Faster Greedy MAP Inference for Determinantal Point Processes

Insu Han [1] Prabhanjan Kambadur [2] Kyoungsoo Park [1] Jinwoo Shin [1]

Abstract

Determinantal point processes (DPPs) are popular probabilistic models that arise in many machine learning tasks, where distributions of diverse sets are characterized by matrix determinants. In this paper, we develop fast algorithms to find the most likely configuration (MAP) of large-scale DPPs, which is NP-hard in general. Due to the submodular nature of the MAP objective, greedy algorithms have been used with empirical success. Greedy implementations require computation of log-determinants, matrix inverses or solving linear systems at each iteration. We present faster implementations of the greedy algorithms by utilizing the complementary benefits of two log-determinant approximation schemes: (a) first-order expansions to the matrix log-determinant function and (b) high-order expansions to the scalar log function with stochastic trace estimators. In our experiments, our algorithms are significantly faster than their competitors for large-scale instances, while sacrificing marginal accuracy.

1. Introduction

Determinantal point processes (DPPs) are elegant probabilistic models, first introduced by (Macchi, 1975), who called them 'fermion processes'. Since then, DPPs have been extensively studied in the fields of quantum physics and random matrices (Johansson, 2006), giving rise to a beautiful theory (Daley & Vere-Jones, 2007). The characteristic of DPPs is repulsive behavior, which makes them useful for modeling diversity.

Recently, they have been applied in many machine learning tasks such as summarization (Gong et al., 2014), human pose detection (Kulesza et al., 2012), clustering (Kang, 2013) and tweet time-line generation (Yao et al., 2016). In particular, their computational advantage compared to other probabilistic models is that many important inference tasks are computationally tractable. For example, conditioning, sampling (Kang, 2013) and marginalization of DPPs admit polynomial-time/efficient algorithms, while those on popular graphical models (Jordan, 1998) do not, i.e., they are NP-hard. One exception is the MAP inference (finding the most likely configuration), which is our main interest; the MAP computation is known to be NP-hard even for DPPs (Kulesza et al., 2012).

The distribution of diverse sets under DPPs is characterized by determinants of submatrices formed by their features, and the corresponding MAP inference reduces to finding a submatrix that maximizes its determinant. It is well known that the matrix log-determinant is a submodular function; that is, the MAP inference of DPPs is a special instance of submodular maximization (Kulesza et al., 2012). Greedy algorithms have been shown to have the best worst-case approximation guarantees for many instances of submodular maximization; for example, $(1 - 1/e)$-approximation for monotone functions. Furthermore, it has been often empirically observed that greedy algorithms provide near optimal solutions (Krause et al., 2008). Hence, greedy algorithms have been also applied for the DPP task (Kulesza et al., 2012; Yao et al., 2016; Zhang & Ou, 2016). Known implementations of greedy selection on DPP require computation of log-determinants, matrix inversions (Kulesza et al., 2012) or solving linear systems (Li et al., 2016b). Consequently, they run in $O(d^4)$ time where d is the total number of items (see Section 2.3). In this paper, we propose faster greedy implementations that run in $O(d^3)$ time.

Contribution. Our high-level idea is to amortize greedy operations by utilizing log-determinant approximation schemes. A greedy selection requires computation of marginal gains of log-determinants; we consider their first-order (linear) approximations. We observe that the computation of multiple marginal gains can be amortized into a single run of a linear solver, in addition to multiple vector inner products. We choose the popular conjugate gradient descent (`CG`) (Saad, 2003) as a linear solver. In addition, for improving the quality of first-order approximations, we partition remaining items into $p \geq 1$ sets (via some cluster-

[1] School of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Republic of Korea. [2] Bloomberg LP, 731 Lexington Avenue, New York, NY, 10069. Correspondence to: Jinwoo Shin <jinsoos@kaist.ac.kr>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

ing algorithm), and apply the first-order approximations in each partition. The resulting approximate computation of multiple marginal gains at each greedy selection requires $2p$ runs of CG under the Schur complement, and the overall running time of the proposed greedy algorithm becomes $O(d^3)$ under the choice of $p = O(1)$ (see Section 3).

Next, for larger-scale DPPs, we develop an even faster greedy algorithm using a batch strategy. In addition to using the first-order approximations of log-determinants under a partitioning scheme, we add $k > 1$ elements instead of a single element to the current set, where we sample some candidates among all possible k elements to relax the expensive cost of computing all marginal gains. Intuitively, the random batch selection makes the algorithm k times faster, while potentially hurting the approximation quality. Now, we suggest running the recent fast log-determinant approximation scheme (LDAS) (Han et al., 2015) p times, instead of running CG pk times under the Schur complement, where LDAS utilizes high-order, i.e., polynomial, approximations to the scalar log function with stochastic trace estimators. Since the complexities of running LDAS and CG are comparable, running the former p times is faster than running the latter pk times if $k > 1$.

Finally, we discovered a novel scheme for boosting the approximation quality by sharing random vectors among many runs of LDAS, and also establish theoretical justification why this helps. Our experiments on both synthetic and real-world dataset show that the proposed algorithms are significantly faster than competitors for large-scale instances, while losing marginal approximation ratio.

Related work. To the best of our knowledge, this is the first work that aims for developing faster greedy algorithms specialized for the MAP inference of DPP, while there has been several efforts on those for general submodular maximization. An accelerated greedy algorithm, called lazy evaluation, was first proposed by (Minoux, 1978) which maintains the upper bounds on the marginal gains instead of recomputing exact values. In each iteration, only elements with the maximal bound compute the exact gain, which still bounds on the exact value due to submodularity. For the DPP case, we also observe that the lazy algorithm is significantly faster than the standard greedy one, while the outputs of both are equal. Hence, we compare our algorithms with the lazy one (see Section 5).

Another natural approach is on stochastic greedy selections computing marginal gains of randomly selected elements. Its worst-case approximation guarantee was also studied (Mirzasoleiman et al., 2015), under the standard, non-batch, greedy algorithm. The idea of stochastic selections can be also applied to our algorithms, where we indeed apply it for designing our faster batch greedy algorithm as mentioned earlier. Recently, (Buchbinder et al., 2015) proposed a 'one-pass' greedy algorithm where each greedy selection requires computing only a single marginal gain, i.e., the number of marginal gains necessary to compute can be significantly reduced. However, this algorithm is attractive only for the case when evaluating a marginal gain does not increase with respect to the size of the current set, which does not hold for the DPP case. As reported in Section 5, it performs significantly worse than ours in both their approximation qualities and running times.

