# MAP Inference for Customized Determinantal Point Processes via Maximum Inner Product Search

**Insu Han**
Korea Advanced Institute of Science and Technology[1]
Daejeon, South Korea

**Jennifer Gillenwater**
Google Research
New York City, USA

## Abstract

Determinantal point processes (DPPs) are a good fit for modeling diversity in many machine learning applications. For instance, in recommender systems, one might have a basic DPP defined by item features, and a customized version of this DPP for each user with features re-weighted according to user preferences. While such models perform well, they are typically applied only to relatively small datasets, because existing maximum a posteriori (MAP) approximation algorithms are expensive. In this work, we propose a new MAP algorithm: we show that, by performing a one-time preprocessing step on a basic DPP, it is possible to run an approximate version of the standard greedy MAP approximation algorithm on any customized version of the DPP in time sublinear in the number of items. Our key observation is that the core computation can be written as a maximum inner product search (MIPS), which allows us to accelerate inference via approximate MIPS structures, e.g., trees or hash tables. We provide a theoretical analysis of the algorithm's approximation quality, as well as empirical results on real-world datasets demonstrating that it is often orders of magnitude faster while sacrificing little accuracy.

## 1 Introduction

Diversification is essential in many machine learning applications, and determinantal point processes (DPPs) have become a popular means to this end. First introduced to model repulsion in quantum physics (Macchi, 1975), DPPs have since been applied to a variety of practical machine learning applications. These include: data summarization (Gillenwater et al., 2012; Chao et al., 2015; Sharghi et al., 2018), recommender systems (Chen et al., 2017; Wilhelm et al., 2018; Gartrell et al., 2019), neural network compression (Mariet and Sra, 2016), kernel approximation (Li et al., 2016a), multi-modal output generation (Elfeki et al., 2019), and batch selection, both for stochastic optimization (Zhang et al., 2017) and for active learning (Bıyık et al., 2019). We believe that the ideas from this paper can be used for many of these, but in this work we use recommender systems as the motivating application.

Consider the problem of movie recommendation, where we might have feature vectors describing $N$ movies, $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N$, and a weight vector $\boldsymbol{w}$ describing the types of movies that a user prefers. Finding the 10 movies whose feature vectors $\boldsymbol{b}_i$ have largest inner product with $\boldsymbol{w}$ would be one way of selecting movies that are very relevant to a user's interests (Koren et al., 2009). However, these movies would likely not be very diverse, and there is a large body of work showing that diversifying items improves user engagement (McNee et al., 2006; Zhang and Hurley, 2008; Yu et al., 2009; Vargas et al., 2014). DPPs formalize one way of trading off the relevance of selected items with their diversity, and this particular trade-off has been shown to be a good fit for real-world recommendation systems (Chen et al., 2017; Wilhelm et al., 2018).

Despite their successes, DPPs have mostly been limited to use on relatively small datasets, due to the expense of core inference algorithms. In particular, consider the problem of selecting from a pool of $N$ items a set of $K$ items to recommend to a user. Ideally, we would want to select the set that maximizes the DPP score, also known as the *maximum a posteriori* (MAP) inference task, but this is NP-hard (Ko et al., 1995; Kulesza et al., 2012). In practice, a set is typically selected by running the standard greedy algorithm from the submodular

[1] Work done during an internship at Google Research.

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108. Copyright 2020 by the author(s).

maximization literature (Nemhauser et al., 1978). This algorithm starts from an empty set and runs for $K$ iterations, each iteration adding the item that most increases the DPP score. Its runtime is $\mathcal{O}(KDN)$, if each item is represented by a length-$D$ feature vector (Chen et al., 2018). This runtime makes it feasible for use only at the end of recommendation pipelines, when the candidate pool of items has been winnowed down to a realtively small value, such as $N = 5000$.

In this work, we propose a variation on the greedy algorithm that allows it to scale to larger $N$. In our experiments, the proposed algorithm runs several orders of magnitude faster than the standard greedy algorithm on real-world recommendation tasks, while only sacrificing marginal objective value. This makes it possible to use DPPs earlier in recommendation pipelines, when the candidate pool is large (or on larger datasets for other applications). Ensuring diversity at such early pipeline stages has long been an issue (Cheng et al., 2016). Improving it would also likely make downstream DPPs more effective, as they cannot do a good job of balancing relevance with diversity if their entire input pool is homogeneous.

**Related work.** Increasing the dataset size on which DPP *sampling* is feasible has been a active topic of research. MCMC algorithms have been developed that are capable of sampling in time linear in $N$ (Anari et al., 2016; Li et al., 2016b). Other recent works showed that, following a one-time preprocessing phase, every subsequent sample can be drawn relatively quickly — Derezinski et al. (2019)'s algorithm requires $\widetilde{\mathcal{O}}(K^6 + DK^4)$ time, and Gillenwater et al. (2019)'s requires $\mathcal{O}(K^2D^2 \log N + D^3)$. However, as shown empirically in Section 4, these sort of algorithms have two disadvantages: 1) they tend to be slower than the greedy MAP approximation algorithm proposed in this work and, 2) samples tend to be inferior to MAP approximations in terms of DPP score.

# 2 Preliminaries

We use $\langle \cdot, \cdot \rangle$ to denote the inner product. For vectors, this is $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \sum_i u_i v_i$, and for matrices it is $\langle L, M \rangle = \sum_i \sum_j L_{ij} M_{ij}$. Given a matrix $L \in \mathbb{R}^{N \times N}$ and $A, B \subseteq [N] := \{1, 2, \ldots, N\}$, we use $L_{AB} \in \mathbb{R}^{|A| \times |B|}$ to indicate the submatrix formed by taking rows $A$ and columns $B$ from $L$. We also use $L_A$ as shorthand for the square matrix $L_{AA}$, and colon to indicate all rows or columns: $L_{A:} \in \mathbb{R}^{|A| \times N}$.

## 2.1 Determinantal point processes (DPPs)

A DPP on a set of $N$ items defines a probability distribution over subsets $Y \subseteq [N]$. It is parameterized by a positive semi-definite (PSD) matrix $L \in \mathbb{R}^{N \times N}$ such that the probability of subset $Y$ is proportional to the determinant of the submatrix $L_Y$: $\mathcal{P}_L(Y) \propto \det(L_Y)$.

