Diversity on the Go! Streaming Determinantal Point Processes under a Maximum Induced Cardinality Objective

Paul Liu*
Stanford University
Stanford, CA, USA
paul.liu@stanford.edu

Akshay Soni
Microsoft
Sunnyvale, CA, USA
akson@microsoft.com

Eun Yong Kang
Microsoft
Sunnyvale, CA, USA
eun.kang@microsoft.com

Yajun Wang
Microsoft
Sunnyvale, CA, USA
yajunw@microsoft.com

Mehul Parsana
Microsoft
Bellevue, WA, USA
mparsana@microsoft.com

ABSTRACT

Over the past decade, Determinantal Point Processes (DPPs) have proven to be a mathematically elegant framework for modeling diversity. Given a set of items N, DPPs define a probability distribution over subsets of N, with sets of larger diversity having greater probability. Recently, DPPs have achieved success in the domain of recommendation systems, as a method to enforce diversity of recommendations in addition to relevance. In large-scale recommendation applications however, the input typically comes in the form of a *stream* too large to fit into main memory. However, the natural greedy algorithm for DPP-based recommendations is memory intensive, and cannot be used in a streaming setting.

In this work, we give the first streaming algorithm for optimizing DPPs under the Maximum Induced Cardinality (MIC) objective of Gillenwater et al. [15]. As noted by [15], the MIC objective is better suited towards recommendation systems than the classically used maximum a posteriori (MAP) DPP objective. In the insertion-only streaming model, our algorithm runs in $\tilde{O}(k^2)$ time per update and uses $\tilde{O}(k)$ memory, where k is the number of diverse items to be selected. In the sliding window streaming model, our algorithm runs in $\tilde{O}(\sqrt{n}k^2)$ time per update and $\tilde{O}(\sqrt{n}k)$ memory where n is the size of the sliding window. The approximation guarantees are simple, and depend on the largest and the k-th largest eigenvalues of the kernel matrix used to model diversity. We show that in practice, the algorithm often achieves close to optimal results, and meets the memory and latency requirements of production systems. Furthermore, the algorithm works well even in a non-streaming setting, and runs in a fraction of time compared to the classic greedy algorithm.

*This work done by the author as an intern with Microsoft, CA, USA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WWW '21, April 19–23,
2021, Ljubljana, Slovenia
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3450089

CCS CONCEPTS

• **Theory of computation → Streaming models**; • **Information systems → Information retrieval diversity**.

KEYWORDS

determinantal point process, maximum induced cardinality, streaming, low-memory, diversity and relevance

ACM Reference Format:
Paul Liu, Akshay Soni, Eun Yong Kang, Yajun Wang, and Mehul Parsana. 2021. Diversity on the Go! Streaming Determinantal Point Processes under a Maximum Induced Cardinality Objective. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3442381.3450089

1 INTRODUCTION

In many retrieval applications, results returned take the form of a list ordered by relevance scores. Such applications include search retrieval tasks and recommendation systems. In these cases, the algorithm typically returns a subset of items from some global item pool N, where a point-wise relevance score r_i is assigned to each item $i \in N$. Unless a system explicitly takes diversity into account, items with high relevance scores are in general highly correlated with each other [25]. For recommendations however, the solutions typically require *diversity* in addition to relevance, since one major goal of the recommendations is to entice users to explore unfamiliar contents. As such, diversity has remained a critical focus of recommendation system research [17, 25, 31].

Determinantal Point Processes. Over the past decade, DPPs have proven to be a mathematically elegant framework for diversity [21]. Instead of assigning scores to items individually in N, a DPP can take into account negative correlations between different items and score sets of items as a whole. Scores of different sets are interpreted as probabilities, and sets with larger probabilities correspond to more diverse sets. To be precise, given a positive semi-definite matrix $L \in \mathbb{R}^{|N|\times|N|}$, a DPP is a probability distribution on $S \subseteq N$ defined by

$$\Pr(S) = \frac{\det(L_S)}{\det(L+I)}$$

where L_S is the $|S|\times|S|$ submatrix induced by rows and and columns in $|S|$. To get a k-set of maximum diversity, one can solve a *maximum*

a posteriori (MAP) optimization problem:

$$\max_{|S|=k} \Pr(S) = \max_{|S|=k} \frac{\det(L_S)}{\det(L+I)}. \quad (1)$$

In contrast with other probabilistic models, DPPs provide efficient algorithms for common operations such as marginalization, sampling, and conditioning [21]. Furthermore, an efficient greedy solution to the MAP problem above can be implemented in $O(|N|k^2)$ time and performs well in practice [9]. Owing to the flexibility of DPPs, applications have been found in diverse areas such as document summarization, image search, product recommendation, and pose estimation [5, 19, 20, 29, 30].

In its original formulation, DPPs are limited to modelling negative local and global correlations. Recently, much work has gone into improving the weaknesses of the original DPP formulation, resulting in DPP variants which now allow for fixing the sample cardinality and modelling positive correlations [4, 11, 12, 14, 18, 22]. As a diversity measure however, one weakness of the MAP objective is that it is unstable under low rank conditions: if $k > \text{rank}(L)$ then a k-set has zero probability of occurring. This strongly conflicts with the intuitive notion that recommending more items should increase user engagement. To address this issue, Gillenwater et al. [15] show that a *maximum induced cardinality* (MIC) objective better fits the setup of recommendation systems used in practice. The MIC objective aims to find a subset S which maximizes the expected cardinality of a subset selected by the DPP distribution within S:

$$\max_{|S|=k;S\subseteq N} \sum_{E\subseteq S} |E| \cdot \Pr(E \mid S) = \max_{|S|=k;S\subseteq N} \text{Tr}\left(I-(L_S+I)^{-1}\right). \quad (2)$$

Need for diversity in streaming settings. In many large-scale applications, the input typically comes in the form of a *stream*, where input is fed to the algorithm in an online manner. We overload the notation used for the item pool N, and refer to the items in the stream by N. In applications, the entire stream is too large to fit into main memory, so techniques are needed to compress or summarize the stream. Such memory considerations are typically not considered in traditional DPP settings, as the item set is usually small. For instance, the fast greedy algorithm of Chen et al. requires memory proportional to $O(|N|k)$ which is prohibitive in the applications described below.

One of the important applications of our work is in online recommendation systems. In these applications, users continuously perform web activities that are used as signals for item recommendations. For instance, in e-commerce, the user activities correspond to their interaction with retail products and the goal is to recommend them other relevant products. Similarly, in ads, the browse and search activities of users are utilized to show them ads they would be interested in. These approaches work as follows:

- For each user activity, a subset of items are selected. The way this selection is done is application specific and may use the current user activity and additional information.
- At the time when recommendations are to be made for a user, a set of candidate items is created as the union of all the selected items for this user over her past activities.
- The items in the candidate set are ranked[1] and the top-k items are presented to the user as final recommendations.

It is then clear that the candidate set is streaming in nature, where whenever a user does a new activity, large number of relevant items are added to this set while the items corresponding to her oldest activity are discarded. Given this streaming candidate set for each user, we provide algorithms that generates recommendations that are engaging (depends on the ranking function) as well as diverse. In practice, large-scale recommendation systems have billions of users and each user does hundreds of activities a day. As such, practical recommendation algorithms face restrictive latency and memory requirements.

Many approaches have been adopted for recommendations in the streaming setting, using techniques such as non-negative matrix factorizations, graph neural networks, and neighbourhood-based collaborative filtering methods [6, 7, 16, 26, 28]. While these approaches are streaming, the main goal for them is to extract a subset of most engaging items. This is usually done by updating the model parameters or the some form of user representation with the new data coming through the stream. In this work, we assume that we have access to a recommendation system that can score an item for its engagement value and we provide a mechanism to select a subset of these items that is collectively most engaging as well as diverse.

