# Interactive Audience Expansion On Large Scale Online Visitor Data

Gromit Yeuk-Yin Chan
gromit.chan@nyu.edu
New York University
New York, USA

Tung Mai
tumai@adobe.com
Adobe Research
San Jose, USA

Anup B. Rao
anuprao@adobe.com
Adobe Research
San Jose, USA

Ryan A. Rossi
ryrossi@adobe.com
Adobe Research
San Jose, USA

Fan Du
fdu@adobe.com
Adobe Research
San Jose, USA

Cláudio T. Silva
csilva@nyu.edu
New York University
New York, USA

Juliana Freire
juliana.freire@nyu.edu
New York University
New York, USA

## ABSTRACT

Online marketing platforms often store millions of website visitors' behavior as a large sparse matrix with rows as visitors and columns as behavior. These platforms allow marketers to conduct Audience Expansion, a technique to identify new audiences with similar behavior to the original target audiences. In this paper, we propose a method to achieve interactive Audience Expansion from millions of visitor data efficiently. Unlike other methods that undergo significant computations upon inputs, our approach provides interactive responses when a marketer inputs the target audiences and similarity measures. The idea is to apply data summarization technique on the large visitor matrix to obtain a small set of summaries representing the similarities in the matrix. We propose efficient algorithms to compute the data summaries on a distributed computing environment (i.e., Spark) and conduct the expansion using the summaries. Our experiment shows that our approach (1) provides 10× more accurate and 27× faster Audience Expansion results on real datasets and (2) achieves a 98% speed-up compared to straightforward data summarization implementations. We also present an interface to apply the algorithm for real-world scenarios.

## CCS CONCEPTS

• **Computing methodologies** → *Parallel algorithms.*

## KEYWORDS

Interactive audience expansion; Look-alike modeling

**ACM Reference Format:**
Gromit Yeuk-Yin Chan, Tung Mai, Anup B. Rao, Ryan A. Rossi, Fan Du, Cláudio T. Silva, and Juliana Freire. 2021. Interactive Audience Expansion On Large Scale Online Visitor Data. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467179

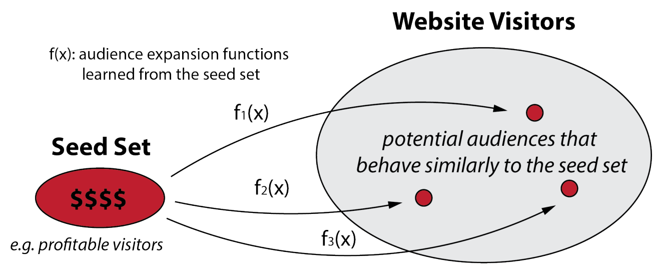

Website Visitors
f(x): audience expansion functions learned from the seed set
f1(x)
f2(x)
f3(x)
Seed Set
$$$$
e.g. profitable visitors
potential audiences that behave similarly to the seed set

**Figure 1: Overview of Audience Expansion.**

## 1 INTRODUCTION

The online advertising industry is driven by the ability to recommend the right ads to the right audience. Marketers often seek to target users similar to their existing users to maximize the likelihood of positive responses to their ads. Audience expansion, also called look-alike modeling, is the task of finding users that are similar to a given set of *seed users*(Figure 1). The seed users can be, for example, purchasers of a product, web subscribers, or loyal customers. In general, the seed set is usually much smaller than the entire set of users. An audience expansion model gives the marketers a set of *look-alike users* that are highly relevant to those seed users, often concerning their Internet activities. The identified look-alike audiences are then targeted in ad campaigns.

Audience expansion provides marketers with a convenient and useful tool to target their users. Yet its usability can further be improved with interactivity. In online advertising, users are often represented by thousands of features. However, marketers are usually interested in using only a small set of those features due to privacy or domain expertise. Moreover, they might not know the final chosen features, and even the seed users, until assessing with several combinations. Interactivity, therefore, is an essential and fundamental aspect of audience expansion. Even though various techniques have been proposed for identifying the look-alike users [4, 5, 21, 30, 33, 38, 42], not much work has been done on *interactive* audience expansion with user defined seed sets and features.

In this paper, we only consider binary features, as continuous features can be bucketed into multiple binary features if needed. Each binary feature is also called a *trait*. Hence, audience expansion is given a large, often very sparse, user-by-trait binary matrix $R$ at the preprocessing step. Each row of $R$ corresponds to a user (also called visitor), and each column corresponds to a trait. At query time, a set of rows is chosen as the seed set, and a set of columns is chosen as relevant traits. The chosen set of traits determines how the algorithm measures similarity among the users and determines how the look-alike set is computed.

