardinality constrained determinantal point processes with \(n \cdot \text{poly}(k)\) time preprocessing. Our implementation of DPP-VFX is provided at https://github.com/guilgautier/DPPy/.

1 Introduction

Given a positive semi-definite (psd) \(n \times n\) matrix \(L\), a determinantal point process DPP(\(L\)), also known as an \(L\)-ensemble, is a distribution over all \(2^n\) index subsets \(S \subseteq \{1, \ldots, n\}\) such that

\[
\Pr(S) \triangleq \frac{\det(L_S)}{\det(I + L)},
\]

where \(L_S\) denotes the \(|S| \times |S|\) submatrix of \(L\) with rows and columns indexed by \(S\). Determinantal point processes naturally appear across many scientific domains [Mac75, BLMV17, Gue83], and they have emerged as an important tool in machine learning [KT12] for inducing diversity in subset selection and as a variance reduction approach. DPP sampling has been successfully applied in core ML problems such as recommender systems [GKVM18, CZZ18, Bru18], stochastic optimization [ZKM17, ZÖMS19], data summarization [CKS+18], Gaussian processes [MS16, BRVDW19],

*Equal contribution.
We refer to this algorithm as the Very Fast and eXact DPP sampler, or VFX-DPP sampler. It achieves an overall runtime of \(O(nk^2)\) where \(k \leq n\) and \(nk^2\) is the preprocessing time. Our method is based on a technique developed recently by [Le19] which (informally) samples \(n\cdot\text{poly}(k)\) samples \(\text{poly}(k)\) time.

We refer to this algorithm as the Very Fast and eXact DPP sampler, or VFX-DPP sampler. It achieves an overall runtime of \(O(nk^2)\) where \(k \leq n\) and \(nk^2\) is the preprocessing time. Our method is based on a technique developed recently by [Le19] which (informally) samples \(n\cdot\text{poly}(k)\) samples \(\text{poly}(k)\) time.

We refer to this algorithm as the Very Fast and eXact DPP sampler, or VFX-DPP sampler. It achieves an overall runtime of \(O(nk^2)\) where \(k \leq n\) and \(nk^2\) is the preprocessing time. Our method is based on a technique developed recently by [Le19] which (informally) samples \(n\cdot\text{poly}(k)\) samples \(\text{poly}(k)\) time.

We refer to this algorithm as the Very Fast and eXact DPP sampler, or VFX-DPP sampler. It achieves an overall runtime of \(O(nk^2)\) where \(k \leq n\) and \(nk^2\) is the preprocessing time. Our method is based on a technique developed recently by [Le19] which (informally) samples \(n\cdot\text{poly}(k)\) samples \(\text{poly}(k)\) time.

We refer to this algorithm as the Very Fast and eXact DPP sampler, or VFX-DPP sampler. It achieves an overall runtime of \(O(nk^2)\) where \(k \leq n\) and \(nk^2\) is the preprocessing time. Our method is based on a technique developed recently by [Le19] which (informally) samples \(n\cdot\text{poly}(k)\) samples \(\text{poly}(k)\) time.

We refer to this algorithm as the Very Fast and eXact DPP sampler, or VFX-DPP sampler. It achieves an overall runtime of \(O(nk^2)\) where \(k \leq n\) and \(nk^2\) is the preprocessing time. Our method is based on a technique developed recently by [Le19] which (informally) samples \(n\cdot\text{poly}(k)\) samples \(\text{poly}(k)\) time.
where the expected size of $\sigma$ is $O(k^2)$. To make the algorithm efficient, we use new connections between determinantal point processes, ridge leverage scores, and Nyström approximation.

**Definition 1** Given a psd matrix $L$, its $i$th $\lambda$-ridge leverage score (RLS) $\tau_i(\lambda)$ is the $i$th diagonal entry of $L(I + L)^{-1}$. The $\lambda$-effective dimension $d_{\text{eff}}(\lambda)$ is the sum of the leverage scores, $\sum_i \tau_i(\lambda)$.

