A Tree-Based Method for Fast Repeated Sampling of Determinantal Point Processes

Jennifer Gillenwater[1] **Alex Kulesza**[1] **Zelda Mariet**[2,3] **Sergei Vassilvitskii**[1]

Abstract

It is often desirable in recommender systems and other information retrieval applications to provide diverse results, and determinantal point processes (DPPs) have become a popular way to capture the trade-off between the quality of individual results and the diversity of the overall set. However, sampling from a DPP is inherently expensive: if the underlying collection contains N items, then generating each DPP sample requires time linear in N following a one-time preprocessing phase. Additionally, results often need to be personalized to a user, but standard approaches to personalization invalidate the preprocessing, making personalized samples especially expensive. In this work we address both of these shortcomings. First, we propose a new algorithm for generating DPP samples in time logarithmic in N, following a slightly more expensive preprocessing phase. We then extend the algorithm to support arbitrary query-time feature weights, allowing us to generate samples customized to individual users while still retaining logarithmic runtime; experiments show our approach runs over 300 times faster than traditional DPP sampling on collections of 100,000 items for samples of size 10.

1. Introduction

Diverse results are desirable in many applications of information retrieval and recommender systems. For search engines, diversity can be a way to combat query ambiguity; for instance, a user querying "TV" might be researching television sets to buy or simply interested in the local listings. For recommender systems, diversity introduces variety and increases the chance of engagement with at least one item (Smyth & McClave, 2001; Herlocker et al., 2004; Ziegler et al., 2005; Hurley & Zhang, 2011).

Determinantal point processes (DPPs) are probabilistic models of diverse subsets that have often been applied to these kinds of problems (Krause et al., 2008; Zhou et al., 2010; Lin & Bilmes, 2012; Chao et al., 2015; Mariet & Sra, 2016a) and have achieved notable success in practice (Chen et al., 2017; Wilhelm et al., 2018). At their core, DPPs provide a formal trade-off between the quality of individual selected results and the diversity of the overall result set. Unlike many heuristic approaches to capturing diversity, DPPs define full probability distributions over sets of items. This means that they can be repeatedly sampled, for instance to provide the user with a new and diverse set of recommendations each time that they reload a webpage.

However, DPPs suffer from computational limitations that can make them difficult to use in practice. Existing algorithms for sampling from a DPP are at least linear in the number of candidate results, which can be prohibitive in modern applications that may have many thousand or even millions of items. And when results need to be personalized according to the preferences of a user, these algorithms can become even more expensive. In this work we show how to address both of these limitations, providing a method for generating DPP samples that runs in time *logarithmic* in the number of items and allows for *personalization* with minimal additional overhead.

Speed. A DPP defines a probability distribution over possible result sets, giving higher probability to sets that contain high quality, diverse results. To instantiate a specific set of results in a recommendation or information retrieval scenario one often *samples* a set from the DPP. Although DPPs can be impressively efficient given the exponential number of subsets being modeled (Kulesza & Taskar, 2012), sampling is often limited by performance in practice. In particular, the most basic algorithm for sampling a DPP over a ground set of N items (e.g., a database of N news articles or videos) requires $\mathcal{O}(N^3)$ time, dominated by the eigendecomposition of an $N \times N$ matrix. This is rarely practical when N is more than a few hundred.

[1]Google Research NYC [2]Massachussetts Institute of Technology [3]Research performed during an internship at Google. Correspondence to: Jennifer Gillenwater <jengi@google.com>, Alex Kulesza <kulesza@google.com>, Zelda Mariet <zelda@csail.mit.edu>, Sergei Vassilvitskii <sergeiv@google.com>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

The situation is improved somewhat when multiple samples are required, since the eigendecomposition can be reused. In this case, after precomputing the eigendecomposition in $\mathcal{O}(N^3)$ time, each sample requires only $\mathcal{O}(k^2 N)$ additional time, where k is the cardinality of the sampled set (Gillenwater, 2014, Algorithm 2). In most practical applications, k is a small constant; for instance, $k = 5$ slots might be reserved for recommended items in the user interface.

Going further, the eigendecomposition itself can be sped up when the items are represented by D-dimensional feature vectors, $D \ll N$, requiring only $\mathcal{O}(ND^2)$ time (see Section 2 for more details), or when the DPP kernel has a Kronecker structure (Mariet & Sra, 2016b). When the item features are not inherently low-dimensional, random projections can be used to reduce the dimensionality to a small D without significantly altering the resulting DPP (Gillenwater et al., 2012). All told, with existing techniques, repeated exact DPP sampling can often made be practical when N is in the tens of thousands.

However, for many modern applications the ground set may contain hundreds of thousands or even millions of items, making even linear time prohibitively slow. Our first contribution is a new algorithm for repeated DPP sampling whose runtime scales only logarithmically with the number of items in the ground set; after preprocessing, our algorithm requires $\mathcal{O}(k^4 D \log N)$ steps to compute a sample of cardinality k. Empirically, we see speedups of over 300$\times$ on collections of 100,000 items for $k = 10$.

To achieve this, we precompute a special binary tree with N leaves, where each node contains enough information to probabilistically decide whether to proceed left or right to reach the next sampled item. This decision takes $\mathcal{O}(k^3)$ time when k items have already been selected. Given that k is typically very small, this remains an efficient procedure in practice despite the high polynomial dependence on k. Generating the tree requires $\mathcal{O}(ND^2)$ time, making it asymptotically no more expensive than the preprocessing phase of the traditional sampling algorithm.

Approximate sampling. Our second contribution is to exploit the structure of our tree to develop an approximate sampling algorithm. In particular, we show how to efficiently identify nodes whose subtrees define almost-uniform distributions; at such nodes we can short-circuit the computation with only a small loss in accuracy. A simple heuristic for ordering the items in the tree creates many such nodes in practice, leading to significant additional savings.

Personalization. While our tree-based algorithm allows for efficient repeated sampling from the same DPP, in practice we often require personalization. Consider, for instance, a movie recommendation system, where there is no one-size-fits-all quality measure: a horror movie may be the pinnacle of its genre, but should perhaps not be recommended to a family with young children.