To preprocess the data for interactively audience expansion, the model partitions $R$ into multiple summaries using the minimum description length principle. Each summary can be viewed as a rank-1 matrix determined by a list of users and a list of traits. The description length principle minimizes the representations of all summaries (model description) and the extra representation needed to exactly recover $R$ from those summaries (correction). At query time, each summary is restricted to the chosen traits. Then any two summaries will be merged if they overlap significantly by rows or by columns. The final summary representations are used to score each user. Specifically, a user has a high score in the model if their traits have high Jaccard similarity with one of the summaries' columns.

This paper makes the following key contributions:
(1) A novel binary matrix summarization method for interactive audience expansion. The method uses MDL to find the best partitions while utilizing hashing strategies to be both fast and scalable for interactive audience expansion on large-scale online visitor data.
(2) A highly scalable parallel implementation of the approach for the interactive audience expansion problem.
(3) Comprehensive experiments demonstrate our approach's effectiveness as it achieves a significant speed-up with accuracy comparable to the state-of-the-art. This naturally enables the approach for real-time interactive audience expansion via an interactive user interface with visualizations.

## 2 RELATED WORK

One of the important online advertisement and marketing tasks is audience expansion (or look-alike modeling) [10, 12, 15, 31, 36]. Given a set of seed users, the goal of the audience expansion task is to find more users (audiences) that are similar to the seed set, and therefore useful to target a campaign around as they are likely to eventually convert (e.g., purchase an advertised product) [30, 47]. Many recent methods have been proposed to solve this problem [4, 5, 21, 30, 33, 38, 42]. Some work uses clustering [38] including graph-based nearest neighbor methods [5], matrix factorization [21], and classification-based methods [4]. There has also been some work that extracts rules/feature vectors and uses them to score [30, 33, 42]. However, none of this work can solve the *interactive audience expansion* problem introduced in our paper.

Other recent work has focused on the mobile marketing audience expansion problem [50]. In that work, they propose a two-stage approach called Hubble [50], where a model is retrained weekly in the offline stage, and another lightweight model is trained in the online stage after every single audience expansion request. However, the online model for a single campaign request takes several minutes to train, as opposed to several seconds required by our interactive audience expansion problem. Furthermore, we also propose a distributed implementation of our approach that is efficient and scalable for the preprocessing step.

There has recently been some work on scalable distributed methods for look-alike modeling [29]. While there is a lot of recent work on parallel clustering algorithms for shared-memory and distributed architectures [3, 6, 8, 14, 17, 48], there are only a few such works that focus on distributed methods for look-alike modeling [29]. This work focused on proposing a distributed implementation of an existing look-alike modeling approach. In our work, we describe a distributed implementation of our approach for *interactive* audience expansion using Spark. This is mainly useful for the preprocessing step as it is offline and doesn't have the same interactive response requirements of the main online interactive audience expansion phase. There has been some recent work on interactive data analysis and visualization [19, 22, 23]. The majority of work in this area has focused mainly on interaction techniques and visual interfaces for exploring data [16, 20, 27, 41] or finding the best clusters by interactively tuning the parameters of a clustering algorithm [7, 26, 46]. In this work, we design fast and scalable methods for the *interactive audience expansion* problem that can naturally support the interactivity requirements needed for such large-scale datasets.

There have also been many edge clustering approaches proposed in the last few years on applications in visualization [11, 49], bioinformatics [44], network science [1] and social media [45]. Edge clustering algorithms have been proposed for bipartite [18, 40, 45] and general graphs [1]. Techniques have also been proposed for different variants of the edge clustering problem, including hierarchical [49] and overlapping edge clustering [24, 43]. Co-clustering of bipartite graphs representing words and documents have also been extensively studied [13]. However, none of these works can be used directly for the interactive audience expansion problem.

## 3 AUDIENCE EXPANSION OVERVIEW

In this section, we describe audience expansion and formulate the process based on the online visitors' traits data. First, online visitor behavior is stored as a binary matrix $R \subseteq U \times T$ where $U$ is a set of visitors $u$ and $T$ is a set of user behavior $t$ (i.e., traits). To begin the audience expansion process, users input both (1) a subset of online visitors $s \subseteq U$ as target customers (seed set) and (2) a subset of user behavior $b \subseteq T$. Given these inputs, the objective is to compute a score $\alpha \rightarrow [0, 1]$ for each visitor in the dataset that quantifies its similarity to $s$ based on $b$.