An important connection between RLSs and DPPs is that when $S \sim \text{DPP}(L)$, the marginal probability of index $i$ being sampled into $S$ is equal to the $i$th 1-ridge leverage score of $L$, and the expected size $k$ of $S$ is equal to the 1-effective dimension,

$$\Pr(i \in S) = [L(I + L)^{-1}]_{ii} = \tau_i(1), \quad k \triangleq \mathbb{E}(|S|) = \text{tr}(L(I + L)^{-1}) = d_{\text{eff}}(1).$$

Intuitively, if the marginal probability of $i$ is high, then this index should likely make it into the intermediate sample $\sigma$. This suggests that i.i.d. sampling of the indices $\sigma_1, ..., \sigma_t$ proportionally to 1-ridge leverage scores, i.e., $\Pr(\sigma_1 = i) \propto \tau_i(1)$, should serve as a reasonable and cheap heuristic for constructing $\sigma$. In fact, we can show that this distribution can be easily corrected by rejection sampling to become the R-DPP that we need. Computing ridge leverage scores exactly costs $O(n^3)$, so instead we compute them approximately by first constructing a Nyström approximation of $L$.

**Definition 2** Let $L$ be a psd matrix and $C$ a subset of its row/column indices with size $m \triangleq |C|$. Then we define the Nyström approximation of $L$ based on $C$ as the $n \times n$ matrix $\tilde{L} \triangleq (L_{C;C})^+ L_{C;C}^+ L_{C;C}$.

Here, $L_{C;C}$ denotes an $m \times n$ matrix consisting of (entire) rows of $L$ indexed by $C$ and $(\cdot)^+$ denotes the Moore-Penrose pseudoinverse. Since we use rejection sampling to achieve the right intermediate distribution, the correctness of our algorithm does not depend on which Nyström approximation is chosen. However, the subset $C$ greatly influences the computational cost of the sampling through the rank of $\tilde{L}$ and the probability of rejecting a sample. Since $\text{rank}(\tilde{L}) = |C|$, operations such as multiplication and inversion involving the Nyström approximation will scale with $m$, and therefore a small subset increases efficiency. However, if $\tilde{L}$ is too different from $L$, the probability of rejecting the sample will be very high and the algorithm inefficient. In this case, a slightly larger subset could improve accuracy and acceptance rate without increasing too much the cost of handling $L$. Therefore, subset $C$ has to be selected so that it is both small and accurately represents the matrix $L$. Here, we once again rely on ridge leverage score sampling which has been effectively used for obtaining good Nyström approximations in a number of prior works such as [AM15, CL17, RCCR18].

While our main algorithm can sample only from the random-size DPP, and not from the fixed-size $k$-DPP, we present a rigorous reduction argument which lets us use our DPP algorithm to sample exactly from a $k$-DPP (for any $k$) with a small computational overhead.

**Related work** Prior to our work, fast exact sampling from generic DPPs has been considered out of reach. The first procedure to sample general DPPs was given by [KKT12] and even more recent exact refinements [LGD18, Der19, Pou19], when the DPP is represented in the form of an $L$-ensemble, require preprocessing that amounts to an expensive $O(n^3)$ computation. Nonetheless, there are well-known samplers for very specific DPPs that are both fast and exact, for instance for sampling uniform spanning trees [AKKT13] provides approximate guarantee for the probability of any subset being sampled as a function of eigengaps of the $L$-kernel. Next, [LJS16a] construct coresets approximating a given $k$-DPP and then use them for sampling. In their Section 4.1, [LJS16a] show in which cases we can hope for a good approximation. These guarantees become tight if these approximations (Nyström subspace, coresets) are aligned with data. In our work, we aim for an adaptive approach that is able to provide a good approximation for any DPP.

As DPPs can be specified by kernels ($L$-kernels or $K$-kernels), a natural approximation strategy is to resort to low-rank approximations [KT11, GKT12, AKKT13, LJS16a]. For example, [AKKT13] provides approximate guarantee for the probability of any subset being sampled as a function of eigengaps of the $L$-kernel. Next, [LJS16a] construct coresets approximating a given $k$-DPP and then use them for sampling. In their Section 4.1, [LJS16a] show in which cases we can hope for a good approximation. These guarantees become tight if these approximations (Nyström subspace, coresets) are aligned with data. In our work, we aim for an adaptive approach that is able to provide a good approximation for any DPP.

The second class of approaches are based on Markov chain Monte-Carlo [MU49] techniques [Kan13, RK15, AGR16, LJS16b, GBV17]. There are known polynomial bounds on the mixing rates [DS91]
of MCMC chains with arbitrary DPPs as their limiting measure. In particular, [AGR16] showed them for cardinality-constrained DPPs and [LJS16b] for the general case. The two chains have mixing times which are, respectively, linear and quadratic in $n$ (see Table 1). Unfortunately, for any subsequent sample we need to wait until the chain mixes again.