Due to the success of DPP-based algorithms in modelling diversity, a natural question to ask is whether DPPs can be used in the streaming setting described above. The most relevant works here appear in the domain of submodular optimization and composable coresets, where algorithms have been developed for both dynamic streams and sliding window streams [3, 8, 13, 24]. In the composable coresets approach of Indyk et al. [24], the results imply a $O(\sqrt{nk})$ memory algorithm for sliding window streams achieving an approximation ratio of $2^{-O(k\log k)}$ where n is the size of the sliding window. Since the MAP objective of Equation (1) is log-submodular, general submodular optimization algorithms can also be used. For sliding window streams in particular, the algorithm of [13] produces a solution of value at least $\sqrt{OPT}$ in $O(\sqrt{n}k^2)$ memory, where OPT is the optimal MAP value. In contrast, not much is known in the streaming model for the MIC objective proposed by [15]. The current state-of-the-art relies on approximating the MIC objective by a submodular function, through a suitably truncated Taylor series [15]. This is unfortunate, as the DPP kernel L is often not close to full rank in practical applications. In the web advertising example above, this issue is prevalent as users tend to do similar activities repeatedly.

Our contributions. In this work, we provide the first streaming algorithms for optimizing DPPs under an MIC objective.

- InsertOnly: In the insert-only streaming setup, where the new elements are added to the stream without removing any existing elements, our InsertOnly algorithm runs in $\tilde{O}(k^2)$ time per update and use $\tilde{O}(k)$ memory. Here $\tilde{O}$ represents approximate order without the logarithmic terms for clearer exposition.
- Mantis: In the sliding window streaming setup, elements are added to the front of the window and deleted from the back of the window. Here we are interested in a diverse subset

[1] Our algorithms are oblivious to the details of the ranking function. Though there is an implicit assumption that the ranking function gives same score for a user-item pair and higher score means better predicted engagement.

over a window of size n, and our algorithm runs in $\tilde{O}(\sqrt{n}k^2)$ time per item with $\tilde{O}(\sqrt{n}k)$ memory. We name this algorithm MANTIS as an acronym for MAximum iNduced cardinaliTy In Streaming.
- MANTIS is amenable to parallelism and can use multiple cores efficiently bringing the runtime down to $\tilde{O}(\sqrt{n}k^2/\#cores)$.
- We compare our algorithms with the GREEDY algorithm [9] on multiple datasets. Our results show that MANTIS works at a fraction of memory cost and computational time minimal impact to performance.

Table 1 summaries the memory and update times of our algorithms. Note that the two proposed algorithms are operate in different streaming scenarios. MANTIS is strictly more general than INSERTONLY as it operates on sliding windows. In addition to fast update and low memory complexity, we show approximation guarantees for the algorithms. The guarantees depend on the largest and k-th largest eigenvalues of the kernel matrix L used to model diversity. Our approach is most similar to the approach of Espato et al. [13] for submodular functions, as we utilize a threshold greedy approach that appears often in the submodular optimization literature. We show that in practice, the algorithm often achieves close to optimal results, even when compared to the classic greedy algorithm running with unlimited memory. The code, as well as all data used in the experiments, can be found at https://gitlab.com/paul.liu.ubc/streaming-mic-dpp.

Algorithm	Update Time	Memory	Type
INSERTONLY	$\tilde{O}(k^2)$	$\tilde{O}(k)$	Insert-only
MANTIS	$\tilde{O}(\sqrt{n}k)$	$\tilde{O}(\sqrt{n}k^2)$	Sliding window

Table 1: Summary of our algorithms.

2 BACKGROUND

Determinantal point processes. Given a set of items N and a positive semi-definite matrix $L \in \mathbb{R}^{|N|\times|N|}$, a *determinantal point process* (DPP) is a probability distribution on $S \subseteq N$ defined by

$$\Pr(S) = \frac{\det(L_S)}{\det(L+I)}$$

where L_S is the $|S|\times|S|$ submatrix induced by rows and columns in $|S|$. The L is referred to as the *kernel* matrix of the DPP. As a convention, we set $\Pr(\emptyset) := \frac{1}{\det(L+I)}$, ensuring that the probabilities sum to 1.

In applications, $n := |N|$ items are given as input, of which k items must be selected that are "relevant" as well as "diverse". Each item has an associated score r_i measuring a notion of relevance and a feature vector $v_i \in \mathbb{R}^d$ with $||v_i||_2 = 1$. These feature vectors and relevance scores are typically computed upstream by specific algorithms whose details are not relevant to this paper. As outlined by Kuleza and Taskar [21], the matrix L can be "roughly" interpreted as a correlation matrix, with off-diagonal elements modelling negative correlation. Since similarity between different items are captured by the inner product of their feature vectors, a suitable kernel used in practice is $L = V^T V$, where $V \in \mathbb{R}^{d\times n}$ is the stacked matrix of feature vectors from the n items. In this inner-product based DPP kernel, in order to write a joint optimization problem for relevance and diversity, the feature vector of each item is generally scaled to have $||v_i||_2 = r_i$, see [30].

The typical way to get a k-set of maximum diversity is to solve the MAP optimization problem:

$$\max_{|S|=k} \Pr(S) = \max_{|S|=k} \frac{\det(L_S)}{\det(L+I)}. \tag{3}$$

Optimizing maximum induced cardinality. In work by Gillenwater et al. [15], it is shown that the *maximum induced cardinality* (MIC) objective better fits the setup of recommendation systems. Instead of maximizing the a posteriori probability, the MIC objective aims to maximize the expected cardinality of a set selected by the DPP distribution

$$\max_{|S|=k; S\subseteq N} \sum_{E\subseteq S} |E| \cdot \Pr(E \mid S). \tag{4}$$

As shown in [15], the objective function can be written in an elegant form:

$$\begin{aligned} f(S) &:= \sum_{E\subseteq S} |E| \cdot \Pr(E \mid S) \\ &= \sum_{E\subseteq S} |E| \cdot \frac{\det(L_E)}{\det(L_S+I)} \\ &= \mathrm{Tr}\Big(I - (L_S+I)^{-1}\Big). \end{aligned} \tag{5}$$

The equalities above follow from work by Kulesza and Taskar [21].

Compared with the MAP objective, the MIC objective $f(S)$ is monotone and fractionally subadditive [15]. Thus even if $\mathrm{rank}(L) < k$, the MIC objective will assign more value to a k-set than any of its subsets. This corresponds with the notion that recommending more items increases user engagement.

Comparison between MAP and MIC. To view MAP and MIC in a unified manner, note that the optimal solution of the MAP objective is unchanged after taking logarithms. Furthermore, $\log\det(L_S) = \mathrm{Tr}(\log(L_S))$. Let $\lambda_{i,S}$ be the i-th eigenvalue of the submatrix L_S and $\mathbf{w}_{i,S}$ be the associated eigenvector. Writing the two objectives in terms of eigenvalues, we have

$$\mathrm{Tr}(\log(L_S)) = \sum \log(\lambda_{i,S}) \qquad \text{(log-MAP objective)}$$

$$\mathrm{Tr}\Big(I - (L_S+I)^{-1}\Big) = \sum \frac{\lambda_{i,S}}{1+\lambda_{i,S}} \qquad \text{(MIC objective)}$$

It's clear that the MIC objective is more stable in low-rank settings as it avoids the pole at 0 that is in the log function. Indeed, a family of such objectives could be constructed as a function of the eigenvalues of the kernel matrix L_S[2].

Notation used in this paper. Throughout this paper, we use + and − to denote the set union and set subtraction operations where convenient (i.e. $S + e = S \cup \{e\}$ and $S - e = S \setminus \{e\}$). Let $\Delta(e|S) := f(S+e) - f(S)$, we refer to this quantity as the *marginal* of e with respect to S. We will also frequently make bounds with respect to

[2]In the inner product DPP kernel setting, MAP **maximizes** the following objective function over S:

$$\sum_{i=1}^{k} (\log(r_i^2) + \log(\lambda_{i,S})).$$

The MIC objective on the other hand **minimizes** the following objective function over S:

$$\sum_{i=1}^{k} \frac{1}{\lambda_{i,S} + r_i^2}.$$

Both of these objective will simultaneously optimize for relevance and diversity albeit via different objective functions.