A trivial solution is to build and store a unique DPP for each user with personalization baked in; however this is infeasible in applications with millions of users. Instead, we propose a simple model for personalization where each user is associated with arbitrary feature weights that reflect their propensity to engage with different kinds of content. For instance, new parents might have higher weight on features corresponding to kid-friendly or animated movies, while a young adult might have higher weight on features corresponding to action and horror genres.

Our third contribution is to show that our tree-based sampling algorithm can be adapted to efficiently incorporate personal feature weights *at query time*. Thus, we can personalize the sample in a flexible and powerful way while still scaling only logarithmically with the number of items.

Related work. We are aware of one publication that is closely related to ours in terms of sampling time complexity, by Dereziński (2018). That work also provides an exact DPP sampling scheme with complexity sublinear in N, after a one-time pre-processing phase. Which method is faster depends on the relative magnitudes of N, D, and the sample size. In contrast to our work, however, it is not yet known how to generalize the method of Dereziński (2018) to k-DPPs, which generate fixed-size samples, and which are favored in practice (e.g., because a web interface reserves a fixed number of slots for recommendations).

There is also some existing work on approximate sampling. For instance, it has been shown that standard MCMC chains are relatively fast mixing (Anari et al., 2016; Li et al., 2016). However, the mixing time is still linear in N, and hence may not be as practical for large datasets as the method proposed in this work.

2. Background and Notation

A DPP on a ground set $\mathcal{Y}$ of N items is specified by an $N \times N$ positive semi-definite (PSD) kernel matrix L. It defines the following distribution over subsets Y of $\mathcal{Y}$:

$$\mathcal{P}_L(Y) = \det(L_Y)/\det(L + I) \,, \tag{1}$$

where L_Y is the submatrix of L whose rows and columns are indexed by the items in Y.

If we factor the PSD kernel as $L = B^\top B$, where $B \in \mathbb{R}^{D \times N}$ (which is always possible for $D \leq N$), then we can think of entry L_{ij} as a dot product between the columns $\boldsymbol{b}_i$ and $\boldsymbol{b}_j$ of B. A basic property of DPPs is that they prefer *diversity* with respect to the similarity measure defined by this dot product; that is, the probability of two items i and

j appearing together in Y decreases as $\boldsymbol{b}_i^\top \boldsymbol{b}_j$ increases. If we think of $\boldsymbol{b}_i$ as a D-dimensional feature vector for item i, then the DPP $\mathcal{P}_L$ favors sets of items with diverse features (Kulesza & Taskar, 2012, Section 2.2.1).

This diversity property is useful for many applications. In order to make use of it, we often want to generate *samples* from $\mathcal{P}_L$, and although there are 2^N possible subsets $Y \subseteq \mathcal{Y}$, DPPs admit polynomial-time sampling algorithms.

The standard algorithm, due to Hough et al. (2006), requires an eigendecomposition of L and is therefore cubic in N, making it too expensive for many real-world applications. Instead, we focus here on the "dual" algorithm, which is usually preferred when the number of features D is much smaller than the number of items N. For the rest of this paper, we will assume that $D \ll N$; if necessary, random projections can be used to reduce D to a suitable level with minimal loss of accuracy (Gillenwater et al., 2012).

Algorithm 1 Dual DPP sampling
(Kulesza & Taskar, 2012, Algorithm 3)

```
1:  **Input:** eigendecomposition $\{(\boldsymbol{v}_i, \lambda_i)\}_{i=1}^D$ of $C = BB^\top$
2:  $E \leftarrow \emptyset, Y \leftarrow \emptyset$
3:  **for** $i = 1, \ldots, D$ **do**
4:      $E \leftarrow E \cup \{i\}$ w.p. $\lambda_i/(\lambda_i + 1)$
5:  $V \leftarrow \left\{\boldsymbol{v}_i / \sqrt{\boldsymbol{v}_i^\top C \boldsymbol{v}_i}\right\}_{i \in E}$
6:  **while** $|V| > 0$ **do**
7:      Select $j$ from $\mathcal{Y}$ w.p. $\frac{1}{|V|} \sum_{\boldsymbol{v} \in V} (\boldsymbol{v}^\top \boldsymbol{b}_j)^2$
8:      $Y \leftarrow Y \cup j$
9:      Choose $\boldsymbol{v}_0 \in V$ such that $\boldsymbol{b}_j^\top v_0 \neq 0$
10:     $V \leftarrow \left\{\boldsymbol{v} - \frac{\boldsymbol{v}^\top \boldsymbol{b}_j}{\boldsymbol{v}_0^\top \boldsymbol{b}_j} \boldsymbol{v}_0 \;\middle|\; \boldsymbol{v} \in V - \{\boldsymbol{v}_0\}\right\}$
11:     Orthonormalize $V$ w.r.t. $\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle = \boldsymbol{v}_1^\top C \boldsymbol{v}_2$
12: **return** $Y$
```

The dual algorithm (Algorithm 1) begins with an eigendecomposition of the dual kernel $C = BB^\top \in \mathbb{R}^{D \times D}$ and then proceeds in two phases: first, a subset $E \subseteq [D]$ of indices is selected. Then, in the second phase, we sample from the *elementary DPP* defined by E (Kulesza & Taskar, 2012, Definition 2.4). Due to the properties of elementary DPPs, we are guaranteed to sample a set with exactly $|E|$ items.