Any PSD matrix can be written as the Gramian matrix for some set of vectors, so throughout this work we use the decomposition $L := B^\top B$, where $B := [\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N] \in \mathbb{R}^{D \times N}$. One can think of column $\boldsymbol{b}_i$ as a feature vector describing item $i$. Our algorithm is designed to work well for modest values of $D$, which often occur in practice (e.g., our experiments on real datasets have values of $D \leq 1{,}152$). In cases where the natural $D$ is larger (e.g., natural language processing applications), random projections can often be used to reduce $D$ without significantly changing the DPP (Gillenwater et al., 2012).

**Geometric interpretation.** The determinant of $L_Y$ equals the squared volume of the parallelepiped spanned by $\{\boldsymbol{b}_i\}_{i \in Y}$ (see Margalit and Rabinoff, 2017, Section 4.3). Intuitively, to get large volume, the feature vectors must point in substantially different directions (e.g., be diverse). If the feature vectors are also scaled so that their magnitude is proportional to their relevance, then larger volume also correlates with more relevant items. Thus, for many applications, by appropriately scaling item feature vectors, we can easily construct a DPP that places high probability only on sets that contain relevant, but diverse, items.

## 2.2 MAP inference

In practice, we often want to find the set that has the highest probability under a DPP. As mentioned earlier, this problem is known as *maximum a posteriori* (MAP) inference. We are usually concerned with the constrained version of the problem, where we must select exactly $K$ items (e.g., because there are a fixed number of slots for recommendations in a user interface). Formally, the problem is to find:

$$Y^* = \underset{Y \subseteq [N], |Y| = K}{\operatorname{argmax}} \det(L_Y). \tag{1}$$

This is known to be NP-hard (Ko et al., 1995; Kulesza et al., 2012). However, since log-determinant is a submodular function (Kelmans and Kimelfeld, 1983), in practice one frequently applies the standard greedy algorithm for submodular maximization (Nemhauser et al., 1978) as a heuristic. The algorithm begins with $Y = \emptyset$ and iteratively adds an item that maximizes the marginal gain:

$$i^* = \underset{i \in [N]}{\operatorname{argmax}} \det(L_{Y \cup \{i\}}), \tag{2}$$

until $|Y| = K$. This algorithm also has some formal approximation guarantees when the minimum eigenvalue of $L$ is greater than 1 (Sharma et al., 2015; Bian et al., 2017). With a naïve implementation, the runtime of the greedy algorithm is $\mathcal{O}(NK^4)$, since computing the determinant of a $K \times K$ matrix requires $\mathcal{O}(K^3)$ operations for general matrices. With a more nuanced implementation, it requires just $\mathcal{O}(KDN)$ time (Chen et al., 2018). Note that in what follows we will always assume that $K \leq D$. (It does not make sense to try to select a set $Y$ of size $K > D$ using $\det(L_Y)$ as the set score, since $\det(L_Y) = 0$ for all $Y$ of size $> D$.) The idea behind the Chen et al. (2018) algorithm is that, in the first iteration of the greedy algorithm, the score of item $i$ is $L_{ii}$, and it is possible to simply update these diagonal values to condition on selected items. Thus, on iteration $j$, $O(D)$ work is done to update each of the $N - j$ scores of the remaining items.

### 2.3 Customization

The algorithm proposed by Chen et al. (2018) is the fastest known for the setting where there is a single, fixed DPP. But often what we want in practice is to run MAP inference for a large number of related DPPs. For instance, consider the recommendation setting alluded to earlier, where every item $i$ is described by a fixed feature vector $\boldsymbol{b}_i$, but each user has their own preferences vector $\boldsymbol{w}$. Let $W$ be a $D \times D$ diagonal matrix with $\boldsymbol{w}$ on its diagonal. If we define customized item features as follows:

$$\widehat{\boldsymbol{b}}_i := W\boldsymbol{b}_i, \quad \widehat{B} = [\widehat{\boldsymbol{b}}_1, \ldots, \widehat{\boldsymbol{b}}_N], \tag{3}$$

then the corresponding customized DPP, $\widehat{L} = \widehat{B}^\top \widehat{B}$, places high probability on sets of items that are diverse, but also relevant to the user's interests. (This is the same style of customization suggested by Gillenwater et al. (2019).) Finding the highest-probability size-$K$ subset (solving the MAP problem) for the customized DPP corresponds to generating a good recommendation set for the user.

When a user visits a website, we need to quickly generate their recommendation set. If there are a large number $N$ of candidate items, then running Chen et al. (2018)'s greedy algorithm in $\mathcal{O}(KDN)$ time will be too slow. Instead, what we propose in this work is to take advantage of the shared aspects of the users' DPPs (the fixed item feature vectors), to do some offline, one-time preprocessing that benefits all users. In particular, we will build a structure that allows us to do fast, approximate maximum inner product search (MIPS) over the $\boldsymbol{b}_i$. With this structure, we can run an approximate version of the greedy algorithm for any *customized* DPP, $\widehat{L}$, in time sublinear in $N$.

We pause here to note some generalizations. First of all, the "user preferences vector" could also be a vector representing a user search query (e.g., an embedding of the text of a search query by a neural net). The customized DPP $\widehat{L}$ in this case would place high probability on sets that are diverse, but relevant to the query. Secondly, $W$ could be any non-diagonal $D \times D$ matrix, allowing preferences to be expressed as linear combinations of the original features from $B$. In what follows we will assume $W$ could be non-diagonal, and will refer to it as the "customization matrix".

## 3 Algorithm

We start by showing that it is possible to write the expression for marginal gain from Equation 2 as a $D \times D$ matrix inner product where one of the matrices is independent of the set selected thus far, $Y$, and the customization matrix, $W$. By applying Schur's determinant identity, we can simplify the expression for marginal gain as follows:

$$\begin{aligned} i^* &= \operatorname*{argmax}_{i \in [N]} \det(\widehat{L}_{Y \cup \{i\}}) && (4) \\ &= \operatorname*{argmax}_{i \in [N]} \det(\widehat{L}_Y) \det(\widehat{L}_{ii} - \widehat{L}_{iY}(\widehat{L}_Y)^{-1}\widehat{L}_{Yi}) && (5) \\ &= \operatorname*{argmax}_{i \in [N]} \widehat{L}_{ii} - \widehat{L}_{iY}(\widehat{L}_Y)^{-1}\widehat{L}_{Yi}. && (6) \end{aligned}$$