There have been also several efforts to design parallel/distributed implementations of greedy algorithms: (Pan et al., 2014) use parallel strategies for the above one-pass greedy algorithm and (Kumar et al., 2015) adapt a MapReduce paradigm for implementing greedy algorithms in distributed settings. One can also parallelize our algorithms easily since they require independent runs of matrix-vector (or vector inner) products, but we do not explore this aspect in this paper. Finally, we remark that a non-greedy algorithm was studied in (Gillenwater et al., 2012) for better MAP qualities of DPP, but it is much slower than ours as reported in Section 5.

2. Preliminaries

We start by defining a necessary notation. Our algorithms for determinantal point processes (DPPs) select elements from the ground set of d items $\mathcal{Y} = [d] := \{1, 2, \dots, d\}$ and denote the set of all subsets of $\mathcal{Y}$ by $2^{\mathcal{Y}}$. For any positive semidefinite matrix $L \in \mathbb{R}^{d\times d}$, we denote $\lambda_{\min}$ and $\lambda_{\max}$ to be the smallest and the largest eigenvalues of L. Given subset $X, Y \subseteq \mathcal{Y}$, we use $L_{X,Y}$ to denote the submatrix of L obtained by entries in rows and columns indexed by X and Y, respectively. For notational simplicity, we let $L_{X,X} = L_X$ and $L_{X,\{i\}} = L_{X,i}$ for $i \in \mathcal{Y}$. In addition, $\overline{L}_X$ is defined as the average of $L_{X\cup\{i\}}$ for $i \in \mathcal{Y} \setminus X$. Finally, $\langle \cdot, \cdot \rangle$ means the matrix/vector inner product or element-wise product sum.

In Section 2.1, we introduce the *maximum a posteriori* (MAP) inference of DPP, then the standard greedy optimization scheme and its naïve implementations are described in Section 2.2 and Section 2.3, respectively.

2.1. Determinantal Point Processes

DPPs are probabilistic models for subset selection of a finite ground set $\mathcal{Y} = [d]$ that captures both quality and diversity. Formally, it defines the following distribution on $2^{\mathcal{Y}}$: for random variable $\mathbf{X} \subseteq \mathcal{Y}$ drawn from given DPP, we have

$$\Pr[\mathbf{X} = X] \propto \det(L_X),$$

where $L \in \mathbb{R}^{d\times d}$ is a positive definite matrix called an *L-ensemble* kernel. Under the distribution, several probabilistic inference tasks are required for real-world applications, including MAP (Gong et al., 2014; Gillenwater et al., 2012; Yao et al., 2016), sampling (Kathuria & Deshpande, 2016; Kang, 2013; Li et al., 2016a), marginalization and conditioning (Gong et al., 2014). In particular, we are interested in the MAP inference, i.e., finding the most diverse subset Y of $\mathcal{Y}$ that achieves the highest probability, i.e., $\arg\max_{Y \subseteq \mathcal{Y}} \det(L_Y)$, possibly under some constraints on Y. Unlike other inference tasks on DPP, it is known that MAP is a NP-hard problem (Kulesza et al., 2012).

2.2. Greedy Submodular Maximization

A set function $f : 2^{\mathcal{Y}} \rightarrow \mathbb{R}$ is submodular if its marginal gains are decreasing, i.e.,

$$f(X \cup \{i\}) - f(X) \geq f(Y \cup \{i\}) - f(Y),$$

for every $X \subseteq Y \subset \mathcal{Y}$ and every $i \in \mathcal{Y} \setminus Y$. We say f is monotone if $f(X) \leq f(Y)$ for every $X \subseteq Y$. It is well known that DPP has the submodular structure, i.e., $f = \log\det$ is submodular.

The submodular maximization task is to find a subset maximizing a submodular function f, which corresponds to the MAP inference task in the DPP case. Hence, it is NP-hard and a popular approximate scheme is the following greedy procedure (Nemhauser et al., 1978): initially, $X \leftarrow \emptyset$ and iteratively update $X \leftarrow X \cup \{i_{\max}\}$ for

$$i_{\max} = \operatorname*{argmax}_{i \in \mathcal{Y} \setminus X} f(X \cup \{i\}) - f(X), \tag{1}$$

as long as $f(X \cup \{i_{\max}\}) > f(X)$. For the monotone case, it guarantees $(1 - 1/e)$-approximation (Nemhauser et al., 1978). Under some modifications of the standard greedy procedure, $2/5$-approximation can be guaranteed even for non-monotone functions (Feige et al., 2011). Irrespectively of such theoretical guarantees, it has been empirically observed that greedy selection (1) provides near optimal solutions in practice (Krause et al., 2008; Sharma et al., 2015; Yao et al., 2016; Zhang & Ou, 2016).

2.3. Naïve Implementations of Greedy Algorithm

Log-determinant or related computations, which are at the heart of greedy algorithms for MAP inference of DPPs, are critical to compute the marginal gain $\log\det L_{X\cup\{i\}} - \log\det L_X$. Since the exact computations of log-determinants might be slow, i.e., requires $O(d^3)$ time for d-dimensional matrices, we introduce recent efficient log-determinant approximation schemes (LDAS). The log-determinant of a symmetric positive definite matrix A can be approximated by combining (a) Chebyshev polynomial expansion of scalar log function and (b) matrix trace estimators via Monte Carlo methods:

$$\log\det A = \mathtt{tr}\left(\log A\right) \overset{(a)}{\approx} \mathtt{tr}\left(p_n(A)\right) \overset{(b)}{\approx} \frac{1}{m}\sum_{t=1}^{m} \mathbf{v}^{(t)\top} p_n(A) \mathbf{v}^{(t)}.$$

Here, $p_n(x)$ is a polynomial expansion of degree n approximating $\log x$ and $\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(m)}$ are random vectors used for estimating the trace of $p_n(A)$. Several polynomial expansions, including Taylor (Boutsidis et al., 2015), Chebyshev (Han et al., 2015) and Legendre (Peng & Wang, 2015) have been studied. For trace estimation, several random vectors have been also studied (Avron & Toledo, 2011), e.g., the Hutchinson method (Hutchinson, 1990) chooses elements of $\mathbf{v}$ as i.i.d. random numbers in $\{-1, +1\}$ so that $\mathbf{E}\left[\mathbf{v}^\top A \mathbf{v}\right] = \mathtt{tr}\left(A\right)$. In this paper, we use LDAS using the Chebyshev polynomial and Hutchinson method (Han et al., 2015), but one can use other alternatives as well.