The second phase of Algorithm 1 can be understood more directly via the elementary DPP's marginal kernel:

$$K = \sum_{i \in E} \frac{1}{\lambda_i} (B^\top \boldsymbol{v}_i)(B^\top \boldsymbol{v}_i)^\top . \tag{2}$$

One can show that item j selected on the first iteration of Line 7 is sampled with probability proportional to K_{jj}. Then, on subsequent iterations, when we have already selected a partial subset of items Y, a new item j is sampled with probability proportional to K^Y_{jj}, where K^Y is the conditional marginal kernel given by:

$$K^Y \triangleq K_{\bar{Y}} - K_{\bar{Y}Y}(K_Y)^{-1} K_{Y\bar{Y}} . \tag{3}$$

Here we use double subscripts to indicate a submatrix whose rows and columns are indexed by different sets. (E.g., $K_{Y\bar{Y}}$ consists of rows Y and columns $\bar{Y}$ from K.) Thus, Algorithm 1 first chooses an elementary DPP indicated by E, which determines the ultimate cardinality (phase one). It then repeatedly samples an item according to the current marginal kernel and conditions the marginal kernel on the selected item (phase two).

Because it directly samples from the multinomial distribution, Algorithm 1 has a linear dependence on N. In this paper we will show how the second phase can be improved such that the dependence becomes $\log N$. Because we do not modify the first phase, our approach is also compatible with k-DPPs, which use an alternative first phase to ensure that exactly k items are sampled (Kulesza & Taskar, 2011). For k-DPPs, this first phase is only slightly more expensive, $\mathcal{O}(kD)$ instead of $\mathcal{O}(D)$ (Kulesza & Taskar, 2012, Algorithm 8).

3. Sublinear Sampling

To sample from a DPP in sublinear time, we cannot afford to compute K_{jj} and K^Y_{jj} for each $j \in \mathcal{Y}$. Instead, the key idea is to create a balanced binary tree of depth $\log N$ such that the root node corresponds to the full set of items and each child node corresponds to half of its parent's items. If we pre-compute and store appropriate summary statistics in this tree (a one-time cost), then we can repeatedly sample in time sublinear in N by exploiting the tree's contents.

We begin by describing the tree construction. Let $\{(\boldsymbol{v}_i, \lambda_i)\}_{i=1}^D$ be an eigendecomposition of the dual kernel C, as in Algorithm 1, let V be the $D \times D$ matrix whose i-th column is equal to $\boldsymbol{v}_i$, and let $\gamma_i = 1/\lambda_i$. We first pre-compute two $D \times N$ matrices G and H:

$$G_{ij} = \boldsymbol{b}_j^\top \boldsymbol{v}_i \;, \qquad H_{ij} = \gamma_i G_{ij} \;. \tag{4}$$

We will use $\boldsymbol{g}_j$ and $\boldsymbol{h}_j$ to represent the jth column of G and H, respectively.

Then, for a tree node corresponding to a set of points $S \subseteq \mathcal{Y}$ we store a D-dimensional vector $\boldsymbol{z}$ and a $D \times D$ matrix A:

$$z_i^{(S)} = \gamma_i \sum_{j \in S} G_{ij}^2 \;, \qquad A_{i_1 i_2}^{(S)} = \sum_{j \in S} H_{i_1 j} H_{i_2 j} \;. \tag{5}$$

Note that $\boldsymbol{z}^{(S)}$ and $A^{(S)}$ can be computed efficiently by recursion: if S, S_ℓ, S_r are the sets of items corresponding to a node and its left and right children, respectively, then we have $\boldsymbol{z}^{(S)} = \boldsymbol{z}^{(S_r)} + \boldsymbol{z}^{(S_\ell)}$ and $A^{(S)} = A^{(S_r)} + A^{(S_\ell)}$.

Algorithm 2 describes how the binary tree for DPP sampling can be constructed efficiently[1]. Note that SPLIT is a function that partitions a set of items into two subsets of (approximately) equal size. We will revisit this function in Section 5.

Algorithm 2 Tree Construction

```
1:  procedure CONSTRUCTTREE($S, \gamma, G, H$)
2:      if $S = \{j\}$ then
3:          $\boldsymbol{z} \leftarrow \gamma \circ g_j^2$
4:          $A \leftarrow h_j h_j^\top$
5:          $\Sigma \leftarrow \boldsymbol{b}_j \boldsymbol{b}_j^\top$                ▷ Personalized only
6:          $\Sigma_{\min}, \Sigma_{\max} \leftarrow \boldsymbol{b}_j \boldsymbol{b}_j^\top$       ▷ Approximate only
7:          return Tree($\boldsymbol{z}, A, \Sigma, \Sigma_{\min}, \Sigma_{\max}, j$)
8:      $S_\ell, S_r \leftarrow$ SPLIT($S$)
9:      $\mathsf{T}_\ell \leftarrow$ CONSTRUCTTREE($S_\ell, \gamma, G, H$)
10:     $\mathsf{T}_r \leftarrow$ CONSTRUCTTREE($S_r, \gamma, G, H$)
11:     $\boldsymbol{z} \leftarrow \mathsf{T}_\ell.\boldsymbol{z} + \mathsf{T}_r.\boldsymbol{z}$
12:     $A \leftarrow \mathsf{T}_\ell.A + \mathsf{T}_r.A$
13:     $\Sigma \leftarrow \mathsf{T}_\ell.\Sigma + \mathsf{T}_r.\Sigma$                  ▷ Personalized only
14:     $\Sigma_{\min} \leftarrow \min(\mathsf{T}_\ell.\Sigma_{\min}, \mathsf{T}_r.\Sigma_{\min})$  ▷ Approximate only
15:     $\Sigma_{\max} \leftarrow \max(\mathsf{T}_\ell.\Sigma_{\max}, \mathsf{T}_r.\Sigma_{\max})$ ▷ Approximate only
16: return Tree($\boldsymbol{z}, A, \Sigma, \Sigma_{\min}, \Sigma_{\max}, \mathsf{T}_\ell, \mathsf{T}_r$)
```