Recalling that $\widehat{L} = \widehat{B}^\top \widehat{B}$, we can rewrite this as the product of $\widehat{\boldsymbol{b}}_i$ with a $D \times D$ matrix:

$$\begin{aligned} &\widehat{L}_{ii} - \widehat{L}_{iY}(\widehat{L}_Y)^{-1}\widehat{L}_{Yi} && (7) \\ &\quad = \widehat{\boldsymbol{b}}_i^\top \widehat{\boldsymbol{b}}_i - \widehat{\boldsymbol{b}}_i^\top \widehat{B}_{:Y}(\widehat{L}_Y)^{-1}(\widehat{B}_{:Y})^\top \widehat{\boldsymbol{b}}_i && (8) \\ &\quad = \widehat{\boldsymbol{b}}_i^\top (I - \widehat{B}_{:Y}(\widehat{L}_Y)^{-1}(\widehat{B}_{:Y})^\top)\widehat{\boldsymbol{b}}_i. && (9) \end{aligned}$$

Splitting $\widehat{\boldsymbol{b}}_i$ into the customization $W$ and the fixed features $\boldsymbol{b}_i$, we have:

$$\boldsymbol{b}_i^\top W^\top (I - \widehat{B}_{:Y}(\widehat{L}_Y)^{-1}(\widehat{B}_{:Y})^\top) W\boldsymbol{b}_i. \tag{10}$$

Defining $\widehat{C}^{(Y)} := \widehat{B}_{:Y}(\widehat{L}_Y)^{-1}(\widehat{B}_{:Y})^\top$ and re-writing as a matrix inner product:

$$i^* = \operatorname*{argmax}_{i \in [N]} \left\langle W^\top W - W^\top \widehat{C}^{(Y)} W, \boldsymbol{b}_i \boldsymbol{b}_i^\top \right\rangle. \tag{11}$$

The second term in this inner product is constant over all iterations of the greedy algorithm and all customizations since it does not depend on either $Y$ or $W$. This means that we have reduced the greedy algorithm to a classic maximum inner product search (MIPS) problem.

**MIPS.** The classical MIPS problem has the following form: given a fixed set of length-$D$ vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N$, find $\operatorname{argmax}_{i \in [N]} \langle \boldsymbol{b}_i, \boldsymbol{q} \rangle$ for a query vector $\boldsymbol{q} \in \mathbb{R}^D$. Naïvely, this can be done in $\mathcal{O}(DN)$ time by a linear search over all the $\boldsymbol{b}_i$. However, for many applications this is too expensive, and there is a substantial body of work on approximately solving MIPS more efficiently. This work largely focuses on building hash structures (Shrivastava and Li, 2014; Yan et al., 2018) or tree structures (Ram and Gray, 2012; Koenigstein et al., 2012; Bachrach et al., 2014; Auvolat et al., 2015) over the $\boldsymbol{b}_i$. Building such structures is typically expensive, but the amortized cost is small, as the same structure can be used for all queries. There are also variants of these algorithms for the more general *matrix* inner product setting (Ram and Gray, 2012; Koenigstein et al., 2012; Shrivastava and Li, 2014; Yan et al., 2018), which takes the following form: given a fixed set of size $D \times D$ matrices $\{M_1, \dots, M_N\}$, find $\operatorname{argmax}_{i \in [N]} \langle M_i, Q \rangle$ for a query matrix $Q \in \mathbb{R}^{D \times D}$. This is exactly the setting we find ourselves in for DPP MAP inference, with $M_i = \boldsymbol{b}_i \boldsymbol{b}_i^\top$ and query matrix $Q = W^\top W - W^\top \widehat{C}^{(Y)} W$, as in Equation 11.

We can thus use any MIPS structure as a black box to do DPP MAP inference via the greedy algorithm. If we employ a MIPS structure $\mathcal{M}$ that guarantees query time $\mathcal{O}(T_\mathcal{M})$, and if we can compute the query matrix $Q$ in time $\mathcal{O}(T_Q)$, then the greedy algorithm requires $\mathcal{O}(T_\mathcal{M} + T_Q)$ time per iteration. The following proposition shows how to compute $Q$ efficiently.

**Proposition 1.** *Given a feature matrix* $B = [\boldsymbol{b}_1, \dots, \boldsymbol{b}_N] \in \mathbb{R}^{D \times N}$ *and a customization matrix* $W \in \mathbb{R}^{D \times D}$*, denote* $\widehat{\boldsymbol{b}}_i := W \boldsymbol{b}_i$ *for* $i \in [N]$ *and* $\widehat{L} = \widehat{B}^\top \widehat{B}$ *for* $\widehat{B} = [\widehat{\boldsymbol{b}}_1, \dots, \widehat{\boldsymbol{b}}_N]$*. Consider* $Y := \{a_1, \dots, a_{|Y|}\} \subseteq [N]$*. Then:*

$$\widehat{C}^{(Y)} := \widehat{B}_{:Y} (\widehat{L}_Y)^{-1} (\widehat{B}_{:Y})^\top = \sum_{k=1}^{|Y|} \widehat{\boldsymbol{c}}_k \widehat{\boldsymbol{c}}_k^\top, \tag{12}$$

*for* $\widehat{\boldsymbol{c}}_k$ *defined as follows:*

$$\widehat{\boldsymbol{c}}_k := \frac{\widehat{\boldsymbol{d}}_k}{\sqrt{\widehat{\boldsymbol{b}}_{a_k}^\top \widehat{\boldsymbol{d}}_k}}, \quad \widehat{\boldsymbol{d}}_k := \widehat{\boldsymbol{b}}_{a_k} - \left( \sum_{j=1}^{k-1} \widehat{\boldsymbol{c}}_j \widehat{\boldsymbol{c}}_j^\top \right) \widehat{\boldsymbol{b}}_{a_k}\text{[1]}. \tag{13}$$

Proofs for all propositions, lemmas, and theorems not given in the main text of the paper can be found in the supplement. This proposition implies we can compute $\widehat{C}^{(Y)}$ spending only $\mathcal{O}(D^2)$ time each iteration, since on iteration $k$ we just have to add in $\widehat{\boldsymbol{c}}_k \widehat{\boldsymbol{c}}_k^\top$. Since we can also compute $W^\top \widehat{\boldsymbol{c}}_i$ in $\mathcal{O}(D^2)$ time, the overall query matrix $Q$ can similarly be computed in $\mathcal{O}(D^2)$.

Algorithm 1 summarizes the proposed algorithm. If the MIPS structure $\mathcal{M}$'s query time is $\mathcal{O}(T_\mathcal{M})$, then the overall runtime is $\mathcal{O}(KT_\mathcal{M} + KD^2)$. In a subsequent section, we discuss MIPS structures and their query times in more detail.