In the real life scenario, seed customers often exhibit multiple representative behavior (e.g. profitable customers might come from different regions). These behavior are sets of traits that appear together from the seed customers. Thus, to reflect the similarities between the visitors and each representative behavior, each visitor $u_i$ has a set of scores where each score $\alpha_{ij}$ corresponds to a representative behavior $r_j$, which is a subset of relations $\hat{r} \subseteq s \times b$. Under these settings, the score can be based on set similarity measures such as Jaccard similarity:

$$\alpha_{ij} = Jaccard(u_i, b_{r_j}) \tag{1}$$

where $b_{r_j} = \{t \in T : \exists u \in U \text{ such that } (u, t) \in r_j\}$. For each customer $u_i$, we can label them as an expanded target if there is at least one score that is higher than a threshold (e.g., $max(\alpha_{i,:}) \geq 0.9$) As a result, users can obtain two insights from the score of each visitor: how similar the visitor is to a set of representative behavior and how many seed customers contain this behavior. The higher the score, the better the affinity the visitor is to the seed customers.

For interactive audience expansion, our challenge is to arrive at the representative behavior $r$ within seconds of time. Previous methods' computation speed often relies on the number of traits or visitors in the dataset. We will demonstrate that by having a way to derive representative behavior without the sizes of visitors and traits as constraints, we can provide a magnitude of speed up with high-quality audience expansion results (Section 5).

## 4 DATA STRUCTURE FOR INTERACTIVE AUDIENCE EXPANSION

While the scoring of each visitor is straightforward and efficient (i.e., calculating set similarity), the challenge lies in the discovery of representative behavior from the seed customers in interactive time. Since the similarity measure based on $b$ is defined interactively from the users, straightforward clustering cannot be applied to group visitors by their similarity based on a custom set of traits. Yet, without compressing the visitor data into a smaller representation, it is not tangible to find the representative behavior from the seed population interactively.

### 4.1 Data Summaries For Audience Expansion

To address both (1) user-defined seed population and similarity measure and (2) interactive representative behavior discovery, our idea is to decompose the binary matrix of visitor data into a much smaller set of summaries. These summaries are submatrices of the binary matrix such that in each summary matrix, the rows and columns are homogeneous (i.e., the row and column vectors inside the summary matrix are similar) Also, the relations (i.e., the "1"s in the binary matrix) are partitioned among these summaries so that the union of relations of the summaries represents the original visitor data. These summaries represent similar behavior based on all traits among all visitors, and we can induce the seed populations' by these summaries instead of searching the whole binary matrix.

To illustrate how the summary works, given an example visitor data $R$, where each row is a visitor and each column is a trait:

$$R = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

For example, the matrix is decomposed into three summaries as the colored partitions $p_1$, $p_2$, and $p_3$. Suppose a user wants to define the first two rows (i.e. $r_1, r_2$) as the seed population and the first four columns (i.e. $c_1, c_2, c_3, c_4$) as the destined user behavior. Using the summaries, we can undergo three steps to acquire a set of representative behavior (Algorithm 1), which is as follows:

(1) For each summary, we first filter the rows and columns inside by the seed population and user behavior. For example, $p_2$ will have its rows $r_3$ and $r_4$ removed and $p_3$ will become empty. Then, we remove all the empty summaries.

(2) Then, for each filtered summary, we find if it overlaps greatly by rows or by columns with other summaries. For example, $p_1$ and $p_2$ overlap greatly by rows relative to their sizes after the filtering. Therefore, we assign the union of their columns to these summaries. Notice now both partitions contain $r_1, r_2$ and $c_1, c_2, c_3, c_4$, which compose the representative behavior in the visitor data under the user settings.