Algorithm 3 Tree-Based Sampling

```
1:  procedure SAMPLE($\mathsf{T}, \lambda, G, H$)
2:      $E \leftarrow \emptyset, Y \leftarrow \emptyset$
3:      for $i = 1, \ldots, D$ do                  ▷ Select elementary DPP
4:          $E \leftarrow E \cup \{i\}$ w.p. $\lambda_i/(\lambda_i + 1)$
5:      $Q \leftarrow 0 \times 0$ matrix                  ▷ Will hold $(K_Y)^{-1}$
6:      for $j = 1, \ldots, |E|$ do
7:          $y \leftarrow$ SAMPLEITEM($\mathsf{T}, E, Y, G, Q$)
8:          $Y \leftarrow Y \cup y$
9:          $Q \leftarrow$ EXTENDINVERSE($Q, G_{Ey}^\top H_{EY}$)   ▷ $\mathcal{O}(|Y|^2)$
        return $Y$
10: procedure SAMPLEITEM($\mathsf{T}, E, Y, G, Q$)
11:     if $\mathsf{T}$ is a leaf then return item at this leaf
12:     $p_\ell \leftarrow$ COMPUTEMARGINAL($\mathsf{T}_\ell, E, Y, G, Q$)
13:     $p_r \leftarrow$ COMPUTEMARGINAL($\mathsf{T}_r, E, Y, G, Q$)
14:     if $\mathcal{U}(0,1) \leq \frac{p_\ell}{p_\ell + p_r}$ then
15:         return SAMPLEITEM($\mathsf{T}_\ell, E, Y, G, Q$)
        return SAMPLEITEM($\mathsf{T}_r, E, Y, G, Q$)
16: procedure COMPUTEMARGINAL($\mathsf{T}, E, Y, G, Q$)
17:     $\Phi \leftarrow G_{EY}^\top\ \mathsf{T}.A_E\ G_{EY}$               ▷ $\mathcal{O}(|E|^2|Y|)$
18:     return $\mathbf{1}^\top \mathsf{T}.\boldsymbol{z}_E - \mathbf{1}^\top (Q \circ \Phi)\mathbf{1}$
19: procedure EXTENDINVERSE($Q, \boldsymbol{u}$)
20:     return Inverse of $\boldsymbol{u}$ appended to $Q^{-1}$; see Hager (1989).
```

Given a tree constructed via Algorithm 2, we can draw a sample from the DPP in sublinear time via Algorithm 3. The main idea is to sample each item by traversing the tree from the root to one of the leaves, choosing a child node at each step according to the probability of sampling an item in the child's subset. That is, given that we have already selected a set of items Y and, to select the next item, have reached the node $S = \{S_\ell \cup S_r\}$, we proceed to S_ℓ with probability:

$$\Pr(S_\ell \mid Y) = \Big[\sum_{j \in S_\ell} K^Y{}_{jj}\Big] \,/\, \Big[\sum_{j \in S} K^Y{}_{jj}\Big]\,. \tag{6}$$

Proposition 1 shows how to compute this probability by combining the statistics stored in the tree with the $|Y| \times |Y|$ marginal matrix K_Y.

Proposition 1. *Consider an elementary DPP with kernel K defined by a subset of indices $E \subseteq [D]$ as in Equation 2. Let Y be a (potentially empty) subset of elements that have already been selected. Then:*

$$\sum_{j \in S} K^Y{}_{jj} = \mathbf{1}^\top \boldsymbol{z}_E^{(S)} - \mathbf{1}^\top \Big[(K_Y)^{-1} \circ \Big(G_{EY}^\top A_E^{(S)} G_{EY}\Big)\Big]\mathbf{1}$$

where K^Y is the conditional marginal kernel (Equation 3), $\circ$ denotes entrywise product, and $\mathbf{1}$ is a vector of all ones.

The proof of this and all subsequent results can be found in the appendix. The most expensive part of this formula is the computation of the matrix product $G_{EY}^\top A_E^{(S)} G_{EY}$, which requires $\mathcal{O}(|E|^2|Y|)$ time. We are now ready to state the overall complexity of the tree-based sampling method.

Theorem 1 (Sublinear DPP sampling)**.** *Let $B \in \mathbb{R}^{D \times N}$ be a feature matrix and let $\{(\boldsymbol{v}_i, \lambda_i)\}_{i=1}^D$ be the eigendecomposition of the dual kernel $C = BB^\top$. Then:*

- *The eigendecomposition of C, the matrices G and H (Equation 4), and the tree $\mathcal{T}$ (Algorithm 2) can be constructed in $\mathcal{O}(ND^2)$ time.*
- *Given these, sampling a set Y of size k from the DPP with kernel $L = B^\top B$ costs $\mathcal{O}(k^4 \log N + D)$.*

For comparison, when using standard dual sampling without the tree we would first construct the C matrix, an $\mathcal{O}(ND^2)$ operation, then eigendecompose it, an $\mathcal{O}(D^3)$ operation. With this as input, drawing each sample costs $\mathcal{O}(k^2 N + D)$; using tree-based sampling replaces the N with $k^2 \log N$. For the common case where N is large and k is small, this can be a dramatic improvement.

4. Personalized Sampling

Can we adapt this approach to allow for user personalization without introducing a linear dependence on N to the sampling? DPPs are often personalized via an arbitrary quality function that independently scales the feature vector $\boldsymbol{b}_i$ for each i (Chen et al., 2017; Wilhelm et al., 2018); however, such a quality function would itself be of size N and hence there would be no hope of generating a personalized sample in time sublinear in N.

[1]Lines marked "Personalized only" are only needed for the algorithm of Section 4, and lines marked "Approximate only" are only needed for the algorithm of Section 5.

Suppose, instead, that we allow each user to place an arbitrary weight w_i on each feature $i \in \{1, \ldots, D\}$. For the discussion that follows, we will consider a single user; let W be a $D \times D$ diagonal matrix where the ith diagonal entry is given by w_i. Then the feature matrix B and the primal and dual kernels L and C can be replaced by personalized versions:

$$\hat{B} = WB, \quad \hat{L} = B^\top WWB, \quad \hat{C} = WBB^\top W \,.$$

To sample from the DPP defined by this new kernel $\hat{L}$, we first need to compute the eigendecomposition $\{(\hat{\boldsymbol{v}}_i, \hat{\lambda}_i)\}_{i=1}^D$ of the dual kernel $\hat{C}$. As there is no simple formula for computing the singular values of WB from those of W and B, we have to do a full eigendecomposition for each user. This can be done in $\mathcal{O}(D^3)$ time. We can then perform the first stage of sampling in the usual manner to select a set of indices E. It remains to show that we can efficiently sample from the elementary DPP defined by E, a nontrivial task given that we cannot afford to create a new tree for each W.