Log-determinant Approximation Scheme (LDAS) (Han et al., 2015)

```
Input: symmetric matrix $A \in \mathbb{R}^{d\times d}$ with eigenvalues in $[\delta, 1-\delta]$, sampling number $m$ and polynomial degree $n$
Initialize: $\Gamma \leftarrow 0$
$c_j \leftarrow j$-th coefficient of Chebyshev expansion of $\log x$ on $[\delta, 1-\delta]$ for $0 \leq j \leq n$.
for $i = 1$ to $m$ do
    Draw a random vector $\mathbf{v}^{(i)} \in \{-1,+1\}^d$ whose entries are uniformly distributed.
    $\mathbf{w}_0^{(i)} \leftarrow \mathbf{v}^{(i)}$ and $\mathbf{w}_1^{(i)} \leftarrow \frac{2}{1-2\delta} A\mathbf{v}^{(i)} - \frac{1}{1-2\delta}\mathbf{v}^{(i)}$
    $\mathbf{u} \leftarrow c_0 \mathbf{w}_0^{(i)} + c_1 \mathbf{w}_1^{(i)}$
    for $j = 2$ to $n$ do
        $\mathbf{w}_2^{(i)} \leftarrow \frac{4}{1-2\delta} A\mathbf{w}_1^{(i)} - \frac{2}{1-2\delta}\mathbf{w}_1^{(i)} - \mathbf{w}_0^{(i)}$
        $\mathbf{u} \leftarrow \mathbf{u} + c_j\, \mathbf{w}_2^{(i)}$
        $\mathbf{w}_0^{(i)} \leftarrow \mathbf{w}_1^{(i)}$ and $\mathbf{w}_1^{(i)} \leftarrow \mathbf{w}_2^{(i)}$
    end for
    $\Gamma \leftarrow \Gamma + \mathbf{v}^{(i)\top}\mathbf{u}/m$
end for
Output: $\Gamma$
```

Observe that LDAS only requires matrix-vector multiplications and its running time is $\Theta\left(d^2\right)$ for constants $m, n = O(1)$. One can directly use LDAS for computing (1) and the resulting greedy algorithm runs in $\Theta(d \cdot T_{\mathtt{GR}}^3)$ time where the number of greedy updates on the current set X is $T_{\mathtt{GR}}$. Since $T_{\mathtt{GR}} = O(d)$, the complexity is simply $O(d^4)$. An alternative way to achieve the same complexity is to use the Schur complement (Ouellette, 1981):

$$\log\det L_{X\cup\{i\}} - \log\det L_X = \log\left(L_{i,i} - L_{i,X} L_X^{-1} L_{X,i}\right). \tag{2}$$

This requires a linear solver to compute $L_X^{-1} L_{X,i}$; conjugate gradient descent (CG) (Greenbaum, 1997) is a popular choice in practice. Hence, if one applies CG to compute the max-marginal gain (1), the resulting greedy algorithm runs in $\Theta(d \cdot T_{\mathtt{GR}}^3 \cdot T_{\mathtt{CG}})$ time, where $T_{\mathtt{CG}}$ denotes the number of iterations of each CG run. In the worst case, CG converges to the exact solution when $T_{\mathtt{CG}}$ grows with the matrix dimension, but for practical purposes, it typically provides a very accurate solution in few iterations, i.e., $T_{\texttt{CG}} = O(1)$. Recently, Gauss quadrature via Lanczos iteration is used for efficient computing of $L_{i,X} L_X^{-1} L_{X,i}$ (Li et al., 2016b). Although it guarantees rigorous upper/lower bounds, CG is faster and accurate enough for most practical purposes.

In summary, the greedy MAP inference of DPP can be implemented efficiently via LDAS or CG. The faster implementations proposed in this paper smartly employ both of them as key components utilizing their complementary benefits.

3. Faster Greedy DPP Inference

In this section, we provide a faster greedy submodular maximization scheme for the MAP inference of DPP. We explain our key ideas in Section 3.1 and then, provide the formal algorithm description in Section 3.2.

3.1. Key Ideas

First-order approximation of log-determinant. The main computational bottleneck of a greedy algorithm is to evaluate the marginal gain (1) for every element not in the current set. To reduce the time complexity, we consider the following first-order, i.e., linear, approximation of log-determinant as:[1]

$$\begin{aligned}
&\operatorname*{argmax}_{i \in \mathcal{Y}\setminus X} \log\det L_{X\cup\{i\}} - \log\det L_X \\
&= \operatorname*{argmax}_{i \in \mathcal{Y}\setminus X} \log\det L_{X\cup\{i\}} - \log\det \overline{L}_X \\
&\approx \operatorname*{argmax}_{i \in \mathcal{Y}\setminus X} \left\langle \overline{L}_X^{-1}, L_{X\cup\{i\}} - \overline{L}_X \right\rangle, \qquad (3)
\end{aligned}$$

where we recall that $\overline{L}_X$ is the average of $L_{X\cup\{i\}}$. Observe that computing (3) requires the vector inner product of a single column (or row) of $\overline{L}_X^{-1}$ and $L_{X\cup\{i\}} - \overline{L}_X$ because $L_{X\cup\{i\}}$ and $\overline{L}_X$ share almost all entries except a single row and a column.

To obtain a single column of $\overline{L}_X^{-1}$, one can solve a linear system using the CG algorithm. More importantly, it suffices to run CG once for computing (3), while the naïve greedy implementation in Section 2.3 has to run CG $|\mathcal{Y}\setminus X|$ times. As we mentioned earlier, after obtaining the single column of $\overline{L}_X^{-1}$ using CG, one has to perform $|\mathcal{Y}\setminus X|$ vector inner products in (3), but it is much cheaper than $|\mathcal{Y}\setminus X|$ CG runs requiring matrix-vector multiplications.