Algorithm 1 MaxTrace(N)

1: $G \leftarrow \text{argmax}_{\tilde{G} \subseteq N} \text{Tr}\left(L_{\tilde{G}}\right)$
2: **return** G

the eigenvalues of L_S for some DPP kernel L and subset S. We denote the eigenvalues of L_S by $\lambda_{i,S}$, where the eigenvalues are sorted in *decreasing* order. Finally, we say that an algorithm has approximation ratio $0 < c \leq 1$ if the solution G obtained by the algorithm is at least c times the value of the optimal solution.

Memory model. When computing memory used in our algorithms, we assume that given the elements of a set S, an oracle returns the kernel matrix L_S without needing to store any additional information. Thus we write all of our bounds in terms of the number of "elements" stored, without referring to any exact representation (such as semantic vectors). Our bounds can be modified on a case-by-case basis depending on the precise representation of data.

3 SIMPLE ALGORITHMS FOR MAXIMIZING INDUCED CARDINALITY

In this section, we introduce several algorithms for maximizing the MIC objective, in both the standard and streaming model of computing.

Optimizing relevance without diversity. As a baseline approximation, we begin with a simple algorithm for approximating MIC: simply taking the submatrix of maximum trace will yield a k-approximation.[3] In practice, this corresponds to taking the set of maximum possible relevance, without any regard for diversity. This algorithm is purely theoretical, and does not typically yield the best results in practice. However, it does admit an easily provable approximation factor and will be useful as a starting point for our other algorithms.

Theorem 3.1. *Given the solution from Algorithm 1 as S, let L_S be a $k \times k$ submatrix of L maximizing $\text{Tr}(L_S)$. Then $f(S) \geq \frac{1}{k} f(O)$ where O is the optimal solution to the MIC objective.*

Proof. We have the following sequence of inequalities:

$$\begin{aligned} f(S) &= \sum_{i=1}^{k} \frac{\lambda_{i,S}}{1+\lambda_{i,S}} \\ &\geq \frac{\lambda_{1,S}}{1+\lambda_{1,S}} \\ &\geq \frac{\frac{1}{k}\sum_{i=1}^{k}\lambda_{i,S}}{1+\frac{1}{k}\sum_{i=1}^{k}\lambda_{i,S}} && \text{(by monotonicity of } x/(1+x)) \\ &\geq \frac{\frac{1}{k}\sum_{i=1}^{k}\lambda_{i,O}}{1+\frac{1}{k}\sum_{i=1}^{k}\lambda_{i,O}} && (\sum_{i=1}^{k}\lambda_{i,S} = \text{Tr}(S) \geq \text{Tr}(O) = \sum_{i=1}^{k}\lambda_{i,O}) \\ &\geq \frac{1}{k}\sum_{i=1}^{k}\frac{\lambda_{i,O}}{1+\lambda_{i,O}} && \text{(by Jensen's inequality)} \\ &= \frac{1}{k} f(O) && \square \end{aligned}$$

This implies a simple $O(n\log k)$ time and $O(k)$ memory algorithm for obtaining a k-approximation to the MIC objective by maintaining the top-k normed vectors in a heap.

A greedy algorithm for approximating MIC. Next, we examine the popular greedy algorithm for maximizing MIC (Algorithm 2). Although the greedy algorithm does not apply to data streams, it provides a helpful benchmark for evaluating the quality of our streaming algorithms.

Algorithm 2 Greedy(N)

1: $G \leftarrow \emptyset$
2: **while** $|G| < k$ **do**
3: $\quad e \leftarrow \text{argmax}_{u \in N} \Delta(u|G)$
4: $\quad G \leftarrow G + e$
5: **return** G

Theorem 3.2. *Greedily maximizing the MIC objective at each step achieves a $\frac{1+\lambda_{k,O}}{(1+\lambda_{1,G})(1+\lambda_{1,O+G})}$ approximation to the MIC objective, where G and O are the greedy and optimal solutions respectively.*

Remark 3.1. *It may seem strange that the approximation bound is conditional on the final greedy solution G. A looser but unconditional bound can be attained by replacing $\lambda_{1,G}$ and $\lambda_{k,G}$ with $\lambda_1(L)$ and $\lambda_n(L)$ respectively. However, in practice $\lambda_{1,G}$ and $\lambda_{k,G}$ provides much better bounds than $\lambda_1(L)$ and $\lambda_n(L)$. For example, if L was singular $\lambda_n(L) = 0$ whereas $\lambda_{k,G}$ is usually non-zero and close to $\lambda_{1,O}$ in practice.*

To prove guarantees on the greedy algorithm, we require the following lemma (proof in appendix):

Lemma 3.3. *For any $S \subset N$ and item $e \in N \setminus S$,*

$$\frac{||e||^2}{(1+\lambda_{1,S+e})(1+\lambda_{1,S})} \leq \Delta(e|S) \leq \frac{||e||^2}{(1+\lambda_{|S|+1,S+e})(1+\lambda_{|S|,S})}.$$

Another tool we'll need is the Cauchy interlacing theorem:

Theorem 3.4. *Cauchy's interlacing theorem Let L be a symmetric $n \times n$ matrix, and let $\tilde{L}$ be any $m \times m$ principal submatrix of L. Then we have the following inequalities:*

$$\lambda_1(L) \geq \lambda_1(\tilde{L}) \geq \lambda_2(L) \geq \lambda_2(\tilde{L}) \geq \cdots \geq \lambda_m(L) \geq \lambda_m(\tilde{L})$$

where the eigenvalues are sorted in decreasing order.

Now we prove the main theorem:

Proof. Let $g_1, g_2, \ldots, g_k$ be the k elements chosen during the steps of the greedy algorithm, and let $o_1, o_2, \ldots, o_k$ be the optimal elements. Let $G_i = \{g_1, g_2, \ldots, g_i\}$. Since the greedy algorithm chooses the g_i that gives the largest possible gain, we have

$$\Delta(g_{i+1}|G_i) \geq \frac{1}{k}\sum_{j=1}^{k}\Delta(o_j|G_i).$$

[3] We note that a simple k-approximation can also be obtained from the subadditive property of the MIC objective. Since the MIC objective is subadditive and monotone, the top singleton element (i.e. the vector with the longest length) along with any other $k-1$ vectors also implies a k-approximation.

By Lemma 3.3,

$$\begin{aligned}\sum_{i=1}^{k}\Delta(o_j|G_i) &\geq \sum_{j=1}^{k}\frac{||o_j||^2}{(1+\lambda_{1,G_i+o_j})(1+\lambda_{1,G_i})}\\ &\geq \sum_{j=1}^{k}\frac{||o_j||^2}{(1+\lambda_{1,G_k+O})(1+\lambda_{1,G_k})}\\ &= \sum_{j=1}^{k}\frac{\lambda_{j,O}}{(1+\lambda_{1,G_k+O})(1+\lambda_{1,G_k})}\\ &\geq \frac{1+\lambda_{k,O}}{(1+\lambda_{1,G_k+O})(1+\lambda_{1,G_k})}\sum_{j=1}^{k}\frac{\lambda_{j,O}}{1+\lambda_{j,O}}\\ &= \frac{1+\lambda_{k,O}}{(1+\lambda_{1,G_k+O})(1+\lambda_{1,G_k})}f(O)\end{aligned}$$

where the second line follows by Cauchy's interlace theorem.