**ALGORITHM 1:** Extract Representative Behavior

```
Input  : P      – a list of data summaries
         s, b   – seed population and visitor behavior
         $\epsilon$      – overlap threshold
Output : $\hat{R}$      – a list of representative behavior
for p in P do                                                  /* Step (1) */
    p.rows ← p.rows ∩ s
    p.cols ← p.cols ∩ b
    if p.rows = ∅ or p.cols = ∅ then
        P.remove(p)
    end
end
for $p_i$ in P do                                                 /* Step (2) */
    for $p_j$ in P \ {$p_i$} do
        if $\frac{|p_i.rows \cap p_j.rows|}{|p_i.rows|} \geq \epsilon$ then
            $p_i$.cols ← $p_i$.cols ∪ $p_j$.cols
        end
        if $\frac{|p_i.cols \cap p_j.cols|}{|p_i.cols|} \geq \epsilon$ then
            $p_i$.rows ← $p_i$.rows ∪ $p_j$.rows
        end
    end
end
$\hat{R}$ ← single_linkage_clustering( P, distance="Jaccard")     /* Step (3) */
```

(3) Lastly, since the summaries after the assignment of additional rows and columns might be similar to each other, we remove the duplicated summaries by clustering them based on their Jaccard similarity of relations. For example, $p_1$ and $p_2$ will be grouped together since they become identical after step (1) and (2). An example of a clustering technique for this purpose is hierarchical clustering with single linkage. The remaining summaries will be the representative behavior $r$ for user scoring in Equation 1.

In the algorithm, it is clear that the search for representative behavior only involves the summaries but not the whole number of visitors and traits, which drastically reduce the number of comparisons. Therefore, our next question now is how to compute these summaries from the large scale visitor data to achieve such interactive discovery. Overall, these summaries should be *compact* (Equation 3) such that the rows and columns inside are homogeneous. Else, the representations of the partition cannot be used to convey accurate representative behavior from the visitor data. Secondly, the number of summaries should also be drastically smaller than the size of the original matrix so that interactive audience expansion can be done (Algorithm 1).

### 4.2 Problem Formulation

To enable interactive discovery of representative behavior from an arbitrary set of seed customers, we define the problem as a graph summarization problem to compress the binary matrix into a much smaller set of summary structure. The goal is to decompose the matrix into disjoint groups of entries (i.e. *summaries*) so that the rectangular intersections of rows and columns within a group are homogeneous (e.g. $p_1$, $p_2$, and $p_3$). Intuitively, within each group of entries $p$, the behavior of rows are similar no matter which subset of columns is removed. Thus, each summary can derive a column representation to represent every row within the group faithfully. Thus, after the input of $s$ and $b$, the seed set's supports and behavior can be derived from the sizes and column representations of the summaries, respectively, without the need to go through each entry in the matrix. Formally, our graph summarization problem is:

- **Given:** bipartite relations $R \subseteq U \times T$
- **Find:** a partition of relations $\{p_1, p_2, ..., p_k\} = R$
- **to Minimize:** number of partitions $k$
- **Subject to:** $Compactness(p, t_i) \geq \epsilon$ for all $p$ in any column $t_i$

**Partition of Relations** Each partition $p_i$ contains a disjoint set of relations ("1"s in the binary matrix). These relations also form a set of rows $u_i$ and columns $t_i$ To represent all rows in the partition, a column representation $r_{p_i}$ can be defined as:

$$r_{p_i} = \left\{ t : t \in t_{p_i} \text{ and } \frac{|u_{p_i} \cap R_t|}{|u_{p_i}|} \geq 0.5 \right\} \tag{2}$$

where $R_t$ is the set of users having trait $t$.

Intuitively, a column will exist in the representation only when more than half of the rows have a relation with it. We can use $r_{p_i}$ to approximately represent the rows in the partition in the audience expansion process.

**Compactness** It indicates how similar the rows' entries are regarding to column $t_i$ within the same partition. This is important as the decrease of similarity among the rows inside a partition after removing columns depends on the compactness. Since we want to identify the column's value of each row without going through each of them, the representation of a column of a partition should be homogeneous to the rows inside. The compactness of a column $t_i$ within a partition $p$ is defined as:

$$Compactness(p, t_i) = 1 - H_b \left( \frac{\mathbb{1}_p(t_i)}{|u_p|} \right) \tag{3}$$

where $H_b$ is the binary entropy function and $\mathbb{1}_p(t_i)$ is the number of rows that contains $t_i$ in $p$. Intuitively, if the majority of rows in the partition are either "1" or "0" on column $t_i$, the compactness will tend to one. Otherwise, it will tend to zero if it is half-and-half. Overall, it indicates the quality of the partition.

**Minimizing the number of partitions** Since our goal is to find the rectangular intersections of relations in the matrix such that rows and columns inside are homogeneous, we would like the intersections to be as large as possible which correspond significant website visitors' patterns. Therefore, one way to achieve it is to minimize the number of partitions. The fewer the number of partitions, the more likely large partitions will be formed.