Partitioning. In order to further improve the quality of first-order approximation (3), we partition $\mathcal{Y}\setminus X$ into p distinct subsets so that

$$\|L_{X\cup\{i\}} - \overline{L}_X\|_F \;\gg\; \|L_{X\cup\{i\}} - \overline{L}_X^{(j)}\|_F,$$

where an element i is in the partition $j \in [p]$, $\overline{L}_X^{(j)}$ is the average of $L_{X\cup\{i\}}$ for i in the partition j, and $\|\cdot\|_F$ is the Frobenius norm. Since $L_{X\cup\{i\}}$ becomes closer to the average $\overline{L}_X^{(j)}$, one can expect that the first-order approximation quality in (3) is improved. But, we now need a more expensive procedure to approximate the marginal gain:

$$\begin{aligned}
&\log\det L_{X\cup\{i\}} - \log\det L_X \\
&= \left(\log\det L_{X\cup\{i\}} - \log\det \overline{L}_X^{(j)}\right) + \left(\log\det \overline{L}_X^{(j)} - \log\det L_X\right) \\
&\approx \underbrace{\left\langle \left(\overline{L}_X^{(j)}\right)^{-1}, L_{X\cup\{i\}} - \overline{L}_X^{(j)} \right\rangle}_{(a)} + \underbrace{\left(\log\det \overline{L}_X^{(j)} - \log\det L_X\right)}_{(b)}.
\end{aligned}$$

The first term (a) can be computed efficiently as we explained earlier, but we have to run CG p times for computing single columns of $\overline{L}_X^{(1)}, \dots, \overline{L}_X^{(p)}$. The second term (b) can be also computed using CG similarly to (2) under the Schur complement. Hence, one has to run CG $2p$ times in total. If p is large, the overall complexity becomes larger, but the approximation quality improves as well. We also note that one can try various clustering algorithms, e.g., k-means or Gaussian mixture. Instead, we use a simple random partitioning scheme because it is not only the fastest method but it also works well in our experiments.

3.2. Algorithm Description and Guarantee

The formal description of the proposed algorithm is described in Algorithm 1.

Algorithm 1 Faster Greedy DPP Inference

```
Input: kernel matrix $L \in \mathbb{R}^{d\times d}$ and number of partitions $p$
Initialize: $X \leftarrow \emptyset$
while $\mathcal{Y} \setminus X \neq \emptyset$ do
    Partition $\mathcal{Y} \setminus X$ randomly into $p$ subsets.
    for $j = 1$ to $p$ do
        $\overline{L}_X^{(j)} \leftarrow$ average of $L_{X\cup\{i\}}$ for $i$ in the partition $j$
        $\mathbf{z}^{(j)} \leftarrow$ $(|X|+1)$-th column of $\left(\overline{L}_X^{(j)}\right)^{-1}$
        $\Gamma_j \leftarrow \log\det \overline{L}_X^{(j)} - \log\det L_X$
    end for
    for $i \in \mathcal{Y} \setminus X$ do
        $\Delta_i \leftarrow \left\langle L_{X\cup\{i\}} - \overline{L}_X^{(j)}, \mathtt{Mat}\left(\mathbf{z}^{(j)}\right)\right\rangle$[2] $+ \Gamma_j$
        where element $i$ is included in partition $j$.
    end for
    $i_{\max} \leftarrow \operatorname{argmax}_{i\in\mathcal{Y}\setminus X} \Delta_i$
    if $\log\det L_{X\cup\{i_{\max}\}} - \log\det L_X < 0$ then
        return $X$
    end if
    $X \leftarrow X \cup \{i_{\max}\}$
end while
```

[1] $\nabla_X \log\det X = \left(X^{-1}\right)^\top$

As we explained in Section 3.1, the lines 7, 8 require to run CG. Hence, the overall complexity becomes $\Theta(T_{\texttt{GR}}^3 \cdot T_{\texttt{CG}} \cdot p + d \cdot T_{\texttt{GR}}^2) = \Theta(T_{\texttt{GR}}^3 + d \cdot T_{\texttt{GR}}^2)$, where we choose $p, T_{\texttt{CG}} = O(1)$. Since $T_{\texttt{GR}} = O(d)$, it is simply $O(d^3)$ and better than the complexity $O(d^4)$ of the naïve implementations described in Section 2.3. In particular, if kernel matrix L is sparse, i.e., number of non-zeros of each column/row is $O(1)$, ours has the complexity $\Theta(T_{\texttt{GR}}^2 + d \cdot T_{\texttt{GR}})$ while the naïve approaches are still worse having the complexity $\Theta(d \cdot T_{\texttt{GR}}^2)$.

We also provide the following approximation guarantee of Algorithm 1 for the monotone case, where its proof is given in the supplementary material.

Theorem 1. *Suppose the smallest eigenvalue of L is greater than 1. Then, it holds that*

$$\log\det L_X \geq (1 - 1/e) \max_{Z \subseteq \mathcal{Y}, |Z| = |X|} \log\det L_Z - 2|X|\varepsilon.$$

where

$$\varepsilon = \max_{\substack{X \subseteq \mathcal{Y}, i \in \mathcal{Y} \setminus X \\ j \in [p]}} \left| \log \frac{\det L_{X \cup \{i\}}}{\det \overline{L}_X^{(j)}} - \left\langle \left(\overline{L}_X^{(j)}\right)^{-1}, L_{X \cup \{i\}} - \overline{L}_X^{(j)} \right\rangle \right|$$

and X is the output of Algorithm 1.

The above theorem captures the relation between the first-order approximation error $\varepsilon > 0$ in (3) and the worst-case approximation ratio of the algorithm.

4. Faster Batch-Greedy DPP Inference

In this section, we present an even faster greedy algorithm for the MAP inference task of DPP, in particular for large-scale tasks. On top of ideas described in Section 3.1, we use a batch strategy, i.e., add k elements instead of a single element to the current set, where LDAS in Section 2.3 is now used as a key component. The batch strategy accelerates our algorithm. We first provide the formal description of the batch greedy algorithm in Section 4.1. In Section 4.2, we describe additional ideas on applying LDAS as a subroutine of the proposed batch algorithm.