Neither the known low-rank approximations or the known MCMC methods are able to provide samples that are exactly distributed according to a DPP (also called perfect sampling). This is not surprising as having scalable and exact sampling is very challenging in general. For example, methods based on rejection sampling are always exact, but they typically scale poorly to high-dimensional data and are adversely affected by the spikes in the distribution [EVCM16], resulting in high rejection rate and inefficiency. Surprisingly, our method is based on both low-rank approximation (a source of inaccuracy) and rejection sampling (a common source of inefficiency). In the following section, we show how to obtain a perfect DPP sampler from a Nyström approximation of the $L$-kernel. Then, to guarantee efficiency, in Section 3 we bound the number of rejections, which is possible thanks to the use of intermediate downsampling.

## 2 Exact sampling using any Nyström approximation

**Notation.** We use $[n]$ to denote the set $\{1, \ldots, n\}$. For a matrix $B \in \mathbb{R}^{n \times m}$ and index sets $C, D$, we use $B_{C,D}$ to denote the submatrix of $B$ consisting of the intersection of rows indexed by $C$ with columns indexed by $D$. If $C = D$, we use a shorthand $B_C$ and if $D = [m]$, we may write $B_{C,I}$. Finally, we also allow $C, D$ to be multisets or sequences, in which case each row/column is duplicated in the matrix according to its multiplicity (and in the case of sequences, we order the rows/columns as they appear in the sequence). Note that with this notation if $L = BB^T$ then $L_{C,D} = B_{C,I}B_{D,I}$.

As discussed in the introduction, our method relies on an intermediate downsampling distribution to reduce the size of the problem. The exactness of our sampler relies on the careful choice of that intermediate distribution. To that end, we use regularized determinantal processes, introduced by [Der19]. In the below definition, we adapt them to the kernel setting.

**Definition 3** Given an $n \times n$ psd matrix $L$, distribution $p \triangleq (p_1, \ldots, p_n)$ and $r > 0$, let $\tilde{L}$ denote an $n \times n$ matrix such that $\tilde{L}_{ij} \triangleq \frac{1}{r\sqrt{p_i p_j}} L_{ij}$ for all $i, j \in [n]$. We define R-DPP$_r^p(L)$ as a distribution over events $A \subseteq \bigcup_{k=0}^{\infty} [n]^k$, where

$$
\Pr(A) \triangleq \frac{\mathbb{E}_{\sigma} [\mathbb{1}_{\{\sigma \in A\}} \det(I + \tilde{L}_\sigma)]}{\det(I + L)}, \quad \text{for} \quad \sigma = (\sigma_1, \ldots, \sigma_t) \overset{i.i.d.}{\sim} p, \quad t \sim \text{Poisson}(r).
$$

Since the term $\det(I + \tilde{L}_\sigma)$ has the same form as the normalization constant of $\text{DPP}(\tilde{L}_\sigma)$, an easy calculation shows that the R-DPP can be used as an intermediate distribution in our algorithm without introducing any distortion in the sampling.

**Proposition 1** ([Der19, Theorem 8]) For any $L$, $p$, $r$, and $\tilde{L}$ defined as in Definition 3, if $\sigma \sim \text{R-DPP}_r^p(L)$ and $S \sim \text{DPP}(\tilde{L}_\sigma)$ then $\{\sigma : i \in S\} \sim \text{DPP}(L)$.

To sample from the R-DPP, our algorithm uses rejection sampling, where the proposal distribution is sampling i.i.d. proportionally to the approximate 1-ridge leverage scores $l_i \approx \tau_i(1)$ (see Definition 1 and the following discussion), computed using any Nyström approximation $\tilde{L}$ of matrix $L$. Apart from $\tilde{L}$, the algorithm also requires an additional parameter $q$, which controls the size of the intermediate sample. Because of rejection sampling and Proposition 1, the correctness of the algorithm does not depend on the choice of $\tilde{L}$ and $q$, as demonstrated in the following result. The key part of the proof involves showing that the acceptance probability in Line 6 is bounded by 1. Here, we obtain a considerably tighter bound than the one achieved by [Der19], which allows us to use a much smaller intermediate sample $\sigma$ (see Section 3) while maintaining the efficiency of rejection sampling.

**Theorem 2** Given a psd matrix $L$, any one of its Nyström approximations $\tilde{L}$ and any positive $q$, DPP-VFX (Algorithm 1) returns $S \sim \text{DPP}(L)$.