## 4.3 Proposed Method

To address the problem in Section 4.2, we propose a scalable and effective algorithm based on the minimum description length (MDL) principle [39] to find the ideal partition for our summarization problem. We leverage the MDL principle to determine a cost function for the best partition of relations. Using the cost function, we propose a randomized greedy search to determine the best rows partition, then further extend the search to find the relation partition.

The MDL principle states that the best model (i.e., partition) of a dataset should minimize the total description length $L$, which is the sum of two quantities: model description length and the data description length with the help of the model:

$$L = L(M) + L(D|M)$$

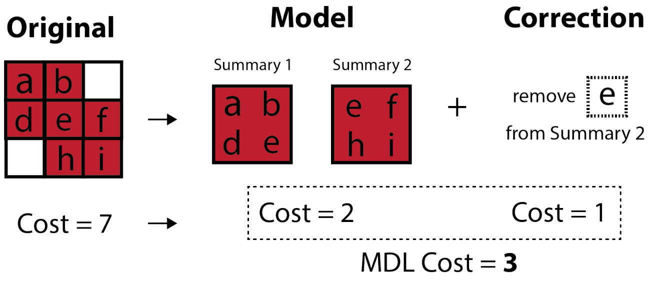


**Figure 2: Illustration of Minimum Description Length (MDL) Principle to find relation partitions from the dataset. A binary matrix can be represented as a set of summaries with corrections. The original matrix requires seven units, while the model and correction representations only require 3.**

The first part $L(M)$ can be seen as the size of the relation partitions. The second part $L(D|M)$ can be seen as the number of corrections in each partition's representation to represent the original data. To make both the sizes of relation partitions and corrections comparable, the description length of a set of relation partitions $P$ is

$$L(P) = \|P\| + \left\| (\cup_{(p \in P)} r_p \times u_p) \oplus R \right\| \tag{4}$$

where $r_p$ is the representation of a partition defined in Equation 2, $u_p$ are the visitors in the partition, and $\oplus$ is the disjunctive union between sets. The difference between the original matrix and the union of the relations reconstructed from the representations and the visitors of the partitions denotes the corrections. The relationships between the MDL formulation and our problem statement is illustrated by Figure 2. By balancing the number of partitions and the corrections as the partitions' quality, we can determine optimal groupings as data summaries for finding representative behavior.

## 4.4 Search Method for Building Summaries

As we define the cost function to seek the optimal partition of relations in Equation 4, we now propose a greedy algorithm to construct the partitions based on the function and the constraints in the problem statement. We apply a bottom-up approach which is based on popular graph summarization algorithms [9, 25, 28, 34]. The main difference is that while most of the summarization techniques focus on summarizing nodes (i.e., rows and columns in our case), our goal is to summarize the edges (i.e., the relations in the matrix). We first provide an overview of the algorithm, and then we describe each step in detail.

*4.4.1 Overview.* Given an input binary matrix R, we are looking for the relation partitions in which each partition contains a subset of rows and columns. Since we only require the relations to be disjoint across different partitions, each partition can share common rows or columns. Therefore, there are $2^{U \times T}$ numerous combinations in the search space to look for the solution. To narrow down the search process, our algorithm (Algorithm 2 in the Appendix) breaks down the search problem into two main steps:

(1) **Row Partitioning (line 6)**: Instead of partitioning the relations directly, the algorithm first attempts to find the optimal row partitions that minimize the cost function. There are two phases involved:

(a) **Generating Candidate Sets**: To effectively compare feasible rows to group together to reduce the cost function, the algorithm first divides the rows in the matrix into a set of candidates. The rows in each candidate set should share similar sets of columns which is likely to decrease the cost function.
(b) **Bottom Up Merging**: In each candidate set, each row starts as a row partition and we repeatedly merge two partitions whose merger decreases the cost most until there are no mergers that can decrease the cost function within a partition.

(2) **Extracting Non-compact Columns (line 7-9)**: Until the previous step, for each row partition, there might be columns that do not satisfy the compactness constraint in Equation 3. We extract the relations from these columns and row partitions and form a new binary matrix $R_t$ at iteration $t$. The new matrix will go through step (1) again, and the whole algorithm will finish until all relations' partitions satisfy the compactness constraint.