4.1. Algorithm Description

Algorithm 2 Faster Batch-Greedy DPP Inference

```
1: Input: kernel matrix $L \in \mathbb{R}^{d \times d}$, number of partitions $p$, batch size $k$ and the number of batch samples $s$
2: Initialize: $X \leftarrow \emptyset$
3: while $\mathcal{Y} \setminus X$ is not empty do
4:     $I_i \leftarrow$ Randomly draw a batch of size $k$ for $i \in [s]$.
5:     Partition $[s]$ randomly into $p$ subsets.
6:     for $j = 1$ to $p$ do
7:         $\overline{L}_X^{(j)} \leftarrow$ average of $L_{X \cup I_i}$ for $i$ in the partition $j$
8:         $Z^{(j)} \leftarrow (|X| + 1)$ to $(|X| + k)$-th columns of $\left(\overline{L}_X^{(j)}\right)^{-1}$
9:         $\Gamma_j \leftarrow \log\det \overline{L}_X^{(j)}$ using LDAS.
10:    end for
11:    for $i = 1$ to $s$ do
12:        $\Delta_i^{\text{Batch}} \leftarrow \left\langle L_{X \cup I_i} - \overline{L}_X^{(j)}, \texttt{Mat}\left(Z^{(j)}\right) \right\rangle$[2] $+ \Gamma_j$
           where a batch index $i$ is included in $j$-th partition.
13:    end for
14:    $i_{\max} \leftarrow \operatorname{argmax}_{i \in [s]} \Delta_i^{\text{Batch}}$
15:    if $\log\det L_{X \cup I_{i_{\max}}} - \log\det L_X < 0$ then
16:        return $X$
17:    end if
18:    $X \leftarrow X \cup I_{i_{\max}}$
19: end while
```

The formal description of the proposed algorithm is described in Algorithm 2. Similar to the line 7 in Algorithm 1, the line 8 of Algorithm 2 can be solved by the CG algorithms. However, the line 9 of Algorithm 2 uses the LDAS and we remind that it runs in $\Theta(d^2)$ time. In addition, the line 12 requires the vector inner products ks times. Thus, the total complexity becomes $\Theta\left(T_{\texttt{GR}}^3 \cdot \left(T_{\texttt{CG}} + \frac{mn}{k}\right) \cdot p + s \cdot T_{\texttt{GR}}^2 + s \cdot T_{\texttt{CG}}\right) = \Theta(T_{\texttt{GR}}^3)$ where $T_{\texttt{GR}}$ is the number of greedy updates on the current set X and we choose all parameters $p, T_{\texttt{CG}}, k, s, m, n = O(1)$. We note that Algorithm 2 is expected to perform faster than Algorithm 1 when both $T_{\texttt{GR}}$ and d are large. This is primarily because the size of the current set X increases by $k > 1$ for each greedy iteration. A larger choice of k speeds up the algorithm up to k times, but it might hurt its output quality. We explain more details of key components of the batch algorithm below.

Batch selection. The essence of Algorithm 2 is adding $k > 1$ elements, called batch, simultaneously to the current set with an improved marginal gain. Formally, it starts from the empty set and recursively updates $X \leftarrow X \cup I_{\max}$ for

$$I_{\max} = \operatorname*{argmax}_{I \subseteq \mathcal{Y} \setminus X, |I| = k} \log\det L_{X \cup I}. \tag{4}$$

until no gain is attained. The non-batch greedy procedure (1) corresponds to $k = 1$. Such batch greedy algorithms have been also studied for submodular maximization (Nemhauser et al., 1978; Hausmann et al., 1980) and recently, (Liu et al., 2016) studied their theoretical guarantees showing that they can be better than their non-batch counterparts under some conditions. The main drawback of the standard batch greedy algorithms is that finding the optimal batch of size k requires computing too many marginal gains of $\binom{|\mathcal{Y} \setminus X|}{k}$ subsets. To address the issue, we sample

[2] For $Z \in \mathbb{R}^{d \times k}$, $\texttt{Mat}(Z) \in \mathbb{R}^{d \times d}$ is defined whose the last k columns and rows are equal to Z and $Z^\top$, respectively, and other entries set to 0.

$s \ll \binom{|\mathcal{Y}\setminus X|}{k}$ bunches of batch subsets randomly and compute approximate batch marginal gains using them. (Mirzasoleiman et al., 2015) first propose an uniformly random sampling to the standard non-batch greedy algorithm. The authors show that it guarantees $(1 - 1/e - O(e^{-s}))$ approximation ratio in expectation and report that it performs well in many applications. In our experiments, we choose $s = 50$ batch samples.

High-order approximation of log-determinant. Recall that for Algorithm 1, we suggest using the CG algorithm under the Schur complement for computing

$$\log\det \overline{L}_X^{(j)} - \log\det L_X. \tag{5}$$

One can apply the same strategy for Algorithm 2, which requires running the CG algorithm k times for (5). Instead, we suggest running LDAS (using polynomial/high-order approximations of the scalar log function) only once, i.e., the line 9, which is much faster if k is large. We remind that the asymptotic complexities of CG and LDAS are comparable.

4.2. Sharing Randomness in Trace Estimators

To improve the approximation quality of Algorithm 2, we further suggest running LDAS using the same random vectors $\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(m)}$ across $j \in [p]$. This is because we are interested in relative values $\log\det \overline{L}_X^{(j)}$ for $j \in [p]$ instead of their absolute ones.

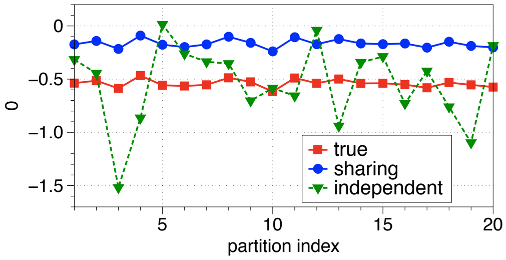


Figure 1. Log-determinant estimation qualities of LDAS for sharing and independent random vectors.

Our intuition is that different random vectors have different bias, which hurt the comparison task. Figure 1 demonstrates an experiment on the estimation of $\log\det \overline{L}_X^{(j)}$ when random vectors are shared and independent, respectively. This implies that sharing random vectors might be worse for estimating the absolute values of log-determinants, but better for comparing them.

We also formally justify the idea of sharing random vectors as stated in the follows theorem whose proof is given in the supplementary material.

Theorem 2. *Suppose* A, B *are positive definite matrices whose eigenvalues are in* $[\delta, 1-\delta]$ *for* $\delta > 0$. *Let* Γ_A, Γ_B *be the estimations of* $\log\det A$, $\log\det B$ *by* LDAS *using the same random vectors* $\mathbf{v}^{(1)}, \dots, \mathbf{v}^{(m)}$ *for both. Then, it holds that*

$$\mathbf{Var}\left[\Gamma_A - \Gamma_B\right] \le \frac{32M^2\rho^2\left(\rho+1\right)^2}{m\left(\rho-1\right)^6\left(1-2\delta\right)^2}\|A-B\|_F^2$$

where $M = 5\log\left(2/\delta\right)$ *and* $\rho = 1 + \frac{2}{\sqrt{2/\delta - 1} - 1}$.