An important implication of Theorem 2 is that even though the choice of $\tilde{L}$ affects the overall execution of the algorithm, it does not affect the distribution of the output. Therefore we can reuse
Algorithm 1 DPP-VFX sampling $S \sim \text{DPP}(L)$

**Input:** $L \in \mathbb{R}^{n \times n}$, its Nyström approximation $\hat{L}$, $q > 0$

1. Compute $l_i = [(L - \hat{L}) + \hat{L}(I + \hat{L})^{-1}]_{ii} \approx \Pr(i \in S)$
2. Initialize $s = \sum_i l_i$, $z = \text{tr}(\hat{L}(I + \hat{L})^{-1})$, $\hat{L} = \frac{1}{q} \left[ \frac{1}{\sqrt{l_i}} (I - L)_{ij} \right]_{ij}$
3. repeat
4. sample $t \sim \text{Poisson}(qe^{s/q})$
5. sample $\sigma_1, \ldots, \sigma_t \overset{i.i.d.}{\sim} (\frac{L_{11}}{s}, \ldots, \frac{L_{tt}}{s})$
6. until $\text{Acc} \sim \text{Bernoulli}(\exp(-\det(I + L_i) - \text{det}(I + L)))$
7. sample $S \sim \text{DPP}(L_{\sigma})$
8. return $S = \{\sigma_i : i \in S\}$

the same $\hat{L}$ to produce multiple independent samples $S_1, S_2, \ldots \sim \text{DPP}(L)$, which we prove in Appendix A. This is particularly important because, as we will see in Section 3, DPP-VFX can generate successive samples $S_2, S_3, \ldots$ much more cheaply than the first sample $S_1$.

**Lemma 1** Let $C \subseteq [n]$ be a random set variable with any distribution. Suppose that $S_1$ and $S_2$ are returned by two executions of DPP-VFX, both using inputs constructed from the same $L$ and $\hat{L} = L_{\mathcal{I} \cap C, \mathcal{I} \cap C} = \mathbf{B}\mathbf{B}^{\top}$. Then $S_1$ and $S_2$ are (unconditionally) independent.

Before proceeding with the proof, we highlight for clarity that the Poisson r.v. $t$ in DPP-VFX (Algorithm 1, Line 4) has a different role than the Poisson r.v. used in the definition of R-DPPs. As a result, the definition of $\hat{L}$ (Line 2) is not exactly the one that would be suggested by Definition 3. In particular, DPP-VFX uses $q e^{s/q}$ instead of just $q$ as the mean parameter for $t$. The extra $e^{s/q}$ factor is necessary to correct for the rejection sampling step in Line 6 during the sampling loop, see the proof below.

**Proof of Theorem 2** We start by showing that the Bernoulli probability in Line 6 is bounded by 1. Note that this is important not only to sample correctly, but also when we later establish the efficiency of the algorithm. If we showed a weaker upper bound, say $c > 1$, we could always divide the expression by $c$ and retain the correctness, however it would also be $c$ times less likely that $\text{Acc} = 1$.

Since $\hat{L}$ is a Nyström approximation for some $C \subseteq \mathcal{I}$, it can be written as

$$\hat{L} = L_{\mathcal{I} \cap C, \mathcal{I} \cap C} = \mathbf{B}\mathbf{B}^{\top}$$

for any $\mathbf{B}$ such that $L = \mathbf{B}\mathbf{B}^{\top}$ where $\mathbf{P} \triangleq \mathbf{B}^{\top} \mathbf{L}_{\mathcal{I} \cap C, \mathcal{I} \cap C}$ is a projection, so that $\mathbf{P}^2 = \mathbf{P}$. Let $\mathbf{B} \triangleq \tilde{B}\mathbf{B}^{\top}$ where the $i$th row of $\mathbf{B}$ is the rescaled $i$th row of $\mathbf{B}$, i.e. $\tilde{b}_i \triangleq \frac{1}{\sqrt{s_i}} b_i$. Then, we have

$$\frac{\det(I + \mathbf{B}^{\top})}{\det(I + \hat{L})} = \frac{\det(I + \tilde{B}^{\top} \tilde{B})}{\det(I + \tilde{B}^{\top} \tilde{B})} = \det((I + \tilde{B}^{\top} \tilde{B})^{-1})$$