Thus for each i, $f(G_i+g_{i+1})-f(G_i)\geq\frac{1}{k}\frac{1+\lambda_{k,O}}{(1+\lambda_{1,G_k+O})(1+\lambda_{1,G_k})}f(O)$. Summing over all the i's, we have the final solution G_k satisfying $f(G_k)\geq\frac{1+\lambda_{k,O}}{(1+\lambda_{1,G_k+O})(1+\lambda_{1,G_k})}f(O)$. □

We note that in the analysis, its not required to have an explicit vector representation for each item. We analyze the algorithm in this setting as it is most commonly used in recommendation applications. When the DPP kernel L is created from other metrics, the $||e||_2$ terms (Lemma 3.3) are replaced with the diagonal kernel $L_{e,e}$ corresponding to item e.

4 STREAMING ALGORITHMS FOR MAXIMIZING INDUCED CARDINALITY

In this section, we show simple streaming algorithms for approximating the MIC objective in both *insertion-only* and *sliding window* settings. The insertion-only algorithm is used as a subroutine for our sliding window algorithm. The approach we take in the construction of our algorithms shares similarities with the algorithm of Espato et al. [13], as well as other algorithms in the submodular literature. However, the analysis of the algorithm is novel and applies directly to the MIC objective. We note that since the MIC objective is not submodular, the analysis of Espato et al. does not apply in the MIC setting.

4.1 Insertion-Only Streams

In this setting, the input stream is ever growing and a new element is put into the stream without removing any of the existing elements. We analyze a simple algorithm where the result set is constructed by adding elements one at a time until it reaches the required cardinality k, see Algorithm 3. At any step it makes a simple decision to either put the current element e in the output set G or not based on the line 3 of Algorithm 3.

Although we write Algorithm 3 as an algorithm that requires the entire stream, note that we can run line 4 in an online manner, and update G whenever a new element is added to the stream. Whenever the user requires a diverse set, the current G can be returned.

Algorithm 3 relies on a parameter τ which controls how tight the approximation factor will be. Furthermore, the optimal value $f(O)$ is required as well. These parameters can be removed (Algorithm 4), given a guess $c_{\mathcal{A}}$ for the optimal value $f(O)$. As we will show below, any k-approximation to $f(O)$ suffices, such as the one given by Theorem 3.1.

Theorem 4.1. *Let $\alpha_G=\frac{1+\lambda_{k,O}}{(1+\lambda_{1,G})(1+\lambda_{1,O+G})}$. Choosing $\tau=\frac{\alpha_G}{1+\alpha_G}$ obtains an approximation ratio of $\frac{\alpha_G}{1+\alpha_G}$ for an insertion-only stream in $O(k)$ memory, where G is the solution returned by the Algorithm 3.*

Proof. The memory bounds are clear, as Algorithm 3 only ever stores the solution set G. Next, we show that setting $\tau=\alpha_G/(1+\alpha_G)$ yields an $\alpha_G/(1+\alpha_G)$ approximation.

If $|G|=k$ at the end of the algorithm, then $f(G)\geq k\frac{\tau}{k}f(O)=\tau f(O)$ by Algorithm 3. Thus we get an $\alpha_G/(1+\alpha_G)$ approximation.

If $s:=|G|<k$, then let $G_i=\{g_1,g_2,...,g_i\}$ for $i\leq s$. By line 3 of Algorithm 3, for each $o\in O-G$, there exists an i_o for which $\Delta(o|G_{i_o})\leq\tau f(O)/k$. Thus we have the following sequence of inequalities:

$$\begin{aligned}\tau f(O) &\geq \sum_{o\in O-G}\Delta(o|G_{i_o})\\ &\geq \sum_{o\in O-G}\frac{||o||^2}{(1+\lambda_{1,G_{i_o}+o})(1+\lambda_{1,G_{i_o}})}\\ &\geq \sum_{o\in O-G}\frac{||o||^2}{(1+\lambda_{1,G+o})(1+\lambda_{1,G})}\\ &= \sum_{i=1}^{|O-G|}\frac{\lambda_{i,O-G}}{(1+\lambda_{1,G+O})(1+\lambda_{1,G})}\\ &\geq \frac{1+\lambda_{|O-G|,O-G}}{(1+\lambda_{1,G+O})(1+\lambda_{1,G})}\sum_{i=1}^{|O-G|}\frac{\lambda_{i,O-G}}{1+\lambda_{i,O-G}}\\ &\geq \frac{1+\lambda_{k,O}}{(1+\lambda_{1,G+O})(1+\lambda_{1,G})}f(O-G)\\ &= \alpha_G f(O-G).\end{aligned}$$

where line 2 follows from Lemma 3.3, lines 3 and 6 follow by Cauchy's interlace theorem (Theorem 3.4), and line 4 follows from the fact that $\sum_o||o||^2=\mathrm{Tr}(L_{O-G})=\sum_i\lambda_{i,O-G}$. Thus $f(O-G)\leq\frac{\tau}{\alpha_G}f(O-G)$ when $|G|<k$.

As shown in [15], the MIC objective f is subadditive and monotone. Thus by the subadditivity and monotonicity of f, we have $f(O)\leq f(G+O)\leq f(G)+f(O-G)$. Hence $f(G)\geq(1-\tau/\alpha_G)f(O)$ when $|G|<k$. When $|G|=k$, $f(G)=\tau f(O)$. Setting $\tau=\alpha_G/(1+\alpha_G)$, we obtain $f(G)\geq\frac{\alpha_G}{1+\alpha_G}$ in both cases. □

Removing the parameter τ. As described above, choosing τ in Algorithm 3 requires knowledge of the constant α_G and the optimal objective value $f(O)$, which is inherently impossible to compute until the end of the algorithm. However, it is sufficient to enumerate over $\log k$ guesses of τ. On a high level, we know by Theorem 3.1 that

Algorithm 3 InsertOnlyBase(N,τ)

1: $G\leftarrow\emptyset$
2: **for** e in the stream N **do**
3: **if** $|G|<k$ and $\Delta(e|G)\geq\frac{\tau}{k}f(O)$ **then**
4: $G\leftarrow G+e$
5: **return** G

Algorithm 4 InsertOnly(N)

1: $c_{\mathcal{A}} \leftarrow k$-approximation of $f(O)$ by Theorem 3.1
2: **for** $i = -\lfloor \frac{2}{\varepsilon}\log k \rfloor, -\lfloor \frac{2}{\varepsilon}\log k \rfloor + 1, \ldots, \lceil \frac{2}{\varepsilon}\log k \rceil$ **do**
3: $\quad \tau_i = (1+\varepsilon)^i c_{\mathcal{A}}$
4: **for** all i in parallel **do**
5: $\quad G_i \leftarrow$ InsertOnlyBase(N, τ_i) [Algorithm 3]
6: **return** $\text{argmax}_i f(G_i)$

we can get a k-approximation to $f(O)$ quite easily at any point in the stream. Given this approximation $c_{\mathcal{A}}$, we may run Algorithm 3 with several different guesses for τ in parallel. By choosing guesses of the form $\tau_i := (1+\varepsilon)^i c_{\mathcal{A}}$ for $i \in \{-\lceil \frac{2}{\varepsilon}\log k \rceil, -\lceil \frac{2}{\varepsilon}\log k \rceil + 1, \ldots, \lceil \frac{2}{\varepsilon}\log k \rceil\}$, we can guarantee that one of the τ_i's will be within a $(1+\varepsilon)$ multiplicative factor of the true τ from the analysis above. This technique has been exploited in several other algorithms in the area of submodular optimization [2, 13, 23]. In particular, Espato et al. [13] show that these thresholds can also be initialized lazily while incurring only a logarithmic overhead in the memory.