Without sharing random vectors, the variance should grow linearly with respect to $\|A\|_F^2 + \|B\|_F^2$. In our case, matrices A and B correspond to some of $\overline{L}_X^{(j)}$, and $\|A - B\|_F^2$ is significantly smaller than $\|A\|_F^2 + \|B\|_F^2$. We believe that our idea of sharing randomness might be of broader interest in many applications of LDAS or its variants, requiring multiple log-determinant computations.

5. Experimental Results

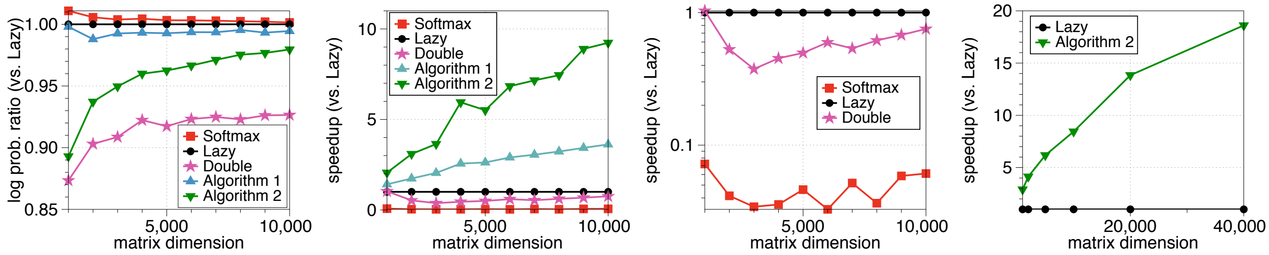
In this section, we evaluate our proposed algorithms for the MAP inference on synthetic and real-world DPP instances. [3]

Setups. The experiments are performed using a machine with a hexa-core Intel CPU (Core i7-5930K, 3.5 GHz) and 32 GB RAM. We compare our algorithms with following competitors: the lazy greedy algorithm (LAZY) (Minoux, 1978), double greedy algorithm (DOUBLE) (Buchbinder et al., 2015) and softmax extension (SOFTMAX) (Gillenwater et al., 2012). In all our experiments, LAZY is significantly faster than the naïve greedy algorithms described in Section 2.3, while they produce the same outputs. Hence, we use LAZY as the baseline of evaluation.

Unless stated otherwise, we choose parameters of $p = 5$, $k = 10$, $s = 50$, $m = 20$ and $n = 15$, regardless matrix dimension, for our algorithms. We also run CG until it achieves convergence error less than 10^{-10} and typically $T_{\texttt{CG}} \le 30$.

Additional tricks for boosting accuracy. For boosting approximation qualities of our algorithms, we use the simple trick in our experiments: recompute top ℓ marginal gains exactly (using CG) where they are selected based on estimated marginal gains, i.e., Δ_i for Algorithm 1 and Δ_i^{Batch} for Algorithm 2. Then, our algorithms choose the best element among ℓ candidates, based on their exact marginal gains. Since we choose small $\ell = 20$ in our experiments, this additional process increases the running times of our algorithms marginally, but makes them more accurate. In fact, the trick is inspired from (Minoux, 1978) where the authors also recompute the exact marginal gains of few elements. In addition, for boosting further approximation

[3] The codes are available in https://github.com/insuhan/fastdppmap.



(a)

(b)

Figure 2. Plot of (a) log-probability ratio and (b) speedup for SOFTMAX, DOUBLE, Algorithm 1 and Algorithm 2 compared to LAZY. Algorithm 1 is about 3 times faster the lazy greedy algorithm (LAZY) while loosing only 0.2% accuracy at $d = 10,000$. Algorithm 2 has 2% loss on accuracy but 9 times faster than LAZY at $d = 10,000$. If dimension is $d = 40,000$, it runs 19 times faster.

qualities of Algorithm 2, we also run Algorithm 1 in parallel and choose the largest one among $\{\Delta_i, \Delta_i^{\text{Batch}}\}$ given the current set. Hence, at most iterations, the batch with the maximal Δ_i^{Batch} is chosen and increases the current set size by k (i.e., making speed-up) as like Algorithm 2, and the non-batch with the maximal Δ_i is chosen at very last iterations, which fine-tunes the solution quality. We still call the synthesized algorithm by Algorithm 2 in this section.

Performance metrics. For the performance measure on approximation qualities of algorithms, we use the following ratio of log-probabilities:

$$\log\det L_X / \log\det L_{X_{\text{LAZY}}}.$$

where X and X_{LAZY} are the outputs of an algorithm and LAZY, respectively. Namely, we compare outputs of algorithms with that of LAZY since the exact optimum is hard to compute. Similarly, we report the running time speedup of each algorithm over LAZY.

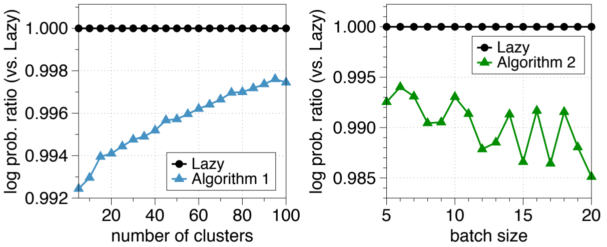
5.1. Synthetic Dataset

In this section, we use synthetic DPP datasets generated as follows. As (Kulesza & Taskar, 2011; Kulesza et al., 2012) proposed, a kernel matrix L for DPP can be reparameterized as

$$L_{i,j} = q_i \phi_i^\top \phi_j q_j,$$

where $q_i \in \mathbb{R}^+$ is considered as the quality of item i and $\phi_i \in \mathbb{R}^d$ is the normalized feature vector of item i so that $\phi_i^\top \phi_j$ measures the similarity between i and j. We use $q_i = \exp(\beta_1 x_i + \beta_2)$ for the quality measurement $x_i \in \mathbb{R}$ and choose $\beta_1 = 0.01, \beta_2 = 0.2$. We choose each entry of ϕ_i and x_i drawn from the normal distribution $\mathcal{N}(0, 1)$ for all $i \in [d]$, and then normalize ϕ_i so that $\|\phi_i\|_2 = 1$.

We first show how much the number of clusters p and the batch size k are sensitive for Algorithm 1 and Algorithm 2, respectively. Figure 3(a) shows the accuracy of Algorithm 1 with different numbers of clusters. It indeed confirms that a larger cluster improves its accuracy since it makes first-order approximations tighter. Figure 3(b) shows the performance trend of Algorithm 2 as the batch size k increases, which shows that a larger batch might hurt its accuracy. Based on these experiments, we choose $p = 5, k = 10$ in order to target 0.01 approximation ratio loss compared to LAZY.