*4.4.2 Generating Candidates for Merging.* The goal of partitioning the rows in the matrix before the merger is to allow the merging step to be efficient and accurate at the same time. To allocate the rows in the binary matrix into different candidate sets for merging, we use *hashing* strategies to divide the rows. The objective is to design a scalar function $h(x)$ so that $h(u_i) = h(u_j)$ when two rows $u_i$ and $u_j$ should be compared for a merger. One important motivation to use hashing is to enable parallelization among different merging operations. In practice, to handle millions of visitor's information, we use distributing computing platform (i.e. Spark) to compute the summaries. In distributed computing architecture, data are distributed across different partitions. All partitions execute the tasks concurrently until there is a need to re-partition the data (i.e. *shuffle*). Therefore, to utilize resources, we need to ensure all mergers take similar computation time. We will discuss more in Section 4.5 for designing the hashing strategies. Here, we first describe the goals for hashing the rows in Spark.

**G.1** *Similar Jaccard Similarities.* For the cost function (Equation 4), the greater the Jaccard similarity among the rows in a summary, the more similar each row will be with the representation which also implies a better likelihood to minimize the cost. Thus, we want rows with high Jaccard similarity to be grouped in the same candidate sets.

**G.2** *Evenly sized candidate sets.* The bottom-up approach discussed next to merge the rows has a quadratic time complexity in each candidate set. Therefore, it is particularly important to make sure a balanced distribution of rows among the partitions in Spark. Otherwise, many data partitions will be left idling and need to wait for the skewed ones to finish.

*4.4.3 Bottom Up Merging.* In each candidate set, we are given a subset of rows from the input matrix in each iteration (line 4). We are interested in finding the partition of rows such that the cost function (Equation 4) is minimized within the candidate set. We use a bottom-up approach to merge the rows (Algorithm 3 in the Appendix). The algorithm performs the following operations:

(1) It starts with all rows belonging to an individual partition and all partitions being the candidates for merging(line 2). For this initial partition C, we compute the cost (line 3).
(2) Among the candidates, we randomly pick a row partition $c_0$. We try to merge it with the rest of the candidates and calculate the new cost (line 9). We keep the best candidate with the minimum cost after merging (line 10-13).
(3) We compare the minimum cost after merging with the cost without any merging. If there is a decrease, we merge $c_0$ with the best candidate and update the current cost (line 16-17). Otherwise, we put $c_0$ into the final result (line 19). In both cases, the candidate set C has one fewer element.
(4) We continue steps 2-3 until there are no more candidates.

The bottom-up algorithm's complexity is $O(U^2T)$, where $U$ is the number of rows, and $T$ is the number of columns in the input matrix. In the beginning, the candidate list has $U$ items. The list removes an item after each iteration. Suppose there are $U_t$ items in the list and in each iteration $t$, the candidate $c_0$ needs to compare with $U_t - 1$ items. Therefore, there are $\frac{U(U+1)}{2}$ comparisons. Lastly, for each comparison, the row partitions need to combine their columns to compute the new cost, which requires $O(T)$ scans of columns. The sizes of candidate sets are crucial for the performance in terms of algorithmic complexity and parallelization in distributed computing, which we will discuss more in Section 4.5.

*4.4.4 Extracting Non-compact Columns.* After the row partitions are created, the goal is to increase the compactness of these partitions so that they become available for interactive representative behavior search in Algorithm 1. Recalling Equation 3, in each row partition (which can also be treated as a relation partition), it contains columns with different compactness values that depend on the relations. Therefore, we can divide each partition's relations based on whether they belong to columns that satisfy a compactness threshold (line 7-8 in Algorithm 2). If a column is compact, its relations stay in the partition and are removed from the input matrix. Else, the relations are extracted and returned to the input matrix. The resulted matrix and relation partitions are pushed to the next iteration and result, respectively.

## 4.5 Speed Up Using Hashing Strategies

Mentioned in Section 4.4.2, the goals of hashing the rows is to ensure similar rows to be grouped in the same partition (**G.1**) and ensure even data distribution among partitions in Spark (**G.2**). We propose two hashing techniques to cope with these goals and discuss how to combine them to acquire both accuracy and efficiency.