**Algorithm 1** Customized DPP MAP via MIPS

```
Input: MIPS structure $\mathcal{M}$, query matrix $W$, and number of items to choose $K$
$Y \leftarrow \emptyset$ and $Q = W^\top W$ and $\widehat{C}^{(\emptyset)} = \mathbf{0}$
for $i = 1$ to $K$ do
    $a, \boldsymbol{b}_a \leftarrow \mathcal{M}(Q)$
    $\widehat{\boldsymbol{b}}_a \leftarrow W \boldsymbol{b}_a$
    $\widehat{\boldsymbol{d}}_i \leftarrow \widehat{\boldsymbol{b}}_a - C^{(Y)} \widehat{\boldsymbol{b}}_a$ and $\widehat{\boldsymbol{c}}_i \leftarrow \widehat{\boldsymbol{d}}_i / \sqrt{\widehat{\boldsymbol{b}}_a^\top \widehat{\boldsymbol{d}}_i}$
    $\widehat{C}^{(Y \cup \{a\})} \leftarrow \widehat{C}^{(Y)} + \widehat{\boldsymbol{c}}_i \widehat{\boldsymbol{c}}_i^\top$
    $Y \leftarrow Y \cup \{a\}$
    $Q \leftarrow Q - W^\top \widehat{\boldsymbol{c}}_i \widehat{\boldsymbol{c}}_i^\top W$
Output : $Y$
```

### 3.1 Approximation guarantee

Intuitively, the output of Algorithm 1 should be close to that of the exact greedy algorithm when queries to the MIPS structure $\mathcal{M}$ return near-optimal results. We provide a more precise characterization of this below.

First, we recall the approximation guarantee of the exact greedy algorithm, where an element that maximizes marginal gain is selected each iteration.

**Theorem 1.** *Given* $\widehat{B} = [\widehat{\boldsymbol{b}}_1, \dots, \widehat{\boldsymbol{b}}_N] \in \mathbb{R}^{D \times N}$*, assume that the smallest singular value of* $\widehat{B}_{:S}$ *for any* $S \subseteq [N]$ *such that* $|S| = K$ *is greater than* 1*. Then, it holds that:*

$$\det(\widehat{L}_{\overline{Y}}) \geq (\det(\widehat{L}_{Y^*}))^{(1-1/e)}, \tag{14}$$

*where* $\overline{Y}$ *is the output of the exact greedy algorithm and* $Y^*$ *is the size-*$K$ *set with maximum value:* $Y^* = \max_{|S|=K} \det(\widehat{L}_S)$.

Briefly, Theorem 1 is a result of the fact that log det is a monotone submodular function when the eigenvalues of the matrices that it operates on are large enough.

Next, we state a formal definition of a $(1-\varepsilon)$-MIPS structure on matrices.

**Definition 1.** *Given* $M_1, \dots, M_N \in \mathbb{R}^{D \times D}$ *and a query matrix* $Q \in \mathbb{R}^{D \times D}$ *such that* $\langle M_i, Q \rangle \geq 0 \ \forall i$*, a function* $\mathcal{M} : \mathbb{R}^{D \times D} \rightarrow [N]$ *is a* $(1-\varepsilon)$*-approximate MIPS structure on* $M_1, \dots, M_N$ *if it holds that:*

$$\langle M_a, Q \rangle \geq (1-\varepsilon) \max_{i \in [N]} \langle M_i, Q \rangle, \tag{15}$$

*where* $a \in [N]$ *is the output of* $\mathcal{M}(Q)$.

We are now ready to state an approximation guarantee for our algorithm.

[1] Assume $\sum_{j=1}^{0} \widehat{\boldsymbol{c}}_j \widehat{\boldsymbol{c}}_j$ is the all-zeros matrix, so $\widehat{\boldsymbol{d}}_1 := \widehat{\boldsymbol{b}}_{a_1}$.

**Theorem 2.** *Given $\boldsymbol{b}_1, \dots, \boldsymbol{b}_N \in \mathbb{R}^D$ and a customization matrix $W \in \mathbb{R}^{D \times D}$, denote $\widehat{\boldsymbol{b}}_i := W\boldsymbol{b}_i$ for $i \in [N]$ and $\widehat{L} = \widehat{B}^\top \widehat{B}$ for $\widehat{B} = [\widehat{\boldsymbol{b}}_1, \dots, \widehat{\boldsymbol{b}}_N]$. Assume that the smallest singular value of $\widehat{B}_{:S}$ for any $S \subseteq [N]$ such that $|S| = K$ is greater than 1. Let $\mathcal{M}$ be a $(1-\varepsilon)$-approximate MIPS structure over the matrices $\boldsymbol{b}_i \boldsymbol{b}_i^\top$. Then, it holds that:*

$$\det(\widehat{L}_Y) \geq (1-\varepsilon)^K (\det(\widehat{L}_{Y^*}))^{(1-1/\epsilon)}, \tag{16}$$

*where $Y$ is the output of Algorithm 1, and $Y^*$ is the size-$K$ set with maximum value: $Y^* = \max_{|S|=K} \det(\widehat{L}_S)$.*

This theorem implies that, for small $\varepsilon$ and $K$, Algorithm 1 is a good DPP MAP approximation algorithm.

### 3.2 Hybrid MIPS structures

Building and querying a matrix-MIPS structure over all the outer product matrices $\boldsymbol{b}_i \boldsymbol{b}_i^\top$ can be too expensive for some tasks, especially when the number of features $D$ is in the thousands. For example, a real-world dataset from YouTube[2] that we experiment with has $D = 1{,}152$, which makes $D^2 \approx 1$ million. Intuitively, building such a structure also seems wasteful, since the $D^2$ entries of $\boldsymbol{b}_i \boldsymbol{b}_i^\top$ only depend on the $D$ values of $\boldsymbol{b}_i$.

As an alternative, we propose to create a hybrid MIPS structure: build a standard vector-MIPS structure on the $\boldsymbol{b}_i$, but equip it with a unique query function that is suited to our matrix-MIPS needs. To make this concrete, we delve into the details of how to implement this for one particular MIPS structure, the $k$-means tree. Auvolat et al. (2015) showed $k$-means trees often outperform other tree and hash methods for the MIPS problem, which motivates our use of them.

#### 3.2.1 Hybrid $k$-means tree

A $k$-means tree over vectors $\boldsymbol{b}_1, \dots, \boldsymbol{b}_N \in \mathbb{R}^D$ can be constructed as shown in Algorithm 2. The root node represents the full set of vectors, $\mathcal{I} = [N]$. The algorithm starts from this set, runs $k$-means clustering, and creates a child node for the contents of each cluster. This process is then iterated on the child nodes. Branching halts when a node represents $N_{\text{leaf}}$ or fewer items. Each node also stores a set of centroid vectors: the average feature vectors of its children.