Combining the analysis above, we get Algorithm 4 with the following guarantees:

Theorem 4.2. *Let $\alpha_G = \frac{1+\lambda_{k,O}}{(1+\lambda_{1,G})(1+\lambda_{1,O+G})}$. There exists a streaming algorithm for the MIC objective that obtains an approximation ratio of $\frac{\alpha_G}{1+\alpha_G} - \varepsilon$ for an insertion-only stream in $O(\frac{k}{\varepsilon}\log k)$ memory, where G is the final solution obtained by the Algorithm 4.*

As we will see in Section 5, Algorithm 3 is fast even when no memory requirements are imposed. Algorithm 3 runs in $O(nk^2)$ time when factoring in the linear algebra operations needed to compute the MIC objective (thus $O(k^2)$ time per update), allowing for submillisecond update times when used in practical applications.

4.2 Approximating MIC in sliding window streams

In the sliding window streaming model, a window of length n is maintained through out the stream. As a new element enters the stream, the oldest element in the window is kicked out, maintaining n elements in total. At any point in the stream, the user may query for a set of k elements that yields the maximum MIC objective.

In this regime, we obtain the following theorem (see Algorithm 5):

Theorem 4.3. *Let $\alpha_G = \frac{1+\lambda_{k,O}}{(1+\lambda_{1,G})(1+\lambda_{1,O+G})}$. There exists an algorithm with approximation ratio $\frac{\alpha_G}{1+\alpha_G}$ for a sliding window stream in $O(\sqrt{n}k\log k/\varepsilon)$ memory, where G is the final solution obtained by the Algorithm 5 and n is the length of the window.*

The input to Mantis are the elements in the stream N with $|N| \geq n$, along with a sliding window size n. The algorithm is described in Algorithm 5.

Mantis is inspired from [13], where a technique transforming insertion-only submodular optimization algorithms to sliding window ones is given. Adapting these techniques to the MIC case, we can obtain a sliding window algorithm from building upon Algorithm 3 as a subroutine.

Mantis begins by partitioning the input stream into $m := \sqrt{n}/k$ contiguous subwindows $W_1, W_2, \ldots, W_m$ of equal length $n/m = \sqrt{n}k$

Algorithm 5 Mantis(N,n)

1: $\mathcal{I} \leftarrow$ {A instance of InsertOnly with no elements inserted}
2: $\mathcal{B} \leftarrow$ empty buffer of size $s := \sqrt{n}k$
3: $t \leftarrow 0$
4: **for** e in the stream N **do**
5: $\quad$ **for** $\mathcal{A} \in \mathcal{I}$ in parallel **do**
6: $\quad\quad$ Stream e into $\mathcal{A}$
7: $\quad$ Add e to $\mathcal{B}$
8: $\quad$ **if** $\mathcal{B}$ contains m elements **then**
9: $\quad\quad \mathcal{A}_r \leftarrow$ A fresh instance of InsertOnly
10: $\quad\quad$ **for** $e_b \in \mathcal{B}$ *in reverse order* **do**
11: $\quad\quad\quad$ **if** $\mathcal{A}_r$ accepts e_b into its solution **then**
12: $\quad\quad\quad\quad$ Add e_b to $\mathcal{A}_r$
13: $\quad\quad\quad\quad$ Add a copy of $\mathcal{A}_r$ into $\mathcal{I}$
14: $\quad\quad \mathcal{B} \leftarrow$ empty buffer of size s
15: $\quad$ Remove any $\mathcal{A} \in \mathcal{I}$ with elements inserted before $t-n$
16: $\quad t \leftarrow t+1$
17: **return** $\max_{\mathcal{A} \in \mathcal{I}}$ solution returned by $\mathcal{A}$

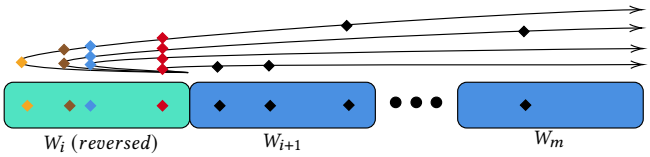

Figure 1: The stream is broken into subwindows of size $\sqrt{n}k$. When a new window $\mathcal{B} = W_i$ is streamed in, we reverse the elements of this subwindow and run k instances of InsertOnly on the reversed window and the rest of the stream $(W_i, W_{i+1}, \ldots W_m)$ in the original order. The lines and markers depicts algorithm instances added on line 13 of Algorithm 5 and their solutions respectively.

(for analysis purposes we assume m divides equally into n). At any moment in time, we keep the most recent subwindow entirely in memory, thus incurring a $O(\sqrt{n}k)$ memory cost. When the end of a subwindow is reached, we instantiate $O(k)$ instances of Algorithm 4. These instances are run on elements from the current subwindow and the future subwindows. To facilitate deletion of elements from the front of the window, one final trick is to rearrange the order of elements streamed to Algorithm 3. In particular, we show that it is enough to simply stream the elements of the current subwindow in reverse order, while keeping the original order for all future subwindows. This process is shown in Figure 1.

We defer the proof to the appendix, as much of the analysis borrows from [13].

Implementation of Mantis *(Algorithm 5).* To implement Mantis in the sliding window model, we maintain the most recent subwindow $\mathcal{B} = W_i$ during the course of the stream. Upon streaming in a new element e in the subwindow, we run lines 5–6. Whenever we reach the end of a subwindow, we run lines 10–13. When the first element of a new subwindow is streamed in, we remove the old subwindow from memory and start saving the new subwindow in $\mathcal{B}$. To handle deletion of an element e from the front of the window, we

simply remove all $\mathcal{A} \in \mathcal{I}$ whose solutions contain e. Finally, when a solution G is requested, we return the maximum valued solution from all the algorithm instances stored in $\mathcal{A}$.

5 FAST MARGINAL UPDATES FOR MIC OBJECTIVES

In this section, we sketch how to improve the update times of the marginal computation $\Delta(e|G)$ via an incremental Cholesky decomposition. This allows us to guarantee fast $O(k^2)$ time updates for all of our streaming algorithms. We note that similar techinques are utilized in Chen et al. [9] to improve algorithms for the DPP MAP objective. Unfortunately, the algorithms of Chen et al. requires $O(|N|k)$ memory, where $|N|$ is the length of the entire stream. In general, the sliding window length n could be much less than $|N|$.

Naive implementation of the MIC objective requires $O(k^3)$ time per evaluation, since one needs to either find an eigen-decomposition, or compute the inverse $(L_S+I)^{-1}$. However, since L (and any of its principal submatrices) are positive-definite, one can incrementally build a Cholesky factorization. Given a Cholesky factorization of $L_S+I=C_S C_S^T$ where C_S is lower triangular, a Cholesky factorization of L_{S+e} can be incrementally obtained in $O(k^2)$ time:

$$\begin{bmatrix} L_S+I & L_{S,e} \\ L_{S,e}^T & L_{e,e}+1 \end{bmatrix} = \begin{bmatrix} C_S & 0 \\ C_{S,e}^T & C_{e,e} \end{bmatrix} \begin{bmatrix} C_S^T & C_{S,e} \\ 0 & C_{e,e} \end{bmatrix}$$

where $L_{S,e}$ is the column vector corresponding to the column e in L with row indices in S. Thus $C_{S,e}=C_S^{-1}L_{S,e}$ and $C_{e,e}=\sqrt{1+L_{e,e}-C_{S,e}^T C_{S,e}}$, of which the former term can be computed in $O(k^2)$ time by back substitution.

Given a Cholesky decomposition of $L_{S+e}+I$, the bottom right diagonal element of $(L_{S+e}+I)^{-1}$ can be easily computed as $1/C_{e,e}^2$. Thus given the MIC objective value at S, the objective value at $S+e$ can be computed in $O(k^2)$ additional time (where the dominating factor is from the back substitution).

From the analysis above, we can derive the following update times for Algorithm 3 and Algorithm 5, for which proofs can be found in the appendix.