$$= \det(I + (\tilde{B}^{\top} \tilde{B})^{-1} \mathbf{P}^{\top} \mathbf{P}^{-1})$$

$$\leq \exp\left(\text{tr}\left((\tilde{B}^{\top} \tilde{B})^{-1} \mathbf{P}^{\top} \mathbf{P}^{-1}\right)^{-1}\right)$$

$$= \exp\left(\sum_{i=1}^{t} \frac{s}{q \sigma_i} \left[B(I + \mathbf{P} \mathbf{B}^{\top})^{-1} \mathbf{B}^{\top}\right]_{\sigma_i, \sigma_i}\right)  e^{-z} = e^{ts/q} e^{-z},$$

where the last equality follows because

$$B(I + \mathbf{P} \mathbf{B}^{\top})^{-1} = B(I - \mathbf{P} \mathbf{B}^{\top})^{-1} = L - \hat{L}(I + \hat{L})^{-1} = L - \hat{L} + \hat{L}(I + \hat{L})^{-1}.$$
Thus, we showed that the expression in Line 6 is valid. Let $\bar{\sigma}$ denote the random variable distributed as $\sigma$ is after exiting the repeat loop. It follows that

$$\Pr(\bar{\sigma} \in A) \propto \mathbb{E}_\sigma \left[ 1_{[\sigma \in A]} \frac{e^{x} \det(I + \tilde{L}_\sigma)}{e^{ts/q} \det(I + \tilde{L})} \right] \propto \sum_{t=0}^{\infty} \left( \frac{q e^{s/q}}{e^{ts/q} t!} \right) \cdot e^{-ts/q} \mathbb{E}_\sigma \left[ 1_{[\sigma \in A]} \det(I + \tilde{L}_\sigma) | t \right]$$

$$\propto \mathbb{E}_{t'} \left[ \mathbb{E}_\sigma \left[ 1_{[\sigma \in A]} \det(I + \tilde{L}_\sigma) | t = t' \right] \right]$$

for $t' \sim \text{Poisson}(q)$.

which shows that $\bar{\sigma} \sim \text{R-DPP}^q(L)$ for $l = \left( \frac{t_1}{s}, \ldots, \frac{t_n}{s} \right)$. The claim follows from Proposition 1.

### 3 Conditions for fast sampling

The complexity cost of DPP-VFX can be roughly summarized as follows: we pay a large one-time cost to precompute $\tilde{L}$ and all its associated quantities, and then we pay a smaller cost in the rejection sampling scheme which must be multiplied by the number of times we repeat the loop until acceptance. We first show that if $\tilde{L}$ is sufficiently close to $L$ then we will exit the loop with high probability. We then analyze how accurate the precomputing step needs to be to satisfy this condition. Finally, we bound the overall computational cost, which includes also the final step where we sample $S$ out of the intermediate sample $\sigma$ using any off-the-shelf exact DPP sampler.

**1) Bounding the number of rejections** The following result presents the two conditions needed for achieving efficient rejection sampling in DPP-VFX. First, the Nyström approximation needs to be accurate enough, and second, the intermediate sample size (controlled by parameter $q$) needs to be $\Omega(k^2)$. This is a significant improvement over the guarantee of $\text{[Der19]}$ where the intermediate sample size is $\Omega(\text{rank}(L) \cdot k)$, which is only meaningful for low-rank kernels. The main novelty in this proof comes from showing the following lower bound on the ratio of the determinants of $I + L$ and $I + \tilde{L}$, when $\tilde{L}$ is a Nyström approximation: $\det(I + L) / \det(I + \tilde{L}) \geq e^{k - z}$, where $z \triangleq \text{tr}(\tilde{L}(I + \tilde{L})^{-1})$. Remarkably, this bound exploits the fact that any Nyström approximation of $L = BB^T$ can be written as $\tilde{L} = BPB^T$, where $P$ is a projection matrix. Note that while our result holds in the worst case, in deployment, the conditions on $\tilde{L}$ and on $q$ can be considerably relaxed.