In a typical vector-MIPS setting, there would be an input query vector $\boldsymbol{q} \in \mathbb{R}^D$, and the search would proceed as follows: find the child $j^*$ of the root node whose centroid $\bar{\boldsymbol{b}}_{j^*}$ has maximum inner product with $\boldsymbol{q}$, then repeat this process for the children of that node, until a leaf node is reached. At the leaf node, the set $N_{\text{leaf}} = \mathcal{O}(1)$ is small enough that an exact search can be performed to select whichever of its $\boldsymbol{b}$ vectors has largest inner product with $\boldsymbol{q}$.

**Algorithm 2** Tree construction

```
procedure CONSTRUCT($\mathcal{I}, \{\boldsymbol{b}_i\}_{i\in\mathcal{I}}, k, N_{\text{leaf}}$)
    $\mathcal{T}.\mathtt{I} \leftarrow \mathcal{I}$
    if $|\mathcal{I}| \leq N_{\text{leaf}}$ then
        $\mathcal{T}.\bar{\boldsymbol{b}}_i \leftarrow \boldsymbol{b}_i$ for $i \in \mathcal{I}$
        $\mathcal{T}.\mathtt{T}_j \leftarrow \emptyset$ for $i \in \mathcal{I}$
        return $\mathcal{T}$
    $\{\mathcal{I}_j\}_{j=1}^k \leftarrow$ Partition of $\mathcal{I}$ using $k$-means clustering on $\{\boldsymbol{b}_i\}_{i\in\mathcal{I}}$
    for $j = 1$ to $k$ do
        $\mathcal{T}.\bar{\boldsymbol{b}}_j \leftarrow \frac{1}{|\mathcal{I}_j|}\sum_{i\in\mathcal{I}_j} \boldsymbol{b}_i$
        $\mathcal{T}.\mathtt{T}_j \leftarrow$ CONSTRUCT($\mathcal{I}_j, \{\boldsymbol{b}_i\}_{i\in\mathcal{I}_j}, k, N_{\text{leaf}}$)
    return $\mathcal{T}$
```

**Algorithm 3** Tree search

```
procedure SEARCH($\mathcal{T}, Q$)
    $j^* \leftarrow \text{argmax}_{j\in[k]} \langle (\mathcal{T}.\bar{\boldsymbol{b}}_j)(\mathcal{T}.\bar{\boldsymbol{b}}_j)^\top, Q \rangle$
    if $\mathcal{T}$ is a leaf node then
        return item in $\mathcal{T}.\mathtt{I}$ corresponding to $j^*$
    SEARCH($\mathcal{T}.\mathtt{T}_{j^*}, Q$)
```

In our setting, we will use the same recursive process, but with two main differences: 1) our input will be a query martix $Q$ (see Line 4 of Algorithm 1), and 2) we will chose the child $j^*$ that maximizes $\left\langle \bar{\boldsymbol{b}}_{j^*} \bar{\boldsymbol{b}}_{j^*}^\top, Q \right\rangle$. Algorithm 3 summarizes this search process.

If we instead built a standard matrix-MIPS $k$-means tree over the $\boldsymbol{b}_i \boldsymbol{b}_i^\top$, each node would require an additional factor of $D$ storage space for centroids, since the centroids would be $D \times D$ matrices rather than length-$D$ vectors. Hence, using our hybrid structure can save substantial space. We can also bound the difference between the inner products computed by standard matrix-MIPS search and the inner products computed by our hybrid search from Algorithm 3.

**Theorem 3.** *Given vectors $\boldsymbol{b}_i$ for $i \in \mathcal{C} \subseteq [N]$, define $\bar{\boldsymbol{b}}_{\mathcal{C}} := \frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}} \boldsymbol{b}_i$ and $\overline{B}_{\mathcal{C}} := \frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}} \boldsymbol{b}_i \boldsymbol{b}_i^\top$. Consider $W$ and $\widehat{C}^{(Y)}$ from Proposition 1, and let $Q = W^\top W - W^\top C^{(Y)} W$ as in Algorithm 1. Then, it holds that:*

$$\left| \left\langle \bar{\boldsymbol{b}}_{\mathcal{C}} \bar{\boldsymbol{b}}_{\mathcal{C}}^\top, Q \right\rangle - \left\langle \overline{B}_{\mathcal{C}}, Q \right\rangle \right| \leq \frac{\|W\|_2^2}{2} \left( \max_{i,j\in\mathcal{C}} \|\boldsymbol{b}_i - \boldsymbol{b}_j\|_2^2 \right). \tag{17}$$

This result implies that, as long as the clusters found by $k$-means are relatively compact, we would expect the results of standard matrix-MIPS search and our hybrid-MIPS search to be similar.

Beyond the space savings, the hybrid approach can also

[2] https://research.google.com/youtube8m/

offer substantial query time savings in the case where the customization matrix $W$ is diagonal: $\mathrm{diag}(W) = \boldsymbol{w}$. This is because the inner product maximized in the search process (Line 2 of Algorithm 3) can be computed more efficiently than the naïve $\mathcal{O}(D^2)$. To make this precise, recall the query $Q = W^\top W - W^\top \widehat{C}^{(Y)} W$ from Algorithm 1. Let $\bar{\boldsymbol{b}} \in \mathbb{R}^D$ be a centroid of some cluster in the MIPS $k$-means tree. Then we can re-write the matrix inner product from the tree search as follows:

$$\left\langle \bar{\boldsymbol{b}}\bar{\boldsymbol{b}}^\top, Q \right\rangle = \bar{\boldsymbol{b}}^\top \left( W^2 - W \sum_{j=1}^{|Y|-1} \widehat{\boldsymbol{c}}_j \widehat{\boldsymbol{c}}_j^\top W \right) \bar{\boldsymbol{b}} \qquad (18)$$

$$= \left\| \boldsymbol{w} \odot \bar{\boldsymbol{b}} \right\|_2^2 - \sum_{j=1}^{|Y|-1} \left( \widehat{\boldsymbol{c}}_j^\top (\boldsymbol{w} \odot \bar{\boldsymbol{b}}) \right)^2, \qquad (19)$$

where $\widehat{\boldsymbol{c}}_j$ is the vector defined in Proposition 1 and $\odot$ indicates elementwise product. This expression requires just $\mathcal{O}(|Y|D)$ time to compute.