Theorem 5.1. *Algorithm 3 and Algorithm 5 have update times $O(k^2)$ and $O(\sqrt{n}k^2\log k/\varepsilon)$ respectively when a new element e is introduced into the stream.*

6 EXPERIMENTAL RESULTS

In this section, we show the performance of our streaming algorithms on several real-world and synthetic datasets. Our goal will be to show that these streaming algorithms are not only theoretically acceptable, but also provides good enough performance to use in production scenarios. [4]

We attempt to answer the following questions with our experiments:

- How good are the solutions returned by Greedy, MaxTrace and Mantis?
- How computationally efficient are these algorithms?

[4]The code and all data used for the experiments can be found at https://gitlab.com/paul.liu.ubc/streaming-mic-dpp

6.1 Data sources

We gathered three datasets for this experiment. Two of these datasets were analyzed by Espato et al. [13]. An additional dataset was collected by us to better analyze how our algorithm would perform in a production setting. Summary statistics regarding the datasets are shown in Table 2.

Dataset	# of Samples	Dim	Kernel
User Browsing Data	2M	768	Cosine
Geolocation Data	6.5M	2	Cosine
Yahoo! Front Page Visits	27.5M	136	Gaussian

Table 2: Summary statistics of test datasets.

User Browsing Data. In this dataset, we mined streams of user browsing events for several hundred users of a major search engine. The event streams span several months, and contains tens of thousands of timestamped events. Each event contains semantic information (such as the title of the webpage the user was on) which was embedded using a pretrained BERT model [27] into a vector v_i of dimension 768. As described in Section 2, a similarity matrix was computed from the embeddings, where the relevance value r_i was taken to be 1 for all events. This similarity matrix was taken as the DPP kernel $L = V^T V$, where V has columns corresponding to the BERT vector embeddings v_i. Here, diversity corresponds to extracting a rich set of page visits which summarize the interests of the user over the duration spanned by the sliding window.

Yahoo! Front Page Visits [1]. This dataset contains events extracted from click logs of news articles displayed on the Yahoo! front page. For each event, the user is associated with a 136-dimensional feature vector that contains information about the user such as age, gender, behavior targeting features, etc. The similarity matrix of these feature vectors (after normalization) is taken to be the DPP kernel $L=V^T V$, where V has columns corresponding to the feature vectors.. Here, diversity corresponds to discovering a wide demographic of users for downstream marketing applications.

Geolocation Data [10]. In this dataset, we use data collected by the Gowalla social network, which contains roughly 6.5 million timestamped geo-location check-ins over a period of almost two years. Each data point consists of a 2D vector of longitude and latitude values. Here, the goal is to determine a set of geo-spatially diverse locations that summarize the varying interests of Gowalla users. As with other works applying to spatial data, we use a Gaussian DPP kernel $L_{ij}=\exp(-d_{ij}^2/h^2)$, where d_{ij} is the distance between geolocation i and j, and $h=1500$ is the kernel radius. Here, diversity corresponds to discovering users with most different locations.

6.2 Experimental results

In the experiments to follow, we ran the MaxTrace (Algorithm 1), Greedy (Algorithm 2), and Mantis (Algorithm 5) algorithms on the three datasets with window size $n=5000$ and $k=10$.[5] The MaxTrace

[5]We did experiments with several values of n and k but the final deductions are same across these values. We therefore did not include those plots in this paper.

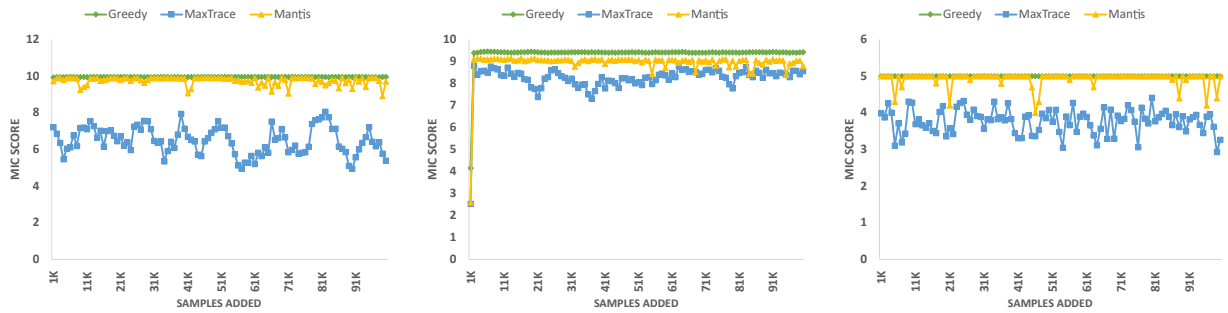

Figure 2: MIC objective values of GREEDY, MAXTRACE, and MANTIS as a function of number of samples processed. $k = 10$, Window size $n = 5000$ for (left to right) User Browsing Data, Yahoo! Front Page, and Gowalla geolocation datasets.

corresponds to selecting elements that maximize relevance regardless of diversity. The GREEDY corresponds to the industry standard of iteratively adding the next best diverse element based on the MIC objective value.

We show that the performance of MANTIS is comparable with, and often exceeds, the better of the two algorithms while using minimal memory and in fraction of time per data point.

Implementation. All the algorithms are implemented in C++. The experiments run on a dual-core commodity machine (3.60 GHz Intel Xeon W-2123 CPU with 64 GB of memory) with one thread.[6]

How good are the solutions returned by the three algorithms? We compare the performance of GREEDY, MAXTRACE, and MANTIS algorithms by applying them to the three datasets used in this paper. Since the streams are rather long, our plots are shown for a random contiguous substream of 100K data points. We then processed the data points by sliding a fixed size window over them from beginning to the end of the stream; for each window we used the three algorithms to generate the k-diverse predictions and recorded the corresponding MIC objective values. The plots show the average score over the 5 experiments.

The results of this experiment are shown in Figures 2 for $k = 10$ and $n = 5000$. A few salient points to note:

- As expected, MAXTRACE expectedly performs much worse than GREEDY and MANTIS.
- Apart from the Yahoo dataset, the average performance of GREEDY and MANTIS is comparable. This is especially surprising, given GREEDY has the entire window stored in memory.

In order to make global conclusions on the performance of different algorithms on the entire data stream, we compute the ratio of the average MIC score attained by the different pairs of the algorithms:

$$\text{MICRatio}(\mathcal{A}_1, \mathcal{A}_2) = \frac{\sum_{i=1}^{W} MIC(w_i, \mathcal{A}_1)}{\sum_{i=1}^{W} MIC(w_i, \mathcal{A}_2)},$$

where W is the total number of sliding windows in a dataset, and $MIC(w_i, \mathcal{A}_1)$ is the MIC score on the i^{th} window by algorithm $\mathcal{A}_1$. If on average $\mathcal{A}_1$ performs at par with $\mathcal{A}_2$, then MICRatio would be close to 1.

[6]We are aware of the additional speedups that multi-threading might bring in but for the sake of fair comparisons we restrict to single thread setting.

The MICRatio for different algorithms and datasets are shown in Table 3. It is clear that both GREEDY and MANTIS are much better than MAXTRACE. The MANTIS algorithm only slightly lower in MICRatio than GREEDY across all three datasets.

User Browsing Data	
$\frac{\text{MANTIS}}{\text{GREEDY}} = 0.979$	$\frac{\text{MANTIS}}{\text{MAXTRACE}} = 1.503$
Yahoo! Front Page Data	
$\frac{\text{MANTIS}}{\text{GREEDY}} = 0.955$	$\frac{\text{MANTIS}}{\text{MAXTRACE}} = 1.084$
Gowalla Geo Location Data	
$\frac{\text{MANTIS}}{\text{GREEDY}} = 0.986$	$\frac{\text{MANTIS}}{\text{MAXTRACE}} = 1.307$

Table 3: MICRatio of MANTIS with other algorithms across three datasets when $k = 10$, Window size $n = 5000$.

How efficient are these algorithms? We answer this question in terms of memory and computation time per data point comparisons across the different algorithms.