**Theorem 3** If the Nyström approximation $\tilde{L}$ and the intermediate sample size parameter $q$ satisfy

$$\text{tr}(L(I + L)^{-1}L - \tilde{L}(I + L)^{-1}\tilde{L}) \leq 1 \quad \text{and} \quad q \geq \max\{s^2, s\},$$

then $\Pr(\text{Acc} = \text{true}) \geq e^{-z}$. Therefore, with probability $1 - \delta$, Algorithm 1 exits the rejection sampling loop after at most $\mathcal{O}(\log(1/\delta))$ iterations and, after precomputing all of the inputs, the time complexity of the rejection sampling loop is $\mathcal{O}(k^6 \log^{-1} + \log^4 \delta^{-1}).$

**Proof** We first bound the number of rejections, and then discuss the complexity of each loop iteration. Let $\sigma$ be distributed as in Line 5. The probability of exiting the repeat loop at each iteration is

$$P \triangleq \mathbb{E}_\sigma \left[ \frac{e^{x} \det(I + \tilde{L}_\sigma)}{e^{ts/q} \det(I + \tilde{L})} \right] = \sum_{t=0}^{\infty} \left( \frac{q e^{s/q}}{e^{ts/q} t!} \right) \cdot e^{-ts/q} \mathbb{E}_\sigma \left[ \det(I + \tilde{L}_\sigma) | t \right]$$

$$= \frac{e^{q e^{s/q} + z}}{\det(I + \tilde{L})} \sum_{t=0}^{\infty} \frac{q^t}{e^{tell}} \mathbb{E}_\sigma \left[ \det(I + \tilde{L}_\sigma) | t \right] = e^{q e^{s/q} + z} \frac{\det(I + L)}{\det(I + \tilde{L})},$$

where the last equality follows because the infinite series computes the normalization constant of $\text{R-DPP}^q(L)$ given in Definition 3. If $s \geq 1$ then $q = s^2$ and the inequality $e^{x} \leq 1 + x + x^2$ for $x \in [0, 1]$ implies that $q < q e^{s/q} + z = s^2(1 - e^{1/s} + 1/s) + z - s \geq -1 + z - s$. On the other hand, if $q = s \in [0, 1]$, then $q < q e^{s/q} + z < (1 - e)s + z \geq -1 + z - s$. We proceed to lower bound the determinantal ratio. Here, let $L = BB^T$ and $\tilde{L} = BPB^T$, where $P$ is a projection matrix. Then,

$$\frac{\det(I + L)}{\det(I + \tilde{L})} = \frac{\det(I + B^TB)}{\det(I + PB^TBP)} = \frac{\det(I - (B^TB - PB^TBP)(I + B^TB)^{-1})^{-1}}{\det(I + B^TB)^{-1}},$$

$$\geq \exp\left( \text{tr}((B^TB - PB^TBP)(I + B^TB)^{-1}) \right)$$

$$= \exp\left( \text{tr}(B(I + B^TB)^{-1}B^T) - \text{tr}(BP(I + B^TB)^{-1}PB) \right),$$
Then, with probability

\[
\exp\left(k - \text{tr}(BP(I + B'B - B'B)(I + B'B)^{-1}PB')\right)
\]

we can now compute the remaining preprocessing costs, given a Nyström approximation where we can reformulate

\[
\text{DPP}(\mathbb{D}) \quad \text{Bounding the overall cost}
\]

there exist many algorithms to sample columns proportionally to their RLSs. For example, we can

\begin{equation}
\text{Theorem } L \text{ denotes }
\end{equation}

\[
\text{We can now fully characterize the computational cost.}
\]

\[
\text{Theorem 1 (restated for DPPs only) For a psd } n \times n \text{ matrix } L, \text{ let } S_1, S_2 \text{ be i.i.d. random sets from DPP}(L). \text{ Denote with } \hat{L} \text{ a Nyström approximation of } L \text{ obtained by sampling } m = \mathcal{O}(k^3 \log(n/\delta)) \text{ of its columns proportionally to their RLSs. If } q \geq \max\{s^2, s\}, \text{ then w.p. } 1 - \delta, \text{ DPP-VFX returns}
\]

\[
a) \text{ subset } S_1 \text{ in: } \mathcal{O}(nk^6 \log^{2}n/\delta + k^9 \log^{3}n/\delta + k^5 \log^{4}n/\delta) \text{ time},
\]