### 3.3 Runtime

Putting together all of the approximations from the previous sections, we are now ready to state the overall runtime complexity of Algorithm 1. The cost of computing the tree is the cost of running $k$-means clustering $\mathcal{O}(\log N)$ times. The cost of querying is the cost of computing $k$ inner products for each of the $\mathcal{O}(\log N)$ levels of the tree: $\mathcal{O}(kD^2 \log N)$, or $\mathcal{O}(kKD \log N)$ for diagonal customization matrix $W$. The overall MAP algorithm queries $K$ times. It also needs to update $Q$ this many times, which, as discussed in the text following Proposition 1, requires $\mathcal{O}(D^2)$ time per update. Combining these, we have Theorem 4.

**Theorem 4.** *Given $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_N \in \mathbb{R}^D$, assume that a $k$-means cluster tree can be built as in Algorithm 2 with depth $\mathcal{O}(\log N)$. Then the cost for constructing the tree is $\mathcal{O}(kDN \log N)$. Given this tree as input, Algorithm 1 can run in $\mathcal{O}(kKD^2 \log N)$ time, or $\mathcal{O}(kK^2 D \log N + KD^2)$ time when the customization matrix $W$ is diagonal.*

This runtime is sublinear in $N$, and hence more practical than previous versions of the greedy algorithm for the large-scale setting.

## 4 Experiments

We benchmark the performance of various algorithms for recommender systems on both synthetic and real-world datasets. In particular, we evaluate two variants of our algorithms, MatrixMIPS and HybridMIPS, based on $k$-means cluster trees over matrices and vectors, respectively. (HybridMIPS is the algorithm presented in Section 3.2.) We compare them to:

- Greedy: The $O(KDN)$ exact greedy algorithm from Chen et al. (2018).
- StochGreedy: An accelerated greedy algorithm by (Mirzasoleiman et al., 2015). It first samples a few items uniformly at random, then selects the one among these samples that has largest marginal gain. We choose the number of samples so that the runtime is comparable to HybridMIPS.
- FastSamp: The DPP sampling algorithm proposed by Derezinski et al. (2019), with parameters optimized for runtime (see supplement Section B). Note that these parameters do not affect the *quality* of the samples; the algorithm always produces exact samples from the DPP.

For FastSamp we use a Python implementation provided by Derezinski et al. (2019), and for all other algorithms we use our own MATLAB implementations.

Since exact DPP MAP inference is NP-hard, we use Greedy as the baseline instead of the optimal MAP set. To evaluate, we use the ratio of log-probabilities:

$$\log\det(\widehat{L}_Y) / \log\det(\widehat{L}_{\overline{Y}}), \qquad (20)$$

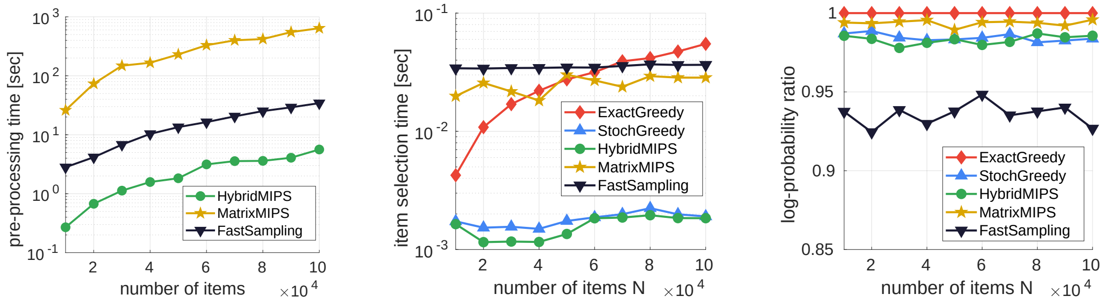
where $\overline{Y}$ is the output of Greedy. This metric was also used for DPP MAP inference by Han et al. (2017) and Chen et al. (2018). We set $K = 10$ for the number of items to select and report the best result by searching $N_{\text{leaf}} \in \{200, 1000\}$ for the size of leaf nodes in our $k$-means trees, and $k = \{50, 100, 500\}$ for the branching factor on these trees. All experiments are preformed using a machine with a hexa-core Intel CPU (Core i7-5930K, 3.5 GHz) and 96 GB of memory.

### 4.1 Synthetic data for non-customized DPP

We generate features by constructing synthetic clusters as follows: 1) sample 10 center points $\bar{\boldsymbol{b}}_1, \ldots, \bar{\boldsymbol{b}}_{10}$ from a standard Gaussian distribution in $D = 128$-dimensional space, 2) sample integers $s_1, \ldots, s_{10}$ from a Poisson distribution with mean 10 and normalize so that $\sum_i s_i = N$, 3) draw $s_i$ points from a Gaussian distribution with mean $\bar{\boldsymbol{b}}_i$ and covariance matrix $0.1I$ where $I$ is a $D$-by-$D$ identity matrix. We vary the number of items $N$ from 10,000 to 100,000. For this setting, the DPP is not customized: $W = I$.

Figure 1 shows the results. Pre-processing for HybridMIPS and MatrixMIPS is construction of the MIPS tree structure, and item selection is running greedy given this structure. Pre-processing for FastSamp involves constructing a Nyström approximation, and item selection is sampling using this.

MatrixMIPS: The slowness of pre-processing and item selection provides evidence that HybridMIPS



(a)

(b)

(c)

Figure 1: Results for synthetic, non-customized DPP: (a) pre-processing time, (b) item selection (sampling or greedy algorithm) time, and (c) log-probability ratio. The sampling time and log-probability ratio of FastSamp are averages of 1,000 independent samples from the DPP.

is more practical in a large-$N$ setting. MatrixMIPS achieves log-probability ratios closest to Greedy, but the ratios of HybridMIPS are also very similar.

FastSamp: Again here, the slowness of pre-processing and item selection provides evidence that HybridMIPS is more practical in a large-$N$ setting. FastSamp's log-probability ratios are also substantially smaller than those of HybridMIPS (which is to be expected, as maximizing log-probability is not the main goal of FastSamp).

StochGreedy: We set the number of random samples to $15 \log N$ for this algorithm so that its runtime is comparable to that of our algorithm. The advantage of HybridMIPS over StochGreedy is not evident from these particular synthetic plots, since the number of clusters of the data is small (i.e., 10). We see a large difference between the two in more complex settings though, as shown in the next section.