To compare the memory between the three methods, we compare the number of data points stored by each as we increase the window size n. Figures 3 shows the number of input data points stored by the three algorithms with various window sizes. For the GREEDY algorithms, the memory usage clearly increases linearly with the window size. MANTIS, on the other hand, only increases as a function of the square root of the window size.

Table 4 shows the computational speed by the three algorithms for our three data sets in terms of the average time per data point for $k = 10$ and $n = 5000$. It shows the excellent speedups achieved by MANTIS over the GREEDY approach, making it suitable for large-scale practical applications where inferences are to be made in sub-millisecond times.

7 CONCLUSION

In this paper, we proposed MANTIS, the first streaming algorithm for diversity based on the maximum induced cardinality objective. MANTIS performs at par with the often used GREEDY algorithm, with

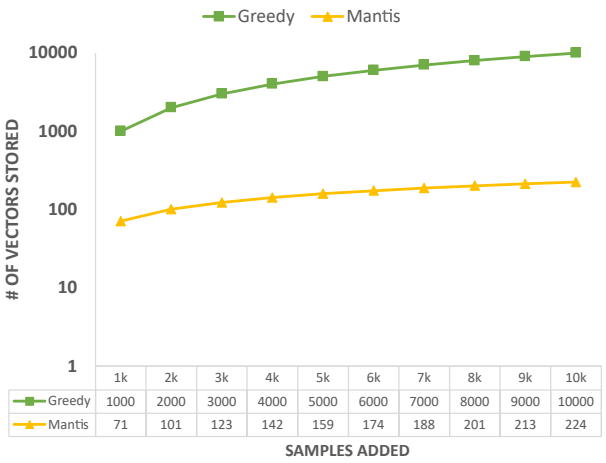

Figure 3: Comparison of the number of vector stored across various window sizes when k = 5.

	Greedy	MaxTrace	Mantis
User Browsing Data			
$k=10$	298946	59	260 [1153×]
$k=20$	1079576	246	1171 [9205×]
$k=30$	2352150	496	3108 [756×]
Yahoo! Front Page Data			
$k=10$	82647	18	65 [1271×]
$k=20$	333150	83	364 [915×]
$k=30$	707345	157	1108 [638×]
Gowalla Geo Location Data			
$k=10$	80704	15	50 [1614×]
$k=20$	288035	61	262 [1099×]
$k=30$	600665	111	676 [888×]

Table 4: Processing time (in microseconds) per element across various data sets when window size = 5000. The number within the bracket in last column shows the factor by which Mantis is faster than Greedy.

just a fraction of memory cost and multiple orders faster inference time. This makes Mantis a competitive choice for applications in large scale streaming recommendation systems.

We are already in the progress of making Mantis a core component of the recommendation pipeline for a major ad serving platform. The implementation details and the online results from these experiments will be part of a future manuscript.

A APPENDIX

Lemma 3.3. *For any $S \subset N$ and $e \in N \setminus S$,*

$$\frac{||e||^2}{(1+\lambda_{1,S+e})(1+\lambda_{1,S})} \leq \Delta(e|S) \leq \frac{||e||^2}{(1+\lambda_{|S|+1,S+e})(1+\lambda_{|S|,S})}.$$

Proof.

$$\begin{aligned}
\Delta(e|S) &= \sum_{i=1}^{|S|+1} \frac{\lambda_{i,S+e}}{1+\lambda_{i,S+e}} - \sum_{i=1}^{|S|} \frac{\lambda_{i,S}}{1+\lambda_{i,S}} \\
&= \sum_{i=1}^{|S|+1} \frac{\lambda_{i,S+e}}{1+\lambda_{i,S+e}} - \frac{\lambda_{i,S}}{1+\lambda_{i,S}} && \text{(setting } \lambda_{i,S}=0\text{)} \\
&= \sum_{i=1}^{|S|+1} \frac{\lambda_{i,S+e}-\lambda_{i,S}}{(1+\lambda_{i,S+e})(1+\lambda_{i,S})} \\
&\geq \sum_{i=1}^{|S|+1} \frac{\lambda_{i,S+e}-\lambda_{i,S}}{(1+\lambda_{1,S+e})(1+\lambda_{1,S})} \\
&= \frac{||e||^2}{(1+\lambda_{1,S+e})(1+\lambda_{1,S})}. && (\mathrm{Tr}(L_{S+e}-L_S)=||e||^2)
\end{aligned}$$

The upper bound follows by similar analysis, where on line 4 we instead lower bound the denominator by $(1+\lambda_{|S|+1,S+e})(1+\lambda_{|S|,S})$. □

Theorem 4.3. *Let $\alpha_G = \frac{1+\lambda_{k,O}}{(1+\lambda_{1,G})(1+\lambda_{1,O+G})}$. There exists an algorithm with approximation ratio $\frac{\alpha_G}{1+\alpha_G}$ for a sliding window stream in $O(\sqrt{n}k)$ memory, where G is the final solution obtained by the algorithm and n is the length of the window.*

Proof. As in Algorithm 5, let $W_1, W_2, \ldots, W_m$ be the $m = \sqrt{n}/k$ windows. For the sake of exposition, we assume m is an integer. Let W^R indicate a stream where the elements of window W are streamed in *backwards* (from earliest to latest).

First we argue correctness. Fix some subwindow W_i and consider the copies of $\mathcal{A}_r$ added to $\mathcal{I}$ on lines 10–13. Let $i_1 < i_2 < \ldots < i_s$ be the indices of the $s \leq k$ elements added to $\mathcal{A}_r$ in W_i. For each algorithm instance $\mathcal{A} \in \mathcal{I}$, let $i_{\mathcal{A}}$ be the earliest index out of elements in the solution returned by $\mathcal{A}$. Since a copy is added to $\mathcal{I}$ whenever an element is included in $\mathcal{A}_r$, there will be s algorithm instances in $\mathcal{I}$ with earliest indices $i_1, i_2, \ldots, i_s$. Label these algorithms $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_s$. Since the elements were streamed to $\mathcal{A}_r$ in reverse order, the part of the stream covered by $\mathcal{A}_{j+1}$ is exactly the elements with indices greater than i_j. Thus when the element with index i_j is deleted from the stream, line 10 guarantees that $\mathcal{A}_{j+1}$ provides a good approximation to the rest of the stream.

Next we compute the memory used. On each subwindow, we add at most k copies of $\mathcal{A}_r$ into $\mathcal{I}$, plus a fresh copy at the beginning of the stream. Thus $|\mathcal{I}| \leq km + m = O(\sqrt{n})$. The total memory cost of $\mathcal{I}$ is then $O(\sqrt{n}k\log k/\varepsilon)$, since each solution instance stores up to $k\log k/\varepsilon$ elements. Furthermore, we need to store all the elements of a subwindow, incurring a memory cost of $O(\sqrt{n}k)$. Thus the total memory cost is $O(\sqrt{n}k)$. □

Theorem 5.1. *Algorithm 3 and Algorithm 5 have update times $O(k^2)$ and $O(\sqrt{n}k^2\log k/\varepsilon)$ respectively when a new element e is introduced into the stream.*

Proof. The update time of Algorithm 3 is clear, as the marginal of each additional element costs $O(k^2)$ time to compute by Section 5.

By the proof of Theorem 4.3, theres at most $O(\sqrt{n})$ algorithm instances in $\mathcal{I}$. Each algorithm instance is takes $O(k^2\log k/\varepsilon)$ time

to update, as Algorithm 4 consists of $O(\log k/\varepsilon)$ instances of Algorithm 3. Thus the total update time of Algorithm 5 is upper bounded by $O(\sqrt{n}k^2\log k/\varepsilon)$. □

REFERENCES

[1] [n.d.]. Yahoo! Front Page Today Module User Click Log Dataset. https://webscope.sandbox.yahoo.com/.