\[
b) \text{ then, } S_2 \text{ in: } \mathcal{O}(k^6 \log^{1}n + \log^{4}n) \text{ time}.
\]
Discussion The above result follows from the bounds on the precompute costs, on the number of iterations, on the iterations cost, and on the fact that the final DPP sampling step \( \tilde{S} \sim \text{DPP}(L_{\alpha}) \) requires \( O(t^3) \leq O((k^2 + \log \delta^{-1})^3) \) time. Note however that due to the nature of rejection sampling, as long as we exit the loop, i.e., we accept the sample, the output of DPP-VFX is guaranteed to follow the DPP distribution for any value of \( m \) and \( q \). In Theorem 1 we set \( m = (k^3 \log \frac{n}{q}) \) and \( q \geq \max\{s^2, s\} \) to satisfy Theorem 3 and guarantee a constant acceptance probability in the rejection sampling loop, but this might not be necessary or even desirable in practice. Experimentally, much smaller values of \( m \), starting from \( m = \Omega(k \log \frac{n}{q}) \) seem to be sufficient to accept the sample, while at the same time a smaller \( m \) greatly reduces the preprocessing costs. In general, we recommend to separate DPP-VFX in three phases. First, compute an accurate estimate of the RLS using off-the-shelf algorithms in \( O(nk^2 \log^2 \frac{n}{q} + k^3 \log^4 \frac{n}{q}) \) time. Then, sample a small number \( m \) of columns to construct an explorative \( \hat{L} \), and try to run DPP-VFX. If the rejection sampling loop does not terminate sufficiently fast, then we can reuse the RLS estimates to compute a more accurate \( \hat{L} \) for a larger \( m \). Using a simple doubling schedule for \( m \), this strategy quickly reaches a regime where DPP-VFX is w.h.p. guaranteed to accept, resulting in faster sampling.

4 Reduction from DPPs to k-DPPs

We next show that with a simple extra rejection sampling step we can efficiently transform any exact DPP sampler into an exact \( k \)-DPP sampler.

A common heuristic to sample \( S \) from a \( k \)-DPP is to first sample \( S \) from a DPP, and then reject the sample if the size of \( S \) is not exactly \( k \). As we show in this section, the success probability of this procedure can be improved by appropriately rescaling \( \hat{L} \) by a constant factor \( \alpha \),

\[
\text{sample } S_\alpha \sim \text{DPP}(\alpha L), \text{ accept if } |S_\alpha| = k.
\]

Note that rescaling the DPP by a constant \( \alpha \) as above only changes the expected size of the set \( S_\alpha \), and not its distribution. Therefore, if we accept only sets with size \( k \), we will be sampling exactly from our \( k \)-DPP. Moreover, if \( k_\alpha = \mathbb{E}[|S_\alpha|] \) is close to \( k \), the success probability will improve. With a slight abuse of notation, in the context of \( k \)-DPPs we will indicate with \( k \) the desired size of \( S_\alpha \), i.e., the final output size at acceptance, and with \( k_\alpha = \mathbb{E}[|S_\alpha|] \) the expected size of the scaled DPP.

While the above rejection sampling heuristic is widespread, until now there has been no proof that this heuristic can provably succeed in few rejections. We solve this open question with two new results. First, we show that for an appropriate rescaling \( \alpha^* \) we only reject \( S_\alpha \) roughly \( O(\sqrt{k}) \) times. Then, we show how to find such an \( \alpha^* \) with a \( \tilde{O}(n \cdot \text{poly}(k)) \) time preprocessing step.

Theorem 4 There exists constant \( C > 0 \) such that for any rank \( n \) psd matrix \( L \) and \( k \in [n] \), there is \( \alpha^* > 0 \) with the following property: if we sample \( S_{\alpha^*} \sim \text{DPP}(\alpha^* L) \), then \( \Pr(|S_{\alpha^*}| = k) \geq \frac{1}{C \sqrt{k}} \).

The proof in Appendix B. relies on a known Chernoff bound for the sample size \( |S_\alpha| \) of a DPP. When applied naïvely, the inequality does not offer a lower bound on the probability of any single sample size. However we show that the probability mass is concentrated on \( O(\sqrt{k} \alpha) \) sizes. This leads to a lower bound on the sample size with the largest probability, i.e., the mode of the distribution. Then, it remains to observe that for any \( k \in [n] \) we can always find \( \alpha^* \) for which \( k \) is the mode of \( |S_{\alpha^*}| \). We conclude that given \( \alpha^* \), the rejection sampling scheme described above transforms any \( \text{poly}(k) \) time DPP sampler into a \( \text{poly}(k) \) time \( k \)-DPP sampler. It remains to efficiently find \( \alpha^* \), which once again relies on using a Nyström approximation of \( \hat{L} \).

Lemma 4 If \( k \geq 1 \), then there is an algorithm that finds \( \alpha^* \) in \( \tilde{O}(n \cdot \text{poly}(k)) \) time.