Returning to our original binary tree, instead of storing $\boldsymbol{z}^{(S)}$ and $A^{(S)}$, we instead store a $D \times D$ matrix $\Sigma^{(S)}$ at each node (Lines 5 and 13 of Algorithm 2):

$$\Sigma^{(S)}_{i_1,i_2} = \sum_{j \in S} b_{i_1 j} b_{i_2 j} \,. \tag{7}$$

This matrix captures pairwise interactions of the features. Its computation does not change the big-$\mathcal{O}$ runtime of the tree construction.

Algorithm 4 Personalized Tree-Based Sampling

```
1: procedure SAMPLE($\mathsf{T}, C, W$)
2:     $\hat{C} \leftarrow WCW$
3:     $\hat{\lambda}, \hat{V} \leftarrow$ EIGENDECOMPOSE($\hat{C}$)                 ▷ $\mathcal{O}(D^3)$
4:     $E \leftarrow \emptyset, Y \leftarrow \emptyset$
5:     for $i = 1, \ldots, D$ do                         ▷ Select elementary DPP
6:         $E \leftarrow E \cup \{i\}$ w.p. $\hat{\lambda}_i/(\hat{\lambda}_i + 1)$
7:     $\hat{\gamma}_E \leftarrow (\hat{\lambda}_E)^{-1}$
8:     $\hat{Q} \leftarrow 0 \times 0$ matrix                        ▷ Will hold $(\hat{K}_Y)^{-1}$
9:     $\hat{H} \leftarrow |E| \times 0$ matrix                       ▷ Will be $|E| \times |Y|$
10:    $M \leftarrow \hat{V}_{:,E}^\top W$
11:    $R \leftarrow M^\top \mathrm{diag}(\hat{\gamma}_E) M$
12:    for $j = 1, \ldots, |E|$ do
13:        $F \leftarrow \hat{H}^\top M$
14:        $y \leftarrow$ SAMPLEITEM($\mathsf{T}, R, F, \hat{Q}$)
15:        $Y \leftarrow Y \cup y$
16:        $\boldsymbol{x} \leftarrow M \boldsymbol{b}_y$
17:        $\hat{H} \leftarrow$ APPENDCOLUMN($\hat{H}, \boldsymbol{x} \circ \hat{\gamma}_E$)
18:        $\hat{Q} \leftarrow$ EXTENDINVERSE($\hat{Q}, \boldsymbol{x}^\top \hat{H}$)           ▷ $\mathcal{O}(|Y|^2)$
       return $Y$
19: procedure SAMPLEITEM($\mathsf{T}, R, F, \hat{Q}$)
20:     Analogous to the non-personalized case
21: procedure COMPUTEMARGINAL($\mathsf{T}, R, F, \hat{Q}$)
22:     $\Phi \leftarrow F\, \mathsf{T}.\Sigma\, F^\top$                                   ▷ $\mathcal{O}(|Y|D^2)$
23:     return $\mathbf{1}^\top (R \circ \mathsf{T}.\Sigma)\mathbf{1} - \mathbf{1}^\top (\hat{Q} \circ \Phi)\mathbf{1}$
```

Algorithm 4 shows how this augmented tree can be used for personalized sampling. As in the non-personalized case, we sample an item by moving from the root of the tree down to one of the leaves. The probability for narrowing down from S to S_ℓ is the same as in Equation 6, but with K replaced by $\hat{K}$, the marginal kernel for $\hat{L}$. Proposition 2 shows how to compute this probability by combining the statistics stored in the tree with the $|Y| \times |Y|$ marginal matrix $\hat{K}_Y$.

Proposition 2. *Consider a* personalized *elementary DPP with kernel $\hat{K}$ defined by a subset of indices $E \subseteq [D]$ as in Equation 2, but with feature matrix B replaced by $\hat{B} = WB$. Let the eigendecomposition of $\hat{C} = \hat{B}\hat{B}^\top$ be $\{(\hat{\boldsymbol{v}}_i, \hat{\lambda}_i)\}_{i=1}^D$. Let Y be a (potentially empty) subset of elements that have already been selected. Define:*

$$\hat{\Gamma} = \mathrm{diag}(1/\hat{\lambda}) \,, \quad M = \hat{V}_{:,E}^\top W \,, \quad \hat{H} = \hat{\Gamma}_E M B_{:,Y} \,, \tag{8}$$

$$R = M^\top \hat{\Gamma}_E M \,, \quad F = \hat{H}^\top M \,, \tag{9}$$

where diag *denotes a diagonal matrix and the ":" subscript indicates selection of all rows. Then:*

$$\sum_{j \in S} \hat{K}^Y_{jj} = \mathbf{1}^\top \left[R \circ \Sigma^{(S)}\right]\mathbf{1} - \mathbf{1}^\top \left[(\hat{K}_Y)^{-1} \circ (F\Sigma^{(S)}F^\top)\right]\mathbf{1} \,, \tag{10}$$

where $\hat{K}^Y$ is the conditional marginal kernel (Equation 3), $\circ$ denotes entrywise product, and $\mathbf{1}$ is a vector of all ones.

The most expensive part of this formula is the computation of the matrix product $F\Sigma^{(S)}F^\top$, which requires $\mathcal{O}(|E|D^2)$ time. We are now ready to state the overall complexity of the personalized tree-based sampling method.