(a)

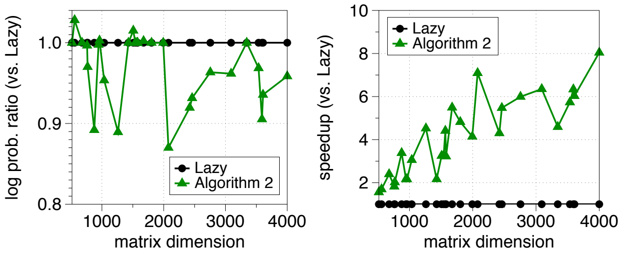
(b)

Figure 3. Log-probability ratios compared to LAZY: (a) Algorithm 1 changing the number of clusters p and (b) Algorithm 2 varying the batch size k. These experiments are done under $d = 1,000$.

We generate synthetic kernel matrices with varying dimension d up to 40,000, and the performances of tested algorithms are reported in Figure 2(a). One can observe that LAZY seems to be near-optimal, where only SOFTMAX often provides marginally larger log-probabilities than LAZY under small dimensions. Interestingly, we found that DOUBLE has the strong theoretical guarantee for general submodular maximization (Buchbinder et al., 2015), but its practical performance for DPP is worst among evaluating algorithms. Moverover, it is slightly slower than LAZY. In summary, one can conclude that our algorithms can be at orders of magnitude faster than LAZY, DOUBLE and SOFTMAX, while loosing 0.01-approximation ratio. For example, Algorithm 2 is 19 times faster than LAZY for $d = 40,000$, and the gap should increase for larger dimension d.

5.2. Real Dataset

We use real-world datasets of the following two tasks of matched and video summarizations.



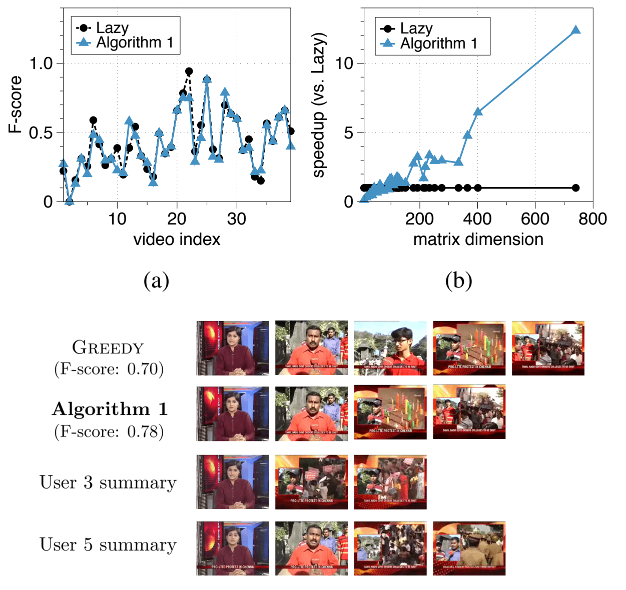
(a) (b)

Figure 4. Plot of log-probability ratio and speedup (log-scale) of Algorithm 2, compared to LAZY, for matched summarization under 2016 Republican presidential primaries.

Matched summarization. We evaluate our proposed algorithms for matched summarization that is first proposed by (Gillenwater et al., 2012). This task gives useful information for comparing the texts addressed at different times by the same speaker. Suppose we have two different documents and each one consists of several statements. The goal is to apply DPP for finding statement pairs that are similar to each other, while they summarize (i.e., diverse) well the two documents. We use transcripts of debates in 2016 US Republican party presidential primaries speeched by following 8 participates: Bush, Carson, Christie, Kasich, Paul, Trump, Cruz and Rubio.[4]

We follow similar pre-processing steps of (Gillenwater et al., 2012). First, every sentence is parsed and only nouns except the stopwords are extracted via NLTK (Bird, 2006). Then, we remove the 'rare' words occurring less than 10% of the whole debates, and then ignore each statement which contains more 'rare' words than 'frequent' ones in it. This gives us a dataset containing $3,406$ distinct 'frequent' words and $1,157$ statements. For each statement pair (i,j), feature vector $\phi_{(i,j)} = w_i + w_j \in \mathbb{R}^{3406}$ where w_i is generated as a frequency of words in the statement i. Then, we normalize $\phi_{(i,j)}$. The match quality $x_{(i,j)}$ is measured as the cosine similarity between two statements i and j, i.e., $x_{(i,j)} = w_i^\top w_j$, and we remove statement pairs (i,j) such that its match quailty $x_{(i,j)}$ is smaller than 15% of the maximum one. Finally, by choosing $q_{(i,j)} = \exp\left(0.01 \cdot x_{(i,j)}\right)$, we obtain $\binom{8}{2} = 28$ kernel matrices of dimension d from 516 to $4,000$.

Figure 4 reports log-probability ratios and speedups of Algorithm 2 under the 28 kernels. We observe that Algorithm 2 looses 0.03-approximation ratio on average, compared to LAZY, under the real-world kernels. Interestingly, SOFTMAX runs much slower than even LAZY, while our algorithm runs faster than LAZY for large dimension, e.g., 8 times faster for $d = 4,000$ corresponding to transcripts of Bush and Rubio.



(a) (b)

(c)

Figure 5. Plot of (a) F-scores for Algorithm 1 compared to LAZY and (b) speedup of both algorithms. (c) shows the summaries of YouTube video of index 99. Images in the first row are summaries produced by LAZY and the second row images illustrate those produced by Algorithm 1. The bottom 2 rows reflect 'real' user summaries.

Video summarization. We evaluate our proposed algorithms video summarization. We use 39 videos from a Youtube dataset (De Avila et al., 2011), and the trained DPP kernels from (Gong et al., 2014). Under the kernels, we found that the numbers of selected elements from algorithms are typically small (less than 10), and hence we use Algorithm 1 instead of its batch version Algorithm 2. For performance evaluation, we use an F-score based on five sets of user summaries where it measures the quality across two summaries.

Figure 5(a) illustrates F-score for LAZY and Algorithm 1 and Figure 5(b) reports its speedup. Our algorithm achieves over 13 times speedup in this case, while it produces F-scores that are very similar to those of LAZY. For some video, it achieves even better F-score, as illustrated in 5(c).