[2] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. 2014. Streaming Submodular Maximization: Massive Data Summarization on the Fly. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. ACM, 671–680. https://doi.org/10.1145/2623330.2623637

[3] Michele Borassi, Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. 2019. Better Sliding Window Algorithms to Maximize Subadditive and Diversity Objectives. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. ACM, 254–268. https://doi.org/10.1145/3294052.3319701

[4] Victor-Emmanuel Brunel. 2018. Learning Signed Determinantal Point Processes through the Principal Minor Assignment Problem. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. 7376–7385. http://papers.nips.cc/paper/7966-learning-signed-determinantal-point-processes-through-the-principal-minor-assignment-problem

[5] L. Elisa Celis, Vijay Keswani, Damian Straszak, Amit Deshpande, Tarun Kathuria, and Nisheeth K. Vishnoi. 2018. Fair and Diverse DPP-Based Data Summarization. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsm*, Vol. 80. PMLR, 715–724. http://proceedings.mlr.press/v80/celis18a.html

[6] Badrish Chandramouli, Justin J. Levandoski, Ahmed Eldawy, and Mohamed F. Mokbel. 2011. StreamRec: a real-time recommender system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*. ACM, 1243–1246. https://doi.org/10.1145/1989323.1989465

[7] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A. Hasegawa-Johnson, and Thomas S. Huang. 2017. Streaming Recommender Systems. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*. ACM, 381–389. https://doi.org/10.1145/3038912.3052627

[8] Jiecao Chen, Huy L. Nguyen, and Qin Zhang. 2016. Submodular Maximization over Sliding Windows. *CoRR* abs / 1611.00129 (2016). arXiv:1611.00129 http://arxiv.org/abs/1611.00129

[9] Laming Chen, Guoxin Zhang, and Eric Zhou. 2018. Fast Greedy MAP Inference for Determinantal Point Process to Improve Recommendation Diversity. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. 5627–5638. http://papers.nips.cc/paper/7805-fast-greedy-map-inference-for-determinantal-point-process-to-improve-recommendation-diversity

[10] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. 2011. Friendship and Mobility: User Movement in Location-Based Social Networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11)*. Association for Computing Machinery, New York, NY, USA, 1082–1090. https://doi.org/10.1145/2020408.2020579

[11] Michal Derezinski, Daniele Calandriello, and Michal Valko. 2019. Exact sampling of determinantal point processes with sublinear time preprocessing. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 11546–11558. http://papers.nips.cc/paper/9330-exact-sampling-of-determinantal-point-processes-with-sublinear-time-preprocessing.pdf

[12] Michał Dereziński and Michael W. Mahoney. 2020. Determinantal Point Processes in Randomized Numerical Linear Algebra. *arXiv:2005.03185 [cs]* (May 2020). http://arxiv.org/abs/2005.03185 arXiv: 2005.03185.

[13] Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. 2017. Submodular Optimization Over Sliding Windows. In *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*. ACM, 421–430. https://doi.org/10.1145/3038912.3052699

[14] Mike Gartrell, Victor-Emmanuel Brunel, Elvis Dohmatob, and Syrine Krichene. 2019. Learning Nonsymmetric Determinantal Point Processes. *arXiv:1905.12962 [cs, stat]* (Dec. 2019). http://arxiv.org/abs/1905.12962 arXiv: 1905.12962.

[15] Jennifer A Gillenwater, Alex Kulesza, Sergei Vassilvitskii, and Zelda E. Mariet. 2018. Maximizing Induced Cardinality Under a Determinantal Point Process. In *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 6911–6920. http://papers.nips.cc/paper/7923-maximizing-induced-cardinality-under-a-determinantal-point-process.pdf

[16] Lei Guo, Hongzhi Yin, Qinyong Wang, Tong Chen, Alexander Zhou, and Nguyen Quoc Viet Hung. 2019. Streaming Session-based Recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. ACM, 1569–1577. https://doi.org/10.1145/3292500.3330839

[17] Marius Kaminskas and Derek Bridge. 2017. Diversity, Serendipity, Novelty, and Coverage: A Survey and Empirical Analysis of Beyond-Accuracy Objectives in Recommender Systems. *ACM Trans. Interact. Intelligent Syst.* 7, 1 (2017), 2: 1–2: 42. https://doi.org/10.1145/2926720

[18] Tarun Kathuria and Amit Deshpande. 2016. On Sampling and Greedy MAP Inference of Constrained Determinantal Point Processes. *arXiv:1607.01551 [cs, math]* (July 2016). http://arxiv.org/abs/1607.01551 arXiv: 1607.01551.

[19] Alex Kulesza and Ben Taskar. 2011. k-DPPs: Fixed-Size Determinantal Point Processes. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*. Omnipress, 1193–1200. https://icml.cc/2011/papers/611_icmlpaper.pdf

[20] Alex Kulesza and Ben Taskar. 2011. Learning Determinantal Point Processes. In *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*. AUAI Press, 419–427. https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2252&proceeding_id=27

[21] Alex Kulesza and Ben Taskar. 2012. Determinantal Point Processes for Machine Learning. *Found. Trends Mach. Learn.* 5, 2-3 (2012), 123–286. https://doi.org/10.1561/2200000044

[22] Chengtao Li, Stefanie Jegelka, and Suvrit Sra. 2016. Fast DPP Sampling for Nystrom with Application to Kernel Methods. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016 (JMLR Workshop and Conference Proceedings)*, Vol. 48. JMLR.org, 2061–2070. http://proceedings.mlr.press/v48/lih16.html

[23] Paul Liu and Jan Vondrák. 2019. Submodular Optimization in the MapReduce Model. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*. 18:1–18:10. https://doi.org/10.4230/OASIcs.SOSA.2019.18

[24] Sepideh Mahabadi, Piotr Indyk, Shayan Oveis Gharan, and Alireza Rezaei. 2019. Composable Core-sets for Determinant Maximization: A Simple Near-Optimal Algorithm. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research)*, Vol. 97. PMLR, 4254–4263. http://proceedings.mlr.press/v97/mahabadi19a.html

[25] Barry Smyth and Paul McClave. 2001. Similarity vs. Diversity. In *Case-Based Reasoning Research and Development, 4th International Conference on Case-Based Reasoning, ICCBR 2001, Vancouver, BC, Canada, July 30 - August 2, 2001, Proceedings (Lecture Notes in Computer Science)*, Vol. 2080. Springer, 347–361. https://doi.org/10.1007/3-540-44593-5_25

[26] Karthik Subbian, Charu C. Aggarwal, and Kshiteesh Hegde. 2016. Recommendations For Streaming Data. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*. ACM, 2185–2190. https://doi.org/10.1145/2983323.2983663

[27] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-Read Students Learn Better: On the Importance of Pre-training Compact Models. *arXiv preprint arXiv:1908.08962v2* (2019).

[28] Weiqing Wang, Hongzhi Yin, Zi Huang, Qinyong Wang, Xingzhong Du, and Quoc Viet Hung Nguyen. 2018. Streaming Ranking Based Recommender Systems. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*. ACM, 525–534. https://doi.org/10.1145/3209978.3210016

[29] Yichao Wang, Xiangyu Zhang, Zhirong Liu, Zhenhua Dong, Xinhua Feng, Ruiming Tang, and Xiuqiang He. 2020. Personalized Re-ranking for Improving Diversity in Live Recommender Systems. *CoRR* abs/2004.06390 (2020). arXiv:2004.06390 https://arxiv.org/abs/2004.06390

[30] Mark Wilhelm, Ajith Ramanathan, Alexander Bonomo, Sagar Jain, Ed H. Chi, and Jennifer Gillenwater. 2018. Practical Diversified Recommendations on YouTube with Determinantal Point Processes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. ACM, Torino Italy, 2165–2173. https://doi.org/10.1145/3269206.3272018

[31] Cai - Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. 2005. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*. ACM, 22–32. https://doi.org/10.1145/1060745.1060754