Theorem 2 (Sublinear personalized DPP sampling)**.** *Let $\hat{B} = WB \in \mathbb{R}^{D \times N}$ be a personalized feature matrix and let $C = BB^\top$ be the non-personalized dual kernel. Then:*

- *The dual kernel C and the tree $\mathcal{T}$ (Algorithm 2) can be constructed in $\mathcal{O}(ND^2)$ time.*
- *Given these, sampling a set Y of size k from the DPP with kernel $\hat{L} = \hat{B}^\top \hat{B}$ costs $\mathcal{O}(k^2 D^2 \log N + D^3)$.*

For comparison, when using standard dual sampling, drawing each sample costs $\mathcal{O}(k^2 N + D^3)$ (after preprocessing); using tree-based sampling replaces the N with $D^2 \log N$. Compared to unpersonalized tree-based sampling (Theorem 1), personalization replaces a factor of k^2 with D^2.

5. Approximate Sampling

We can further speed up sampling by introducing approximations that leverage the structure of our tree. In particular, we show that we can use the information stored in the tree to efficiently compute how far the DPP sampling distribution at a node S lies from the *uniform* distribution over S. Intuitively, if the DPP distribution is nearly uniform (say, within some ϵ), then we can avoid traveling further down the tree and simply choose an item from S uniformly, incurring only a small loss in fidelity. This also allows us to potentially keep less of the tree cached in memory.

We will consider only the personalized sampling case; the non-personalized case is recovered by setting $W = I$.

Proposition 3. *Let $\Pr(j \mid S, Y)$ be the probability of sampling item $j \in S$ under an elementary DPP with feature matrix $\hat{B} = WB$ and indicator indices $E \subseteq [D]$, conditioned on the fact that items in the (potentially empty) subset Y have already been selected. Then, letting:*

$$\tilde{\Sigma}^{(S)}_{\ell_1 \ell_2} = \max_{j \in S} \left| \Sigma^{(j)}_{\ell_1, \ell_2} - \frac{1}{|S|} \Sigma^{(S)}_{\ell_1 \ell_2} \right| ,$$

the distance from $\Pr(j \mid S, Y)$ to uniform sampling over the subtree with S at its root is upper-bounded as follows:

$$\Big| \Pr(j \mid S, Y) - \frac{1}{|S|} \Big| \leq \big(\sum_{j \in S} \hat{K}^{Y}{}_{jj} \big)^{-1} \times$$
$$\Big[\mathbf{1}^\top \big[|R| \circ \tilde{\Sigma}^{(S)} \big] \mathbf{1} + \mathbf{1}^\top \big[|(\hat{K}_Y)^{-1}| \circ \big(|F| \tilde{\Sigma}^{(S)} |F^\top| \big) \big] \mathbf{1} \Big] ,$$

where $| \cdot |$ is elementwise absolute value, and the matrices R and F are as defined in Proposition 2.

Note that $\tilde{\Sigma}$ can be computed by storing intermediate values $\Sigma_{\min}$ and $\Sigma_{\max}$ (as shown in Lines 6, 14, and 15 of Algorithm 2) via $\tilde{\Sigma} = \max(|\Sigma_{\min} - \frac{1}{|S|}\Sigma|, |\Sigma_{\max} - \frac{1}{|S|}\Sigma|)$. Thus, the overall upper bound can be computed in $\mathcal{O}(|E|D^2)$ time, just like the expression from Proposition 2.

Algorithm. To exploit Proposition 3 for approximate tree-based sampling, we modify SAMPLEITEM by adding the following check at its start: if the upper bound on the difference from uniform is $\leq \frac{\epsilon}{|S|}$, then return y sampled uniformly from $S \setminus Y$. The following theorem characterizes the quality of a sample obtained via this modified algorithm.

Theorem 3. *Let $Y = \{y_1, \ldots, y_k\} \subseteq \mathcal{Y}$. Let $B \in \mathbb{R}^{D \times N}$ be a feature matrix defining a DPP. Let $p(Y)$ be the probability of sampling Y in the order $y_1, \ldots, y_k$ from the tree-based sampling algorithm, and let $q(Y)$ be the probability of sampling Y in order from the approximate tree-based sampling algorithm. Then $|p(Y) - q(Y)| \leq (1 + \epsilon)^k - 1$.*

Splitting function. In order to make it more likely for the approximation to apply, we can engage a so-far unexplored degree of freedom: the node splitting function used to divide into left and right subtrees (see Line 8 of Algorithm 2). The splitter could be relatively cheap; e.g., find two distinct items in the parent node, seed left and right subtrees with these, and then place each remaining item into the subtree with the most similar seed. The splitter could also be much more sophisticated; e.g., build a complete graph over a node's items with pairwise similarities as edge weights, then run a min-cut algorithm to partition. In general, if a splitter requires time $\mathcal{O}(|S|^r D)$ for a node with items S, then the overall tree construction complexity is increased from $\mathcal{O}(ND^2)$ to $\mathcal{O}(ND^2 + N^r D \log N)$. A more expensive splitter means slower tree construction, but this might pay off if the tree will be used to generate many samples.

6. Experiments

We now present empirical results demonstrating the performance of our sampling algorithms on both synthetic and real-world data sets. Throughout, we use k-DPP sampling (Kulesza & Taskar, 2011) to fix the size of the sampled sets, as is frequently done in practice. We compare our tree-based sampler to the fastest existing exact sampling algorithm for k-DPPs (Gillenwater, 2014, Algorithm 3). We will refer to this method as "the dual sampler (DS)".

6.1. Preprocessing costs

Figure 1 shows the time and memory required to complete the preprocessing step for each algorithm (precomputing eigendecompositions and feature products for DS, and generating the tree for our sampler). Results are only shown for the personalized case, but the unpersonalized results are similar. Only the dimensions of the feature matrix B affect these results—the actual feature values are irrelevant—so synthetic data are used here.

Although for many applications we can treat the preprocessing step as a one- time, latency-insensitive operation, the trends here are interesting, and memory use in particular may be a relevant limitation in some cases. While the tree grows linearly with N, and is only a constant factor larger than the storage required by DS, the dependence on D is quadratic (versus linear for DS). Thus, our sampler may require keeping the number of features relatively low. This is usually not an issue in practice, especially if using random projections (Gillenwater et al., 2012). A modern machine can easily handle millions of items with tens of features

6.2. Sampling speed

Figure 2 shows the actual time required to compute a single DPP sample under various settings. Again, the contents of the feature matrix do not affect the speed of sampling, so synthetic data are used here as well.