While the existence proof of \( \alpha^* \) is general and based on simple unimodality, this characterization is not sufficient to control the mode of the DPP when \( \alpha^* \) is perturbed, as it happens during an approximate optimization of \( \alpha \). However, for DPPs the mode can be expressed as a Poisson binomial random variable [Hö66] based on the spectrum of \( L \), which can be controlled [Dö64] using the approximate spectrum of \( \hat{L} \). While our analysis shows that in the worst case a much more accurate \( \hat{L} \) is necessary compared to Theorem 3, in practice the same \( m = \Omega(k \log \frac{n}{q}) \) seems to suffice.
5 Experiments

In this section, we experimentally evaluate the performance of DPP-VFX and compare it to exact sampling [HKP+06] and MCMC-based approaches [AGR16]. In particular, we are only interested in evaluating computational performance, i.e., how DPP-VFX and baselines scale with the size $n$ of the matrix $L$. Note that an inexact DPP sampler, e.g., MCMC samplers or previous approximate rejection samplers [Der19], also had another metric to validate, i.e., they had to empirically show that their samples were close enough to a DPP distribution. However, Section 2 proves that DPP-VFX’s output is distributed exactly according to the DPP and therefore it is strictly equivalent to any other exact DPP sampler, e.g., the Gram-Schmidt [Gil14] or dual [KT12] samplers implemented in DPPy.

We use the infinite MNIST digits dataset (i.e., MNIST8M [LCB07]) as input data, where $n$ varies up to $10^6$ and $d = 784$. All algorithms, including DPP-VFX, are implemented in python as part of the DPPy library [GBV19]. All experiments are carried out on a 24-core CPU and fully take advantage of potential parallelization. For each experiment we report the mean and a 95% confidence interval over 10 runs of each algorithm’s runtime.

This first experiment compares the time required to generate a first sample. We consider subsets of MNIST8M that go from $n = 10^3$ to $n = 7 \cdot 10^4$, i.e., the original MNIST dataset, and use an RBF kernel with $\sigma = \sqrt{3d}$ to construct $L$. For the Nyström approximation we set $m = 10d_{\text{eff}}(1) \approx 10k$. While this is much lower than the $O(k^3)$ value suggested by the theory, as we will see it is already accurate enough to result in drastic runtime improvements over exact and MCMC. Following the strategy of Section 4, for each algorithm we control the size of the output set by rescaling the input matrix $L$ by a constant $\alpha$ such that $\mathbb{E}[|S_{\alpha^*}|] = 10$. Results are reported in Figure 1.

Exact sampling is performed using eigendecomposition and DPPy’s default Gram-Schmidt [Gil14] sampler. It is clearly cubic in $n$, and we could not push it beyond $n = 1.5 \cdot 10^4$. For MCMC, we enforce mixing by running the chain for $n\alpha$ steps, the minimum recommended by [AGR16]. However, for $n = 7 \cdot 10^5$ the MCMC runtime is 160s and exceeds the plot’s limit, while DPP-VFX completes in 16s, an order of magnitude faster. Moreover, DPP-VFX rarely rejects more than 10 times, and the mode of the rejections up to $n = 7 \cdot 10^5$ is 1, i.e., we mostly accept at the first iteration.

Since in Figure 1, the resampling time of both exact and DPP-VFX are negligible, we investigate the cost of a second sample, i.e., of resampling, on a much larger subset of MNIST8M up to $n = 10^6$, normalized to have maximum row norm equal to 1. In this case it is not possible to perform an eigendecomposition of $L$, so we replace RBFs with a linear kernel, for which the input $X$ represents a factorization $L = XX^\top$ and we use an exact dual sampler on $X$ [KT12]. However, as Figure 2 shows, even with a special algorithm for this simpler setting the resampling process still scales with $n$. On the other hand, DPP-VFX’s complexity (after preprocessing) is light years better as it scales only with $k$ and remains constant regardless of $n$.

---

3The code used for these experiments is available at https://github.com/LCSL/dpp-vfx.
4For simplicity we do not perform the full $k$-DPP rejection step, but only adjust the expected size of the set.
Acknowledgements

MD thanks the NSF for funding via the NSF TRIPODS program. This material is based upon work supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216, and the Italian Institute of Technology. We gratefully acknowledge the support of NVIDIA Corporation for the donation of the Titan Xp GPUs and the Tesla k40 GPU used for this research.

References