6. Conclusion

We have presented fast algorithms for the MAP inference task of large-scale DPPs. Our main idea is to amortize common determinant computations via linear algebraic techniques and recent log-determinant approximation methods. Although we primarily focus on a special matrix optimization, we expect that several ideas developed in this paper would be useful for other related matrix computational problems, in particular, involving multiple determinant computations.

[4] Details of the primaries are provided in http://www.presidency.ucsb.edu/debates.php.

Acknowledgements

This work was supported in part by the ICT R&D Program of MSIP/IITP, Korea, under [2016-0-00563, Research on Adaptive Machine Learning Technology Development for Intelligent Autonomous Digital Companion].

References

Avron, Haim and Toledo, Sivan. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)*, 58(2): 8, 2011.

Bird, Steven. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pp. 69–72. Association for Computational Linguistics, 2006.

Boutsidis, Christos, Drineas, Petros, Kambadur, Prabhanjan, and Zouzias, Anastasios. A randomized algorithm for approximating the log determinant of a symmetric positive definite matrix. *arXiv preprint arXiv:1503.00374*, 2015.

Buchbinder, Niv, Feldman, Moran, Seffi, Joseph, and Schwartz, Roy. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM Journal on Computing*, 44(5):1384–1402, 2015.

Daley, Daryl J and Vere-Jones, David. *An introduction to the theory of point processes: volume II: general theory and structure*. Springer Science & Business Media, 2007.

De Avila, Sandra Eliza Fontes, Lopes, Ana Paula Brandão, da Luz, Antonio, and de Albuquerque Araújo, Arnaldo. Vsumm: A mechanism designed to produce static video summaries and a novel evaluation method. *Pattern Recognition Letters*, 32(1):56–68, 2011.

Feige, Uriel, Mirrokni, Vahab S, and Vondrak, Jan. Maximizing non-monotone submodular functions. *SIAM Journal on Computing*, 40(4):1133–1153, 2011.

Gillenwater, Jennifer, Kulesza, Alex, and Taskar, Ben. Near-optimal map inference for determinantal point processes. In *Advances in Neural Information Processing Systems*, pp. 2735–2743, 2012.

Gong, Boqing, Chao, Wei-Lun, Grauman, Kristen, and Sha, Fei. Diverse sequential subset selection for supervised video summarization. In *Advances in Neural Information Processing Systems*, pp. 2069–2077, 2014.

Greenbaum, Anne. *Iterative methods for solving linear systems*. SIAM, 1997.

Han, Insu, Malioutov, Dmitry, and Shin, Jinwoo. Large-scale log-determinant computation through stochastic chebyshev expansions. In *ICML*, pp. 908–917, 2015.

Hausmann, Dirk, Korte, Bernhard, and Jenkyns, TA. Worst case analysis of greedy type algorithms for independence systems. In *Combinatorial Optimization*, pp. 120–131. Springer, 1980.

Hutchinson, Michael F. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450, 1990.

Johansson, Kurt. Course 1 random matrices and determinantal processes. *Les Houches*, 83:1–56, 2006.

Jordan, Michael Irwin. *Learning in graphical models*, volume 89. Springer Science & Business Media, 1998.

Kang, Byungkon. Fast determinantal point process sampling with application to clustering. In *Advances in Neural Information Processing Systems*, pp. 2319–2327, 2013.

Kathuria, Tarun and Deshpande, Amit. On sampling and greedy map inference of constrained determinantal point processes. *arXiv preprint arXiv:1607.01551*, 2016.

Krause, Andreas, Singh, Ajit, and Guestrin, Carlos. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(Feb):235–284, 2008.

Kulesza, Alex and Taskar, Ben. Learning determinantal point processes. In *In Proceedings of UAI*. Citeseer, 2011.

Kulesza, Alex, Taskar, Ben, et al. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 5(2–3):123–286, 2012.

Kumar, Ravi, Moseley, Benjamin, Vassilvitskii, Sergei, and Vattani, Andrea. Fast greedy algorithms in mapreduce and streaming. *ACM Transactions on Parallel Computing*, 2(3):14, 2015.

Li, Chengtao, Jegelka, Stefanie, and Sra, Suvrit. Efficient sampling for k-determinantal point processes. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pp. 1328–1337, 2016a.

Li, Chengtao, Sra, Suvrit, and Jegelka, Stefanie. Gaussian quadrature for matrix inverse forms with applications. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1766–1775, 2016b.

Liu, Yajing, Zhang, Zhenliang, Chong, Edwin KP, and Pezeshki, Ali. Performance bounds for the k-batch greedy strategy in optimization problems with curvature. In *American Control Conference (ACC), 2016*, pp. 7177–7182. IEEE, 2016.

Macchi, Odile. The coincidence approach to stochastic point processes. *Advances in Applied Probability*, 7(01): 83–122, 1975.

Minoux, Michel. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques*, pp. 234–243. Springer, 1978.

Mirzasoleiman, Baharan, Badanidiyuru, Ashwinkumar, Karbasi, Amin, Vondrák, Jan, and Krause, Andreas. Lazier than lazy greedy. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Nemhauser, George L, Wolsey, Laurence A, and Fisher, Marshall L. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming*, 14(1):265–294, 1978.

Ouellette, Diane Valerie. Schur complements and statistics. *Linear Algebra and its Applications*, 36:187–295, 1981.

Pan, Xinghao, Jegelka, Stefanie, Gonzalez, Joseph E, Bradley, Joseph K, and Jordan, Michael I. Parallel double greedy submodular maximization. In *Advances in Neural Information Processing Systems*, pp. 118–126, 2014.

Peng, Wei and Wang, Hongxia. Large-scale log-determinant computation via weighted l_2 polynomial approximation with prior distribution of eigenvalues. In *International Conference on High Performance Computing and Applications*, pp. 120–125. Springer, 2015.

Saad, Yousef. *Iterative methods for sparse linear systems*. SIAM, 2003.

Sharma, Dravyansh, Kapoor, Ashish, and Deshpande, Amit. On greedy maximization of entropy. In *ICML*, pp. 1330–1338, 2015.

Yao, Jin-ge, Fan, Feifan, Zhao, Wayne Xin, Wan, Xiaojun, Chang, Edward, and Xiao, Jianguo. Tweet timeline generation with determinantal point processes. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 3080–3086. AAAI Press, 2016.

Zhang, Martin J and Ou, Zhijian. Block-wise map inference for determinantal point processes with application to change-point detection. In *Statistical Signal Processing Workshop (SSP), 2016 IEEE*, pp. 1–5. IEEE, 2016.