Several interesting trends are apparent. Firstly, our sampler is faster than DS in essentially all cases (though this would probably change for ≤ 1000 items). While the scaling is similar to DS for the number of features D and the sample

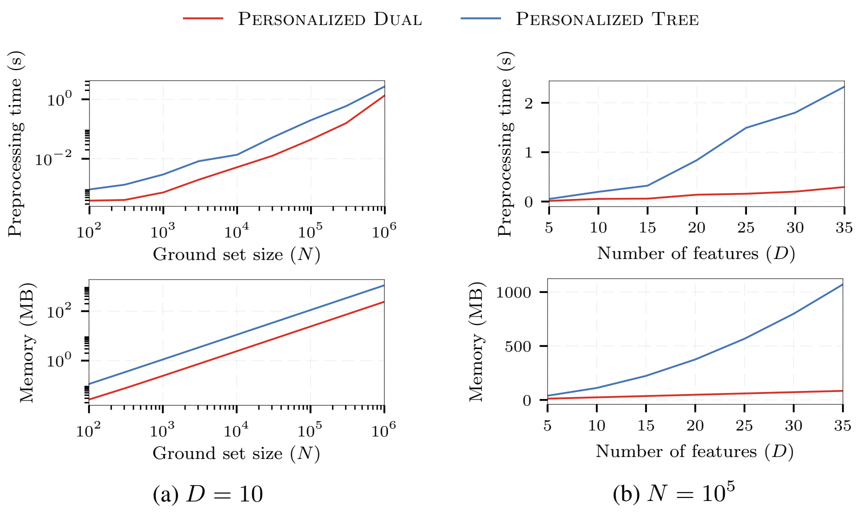


Figure 1. Time and memory required for preprocessing. Each point is an average over ten samples.

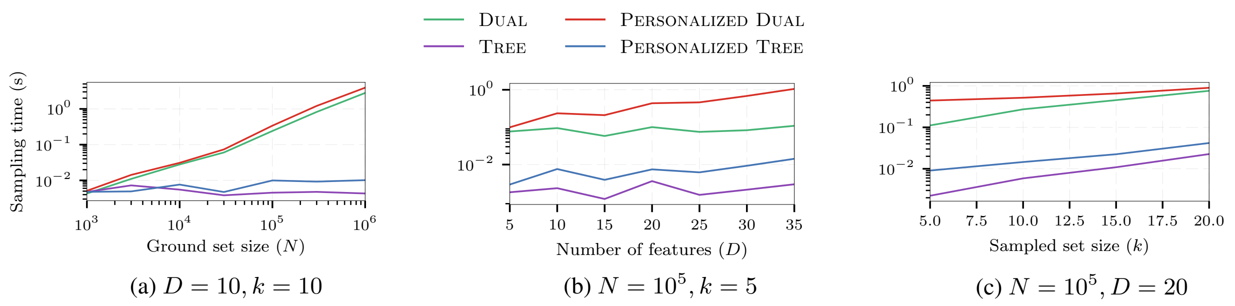


Figure 2. Time required to produce a k-DPP sample. Each point is an average over 10,000 samples.

size k, our sampling scales logarithmically with N, so for large numbers of items it is several orders of magnitude faster than DS. This is precisely the setting in which modern systems usually find themselves. When N is one million, our sampler returns a personalized result in about 0.01 seconds, versus almost 4 seconds for DS. Secondly, the costs for the method of personalization that we have proposed are quite low; the overhead is negligible in most cases.

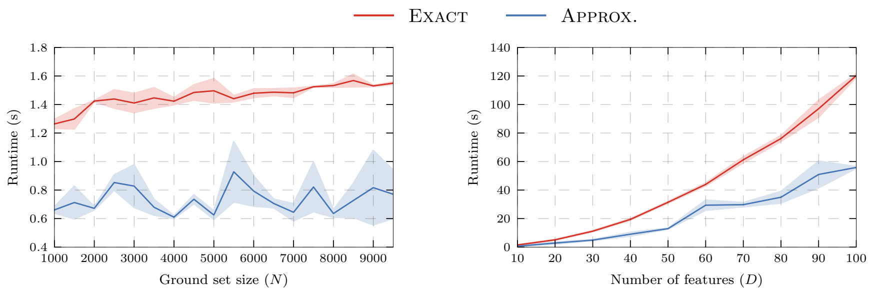
6.3. Exact vs. approximate sampling

To synthetically evaluate the approximation technique proposed in Section 5, we generate feature matrices B such that each column $\boldsymbol{b}_i$ is drawn from a mixture of multivariate Gaussians with n components, and we draw personalization vectors $\boldsymbol{w}$ uniformly at random from $[0, 1]^D$.

When constructing the tree, we use the following simple splitting heuristic: we use the items i, j that maximize $\|\boldsymbol{b}_i - \boldsymbol{b}_j\|_\infty$ to initialize the left and right subtrees; all other items are then greedily added to the left or right tree in order to minimize their (infinity-norm) distance to the initial item. This encourages similar nodes to lie in the same subtree, leading to more uniform distributions. This splitter requires $\mathcal{O}(|S|^2 D)$ time to compute, and so the tree construction in this case is $\mathcal{O}(ND^2 + N^2 D \log N)$.

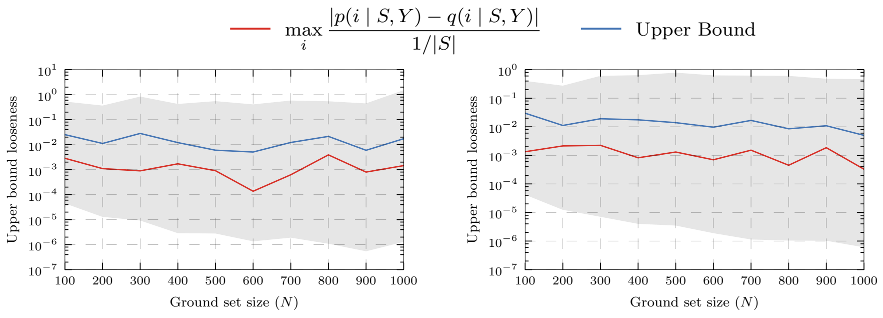
Figure 3 reports the time required to sample sets of $k = 5$ elements for various feature lengths and ground set sizes. We check for approximate sampling between depths 5 and $d - 2$ where d is the total depth of the tree. Approximate sampling is significantly faster under all conditions, and the gap increases as $\log(N)$ if D and k remain constant.