**Lazy evaluations**: We briefly note here that lazy evaluations (Minoux, 1978) can further speed up StochGreedy, and our MIPS algorithms as well; since the tree search (Algorithm 3) returns multiple candidate items, it is possible to apply lazy evaluations to them just as to StochGreedy's candidates. For the setting $N = 100{,}000, D = 128, K = 10$, we see that the average runtime for StochGreedy improves from 1.416ms to 1.334ms with lazy evaluations, while the runtime for HybridMIPS improves slightly more, from 1.447ms to 1.163ms.

## 4.2 Real-world data for customized DPP

In this section we consider four datasets: Netflix[3], MovieLens[4], Yahoo!Music[5] and YouTube[6]. Netflix, MovieLens, and Yahoo!Music provide user-item ratings. We randomly split the rating entries of Netflix and MovieLens into 90% for training and 10% for test. For Yahoo!Music, the split is instead the default one provided by the dataset. For these three training sets, we then run non-negative matrix factorization with fixed dimension $D = 128$ (Koren et al., 2009) to extract user and item features. Then, at test time, the item features constitute the $\boldsymbol{b}_i$ and each user's features $\boldsymbol{w}$ form the diagonal of their customization matrix $W$. We select the 1,000 users with the highest number of ratings and report results averaged over these users.

We also compare to collaborative filtering (CollabFilt) for these three datasets. Given a user feature vector $\boldsymbol{w}$, it finds the $K$ items that have maximum inner product with $\boldsymbol{w}$. For a search among $N$ items, its runtime is $\mathcal{O}(ND)$.

To get timing results on a larger dataset, we consider YouTube. This dataset has a total of 3.8 million videos, each described by $D =$ 1,152 features. We uniformly sample these to get a dataset of size $N =$ 1 million. For this dataset there are no public user ratings, so we use a non-customized DPP, i.e., $W = I$.

### 4.2.1 Timing and approximation ratios

Results are reported in Table 1. Note that we do not compare to FastSamp here because it is not applicable to the customized setting.

Greedy: HybridMIPS achieves near-optimal log-probability ratios while running orders of magnitude faster—a ×224 speedup on YouTube.

StochGreedy: We set the parameters so that its runtime is comparable to that of our algorithm, and we see that it is unable to match our algorithm's log-probability ratios in that time.

CollabFilt: Our method typically runs in a compa-

[3] https://www.kaggle.com/netflix-inc/netflix-prize-data

[4] https://grouplens.org/datasets/movielens/

[5] https://webscope.sandbox.yahoo.com/

[6] https://research.google.com/youtube8m/

Table 1: Results for $K = 10$, averaged over 1,000 users.

| Dataset | Number of items $N$ | Feature dimension $D$ | Method | Running time<br>Pre-processing | <br>Item selection | Log-probability ratio |
|---|---|---|---|---|---|---|
| Netflix | 17,770 | 128 | Greedy | – | 7.5 ms | 1 (baseline) |
| | | | StochGreedy | – | 2.3 ms | 0.8795 |
| | | | CollabFilt | – | **1.1** ms | 0.6713 |
| | | | HybridMIPS | 10.23 s | 2.3 ms | **0.9946** |
| MovieLens | 53,389 | 128 | Greedy | – | 31.4 ms | 1 (baseline) |
| | | | StochGreedy | – | 2.4 ms | 0.7574 |
| | | | CollabFilt | – | 3.1 ms | 0.7738 |
| | | | HybridMIPS | 42.4 s | **2.2** ms | **0.9983** |
| Yahoo!Music | 136,736 | 128 | Greedy | – | 69.2 ms | 1 (baseline) |
| | | | StochGreedy | – | **2.5** ms | 0.9317 |
| | | | CollabFilt | – | 7.4 ms | 0.7830 |
| | | | HybridMIPS | 104.8 s | **2.5** ms | **0.9838** |
| YouTube | 1,000,000 | 1,152 | Greedy | – | 3.3 s | 1 (baseline) |
| | | | StochGreedy | – | 15 ms | 0.9391 |
| | | | HybridMIPS | 3,448 s | **14.7** ms | **0.9665** |

rable amount of time, but has a substantially higher log-probability ratio.

### 4.2.2 Relevance and diversity scores

MovieLens and Yahoo!Music also include class labels for items (e.g., movie and music genres). Thus, for these datasets we can provide separate relevance and diversity scores. We denote item $i$'s classes by $\mathcal{C}_i$. If $Z$ is the items in the test set to which a user gave high ratings (not less than 3), then we compute relevance (rel) and diversity (div) for an output $Y$ as follows:

$$\text{rel} := \frac{|(\cup_{i\in Y}\mathcal{C}_i) \cap (\cup_{i\in Z}\mathcal{C}_i)|}{|\cup_{i\in Y}\mathcal{C}_i|}, \quad \text{div} := \frac{|\cup_{i\in Y}\mathcal{C}_i|}{\sum_{i\in Y}|\mathcal{C}_i|}.$$

Note that both scores are in $[0, 1]$. We observe from Table 2 that CollabFilt achieves the best relevance ratio for all cases, but its diversity is relatively small compared to Greedy and HybridMIPS. The diversity ratio of HybridMIPS is similar to Greedy, but recall that HybridMIPS runs significantly faster.

Table 2: Results for $K = 10$, averaged over 1,000 users.

| Dataset | Method | Relevance | Diversity |
|---|---|---|---|
| MovieLens | Greedy | 0.9898 | 0.4021 |
| | HybridMIPS | 0.9901 | **0.4031** |
| | CollabFilt | **0.9930** | 0.3706 |
| Yahoo!Music | Greedy | 0.5211 | **0.2688** |
| | HybridMIPS | 0.5563 | 0.2386 |
| | CollabFilt | **0.6521** | 0.1820 |

### 4.2.3 Example user

For Figure 2, we selected a single user from MovieLens, and made a histogram of classes of items based on the user's high-rated items, HybridMIPS's items, and CollabFilt's items. Observe that HybridMIPS recommends more diverse genres, e.g., HybridMIPS recommends some crime, horror, and mystery movies while CollabFilt does not include any of these.

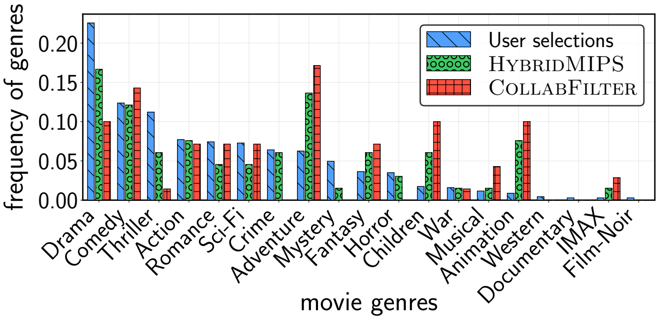


Figure 2: $K = 20$ movie genres from user ratings (blue), HybridMIPS (green), and CollabFilt (red).