We also investigate the tightness of the upper bound in Prop. 3. Figure 4 shows the predicted upper bound (blue) and the true distance to the uniform distribution (red) for varying ground set sizes; the reported measurements are averaged over all nodes in the tree for each fixed ground set size, with $D = 10$ features and $n = 5$ components to the Gaussian mixture generating B. Figure 4 also shows the spread of the difference between our upper bound and the true distance to uniform: although it is very loose, the two values differ by less than 10^{-6} for certain sizes, showing that our upper bound is (theoretically) tight.



(a) $D = 10, k = 5, n = 5$ (b) $N = 5000, k = 5, n = 5$

Figure 3. Comparison of exact and approximate personalized tree-based DPP sampling ($\epsilon = 0.1$).



(a) $D = 10, n = 5, Y = \emptyset$ (b) $D = 10, n = 5, |Y| = 4$

Figure 4. Comparison of the upper bound (Prop. 3) and the true distance to uniform. The shaded grey area represents the minimum and maximum gap between the two lines.

6.4. Movie recommendation

As a practical demonstration of the proposed techniques, we build a realistic movie recommender system based on the MovieLens dataset (Harper & Konstan, 2016), which contains over 20M user ratings for over 25k movies. To obtain a feature vector for each movie suitable for constructing a DPP kernel, we apply nonnegative matrix factorization to decompose the rating matrix into factors of rank 30. Details are omitted due to space constraints, but our factorization achieves an RMSE of 0.88 on held out test data, which is roughly comparable to existing results in the literature (Williamson & Ghahramani, 2008; Sedhain et al., 2015).

We use the resulting movie factor matrix as B ($N = 26744$, $D = 30$) to define the DPP kernel, and the corresponding user factor matrix as a source of realistic weights for personalization, uniformly sampling rows to simulate the arrival of random users at our recommendation engine. Table 1 shows that, as before, our sampler dramatically outperforms DS.

Of course, as noted above, the time required to compute exact samples does not depend on the data itself, so perhaps the more interesting aspect here is the effect of the uniform approximation. On a random subset of 5000 movies, using a threshold of $\epsilon = 0.4$, approximate sampling is on average 1.5 times faster than exact sampling with personalization.

Sampler	Time per sample
DUAL	42.2 ms
TREE	3.2 ms
PERSONALIZED DUAL	52.2 ms
PERSONALIZED TREE	5.8 ms

Table 1. Time to sample a set of movie recomendations, averaged over 10,000 samples.

This suggests that the gains illustrated synthetically in Figure 3, which depend on pockets of similar items that can safely be grouped together during inference, translate to realistic data as well.

7. Conclusion

Our tree-based sampling algorithm can be used to generate DPP samples in sublinear time after an initial pre-processing phase. This includes personalized samples that depend on user preferences. Our algorithm dramatically outperforms existing sampling algorithms for large item sets, making large-scale DPP systems practical. The unique tree structure we use also enables approximations that reduce the computational costs even further, and we suspect that further optimizations may be possible.

References

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

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

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

Dereziński, M. Fast Determinantal Point Processes via Distortion-Free Intermediate Sampling, 2018.

Gillenwater, J. *Approximate Inference for Determinantal Point Processes*. PhD thesis, University of Pennsylvania, 2014.

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

Hager, W. Updating the Inverse of a Matrix. *SIAM Review*, 31(2), 1989.

Harper, F. M. and Konstan, J. A. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems*, 5(4), 2016.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems*, 22 (1), 2004.

Hough, J. B., Krishnapur, M., Peres, Y., and Virág, B. Determinantal Processes and Independence. *Probability Surveys*, 3, 2006.

Hurley, N. and Zhang, M. Novelty and Diversity in Top-N Recommendation–Analysis and Evaluation. *ACM Transactions on Internet Technology (TOIT)*, 10(4), 2011.

Krause, A., Singh, A., and Guestrin, C. Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. *Journal of Machine Learning Research (JMLR)*, 9, 2008.

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

Kulesza, A. and Taskar, B. Determinantal Point Processes for Machine Learning. *Foundations and Trends in Machine Learning*, 5, 2012.

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

Lin, H. and Bilmes, J. Learning Mixtures of Submodular Shells with Application to Document Summarization. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2012.

Mariet, Z. and Sra, S. Diversity Networks: Neural Network Compression Using Determinantal Point Processes. 2016a.

Mariet, Z. and Sra, S. Kronecker Determinantal Point Processes. In *Neural Information Processing Systems (NIPS)*, 2016b.

Sedhain, S., Menon, A. K., Sanner, S., and Xie, L. Autorec: Autoencoders Meet Collaborative Filtering. In *International Conference on the World Wide Web (WWW)*, 2015.

Smyth, B. and McClave, P. Similarity vs. Diversity. In *International Conference on Case-Based Reasoning*, 2001.

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

Williamson, S. and Ghahramani, Z. Probabilistic Models for Data Combination in Recommender Systems. In *Learning from Multiple Sources (NIPS Workshop)*, 2008.

Zhou, T., Kuscsik, Z., Liu, J., Medo, M., Wakeling, J., and Zhang, Y. Solving the Apparent Diversity-Accuracy Dilemma of Recommender Systems. *Proceedings of the National Academy of Sciences*, 107(10), 2010.

Ziegler, C.-N., McNee, S. M., Konstan, J. A., and Lausen, G. Improving Recommendation Lists Through Topic Diversification. In *International Conference on the World Wide Web (WWW)*, 2005.