## 5 Conclusion

In this work we introduced a method leveraging MIPS for customized DPP MAP inference. Empirical results on recommendation tasks indicate that this technique significantly speeds up runtimes while sacrificing little accuracy. We also believe that the proposed MAP algorithm is likely of interest for other applications. For instance, in batch-mode active learning, it is often the goal to select a diverse set of examples about which the current model is uncertain. If we encode model uncertainty in the customization matrix $W$, then the resulting DPP places high probability on exactly these sets of examples. Running HybridMIPS would hence be a reasonable active learning method.

## References

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

Auvolat, A., Chandar, S., Vincent, P., Larochelle, H., and Bengio, Y. (2015). Clustering is Efficient for Approximate Maximum Inner Product Search. *arXiv preprint arXiv:1507.05910*.

Bachrach, Y., Finkelstein, Y., Gilad-Bachrach, R., Katzir, L., Koenigstein, N., Nice, N., and Paquet, U. (2014). Speeding Up the Xbox Recommender System Using a Euclidean Transformation for Inner-product Spaces. In *Conference on Recommender Systems (RecSys)*.

Bian, A. A., Buhmann, J. M., Krause, A., and Tschiatschek, S. (2017). Guarantees for Greedy Maximization of Non-submodular Functions with Applications. In *International Conference on Machine Learning (ICML)*.

Bıyık, E., Wang, K., Anari, N., and Sadigh, D. (2019). Batch Active Learning Using Determinantal Point Processes. *arXiv:1906.07975*.

Chao, W., Gong, B., Grauman, K., and Sha, F. (2015). Large-Margin Determinantal Point Processes. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Chen, L., Zhang, G., and Zhou, E. (2018). Fast greedy MAP inference for Determinantal Point Process to improve recommendation diversity. In *Neural Information Processing Systems (NIPS)*.

Chen, L., Zhang, G., and Zhou, H. (2017). Improving the Diversity of Top-N Recommendation via Determinantal Point Process. In *Large Scale Recommendation Systems Workshop*.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., and Shah, H. (2016). Wide & Deep Learning for Recommender Systems. In *Workshop on Deep Learning for Recommender Systems*.

Derezinski, M., Calandriello, D., and Valko, M. (2019). Exact sampling of determinantal point processes with sublinear time preprocessing. *arXiv:1905.13476*.

Elfeki, M., Couprie, C., Riviere, M., and Elhoseiny, M. (2019). GDPP: Learning Diverse Generations using Determinantal Point Processes. In *International Conference on Machine Learning (ICML)*.

Gartrell, M., Brunel, V.-E., Dohmatob, E., and Krichene, S. (2019). Learning Nonsymmetric Determinantal Point Processes. *arXiv:1905.12962*.

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

Gillenwater, J., Kulesza, A., and Taskar, B. (2012). Discovering Diverse and Salient Threads in Document Collections. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Han, I., Kambadur, P., Park, K., and Shin, J. (2017). Faster greedy MAP inference for determinantal point processes. In *International Conference on Machine Learning (ICML)*.

Kelmans, A. and Kimelfeld, B. (1983). Multiplicative submodularity of a matrix's principal minor as a function of the set of its rows and some combinatorial applications. *Discrete Mathematics*.

Ko, C.-W., Lee, J., and Queyranne, M. (1995). An Exact Algorithm for Maximum Entropy Sampling. *Operations Research*.

Koenigstein, N., Ram, P., and Shavitt, Y. (2012). Efficient retrieval of recommendations in a matrix factorization framework. In *Conference on Information and Knowledge Management (CIKM)*.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *Computer*.

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

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

Li, C., Jegelka, S., and Sra, S. (2016b). Fast Mixing Markov Chains for Strongly Rayleigh Measures, DPPs, and Constrained Sampling. In *Neural Information Processing Systems (NIPS)*.

Macchi, O. (1975). The Coincidence Approach to Stochastic Point Processes. *Advances in Applied Probability*.

Margalit, D. and Rabinoff, J. (2017). *Interactive Linear Algebra*.

Mariet, Z. and Sra, S. (2016). Diversity Networks: Neural Network Compression Using Determinantal Point Processes. In *International Conference on Learning Representations (ICLR)*.

McNee, S. M., Riedl, J., and Konstan, J. A. (2006). Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems. In *CHI Conference on Human Factors in Computing Systems*.

Minoux, M. (1978). Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization techniques*.

Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A., Vondrák, J., and Krause, A. (2015). Lazier Than Lazy Greedy. In *Conference on Artificial Intelligence (AAAI)*.

Nemhauser, G., Wolsey, L., and Fisher, M. (1978). An Analysis of Approximations for Maximizing Submodular Set Functions I. *Mathematical Programming*, 14(1).

Ram, P. and Gray, A. G. (2012). Maximum inner-product search using cone trees. In *Conference on Knowledge Discovery and Data Mining (KDD)*.

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

Sharma, D., Kapoor, A., and Deshpande, A. (2015). On Greedy Maximization of Entropy. In *International Conference on Machine Learning (ICML)*.

Shrivastava, A. and Li, P. (2014). Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Neural Information Processing Systems (NIPS)*.

Vargas, S., Baltrunas, L., Karatzoglou, A., and Castells, P. (2014). Coverage, Redundancy and Size-awareness in Genre Diversity for Recommender Systems. In *Conference on Recommender Systems (RecSys)*.

Wilhelm, M., Ramanathan, A., Bonomo, A., Jain, S., Chi, E. H., and Gillenwater, J. (2018). Practical Diversified Recommendations on YouTube with Determinantal Point Processes. In *Conference on Information and Knowledge Management (CIKM)*.

Yan, X., Li, J., Dai, X., Chen, H., and Cheng, J. (2018). Norm-ranging lsh for maximum inner product search. In *Neural Information Processing Systems (NIPS)*.

Yu, C., Lakshmanan, L., and Amer-Yahia, S. (2009). It Takes Variety to Make a World: Diversification in Recommender Systems. In *Conference on Extending Database Technology (EDBT)*.

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

Zhang, M. and Hurley, N. (2008). Avoiding Monotony: Improving the Diversity of Recommendation Lists. In *Conference on Recommender Systems (RecSys)*.