*4.5.1 Overview of hashing strategies.* The overview of applying hash strategies to speed up the merging process can be seen in Algorithm 4 in the Appendix and Figure 3. Overall, the algorithm generates a candidate set based on the row partitions for bottom up merging (Algorithm 3) for a number of iterations. Within each iteration, it first groups the input row partitions by *MinHash LSH* (line 2) based on their representations' similarities (**G.1**). Then, within each group, the algorithm performs a two-stage merger by first dividing the partitions randomly (line 4) for a more even data distribution across different merging operations (**G.2**). After merging is done within each random partition, the results from these partitions will be merged again inside each LSH partition.

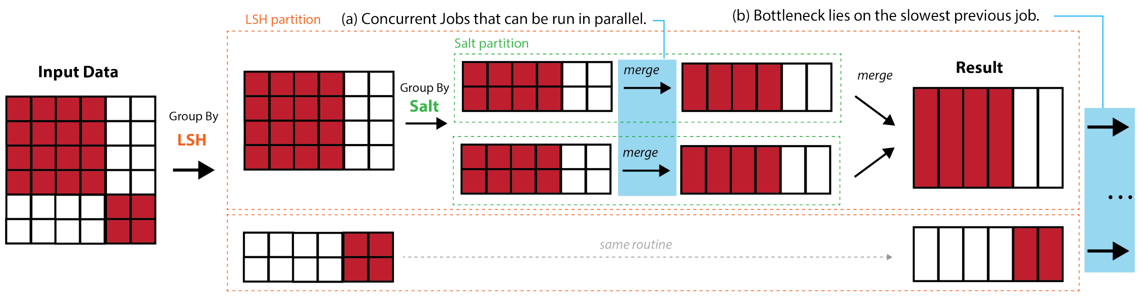

LSH partition
(a) Concurrent Jobs that can be run in parallel.
(b) Bottleneck lies on the slowest previous job.
Salt partition
Input Data
Group By
LSH
Group By
Salt
merge
merge
merge
Result
same routine
...

**Figure 3: Overview of hashing strategies to generate candidate sets for merging operations (Algorithm 3). First, the rows are partitioned by Locality Sensitive Hashing (LSH) that allocates similar rows to different partitions. Then, a salted key is generated to further partition the rows inside a LSH partition. The merging operation is done first on the rows in each salted partition then again on the results from the partitions within a LSH partition. Noted that all merging operations among different partitions can be run concurrently in Spark.**

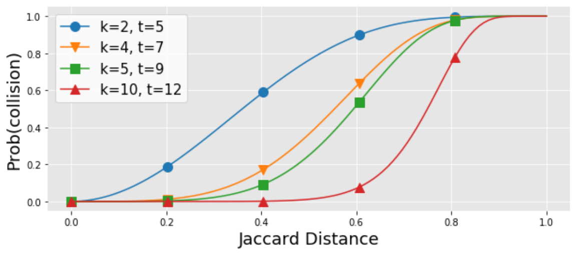


**Figure 4: Theoretical probability of two rows being allocated in the same candidate set at least once with different number of hash functions $k$ and iterations $t$. Different parameters can result in similar collision probability but the run time might differ significantly.**

*4.5.2 MinHash LSH.* To allocate rows whose Jaccard similarities exceed a predefined threshold, we can use MinHash, which performs a random projection of high dimensional data into lower dimensional space, and then LSH, which hashes similar items into the same hash table (i.e. buckets) with high probability.

**MinHash signature generation.** For each row $u_i$ (i.e. visitor), its MinHash signature is a vector of MinHash values from $n$ independent hash functions $[h_1(u_i), h_2(u_i), ..., h_n(u_i)]$ so that the probability $p$ of a MinHash collision between the rows is the same with their similarities (i.e. $sim(u_i, u_j) = Pr[h_k(u_i) = h_k(u_j)]$). Therefore, by increasing the number of hash functions $k$, the collision probability of the hash signature decreases to $p^k$ [37].

**Controlling the Probability of Comparisons Among Rows.** The purpose of iterations (line 1) and LSH partition (line 2) in Alg. 4 are to ensure that similar rows can be encountered within the same candidate set at least once, while disallowing non-similar rows to be grouped in the same partition. Formally, given $k$ hash functions and $t$ iterations, the probability that two rows with Jaccard similarity $s$ collide at least once (i.e. a successful comparison) is:

$$P[s] = 1 - (1 - s^k)^t \tag{5}$$

For example, if we want rows with similarity more than 0.7 to be allocated in the same candidate set with probability higher than 0.95, we can produce a LSH table with 2 MinHash functions ($k = 2$) and merge the rows with 5 iterations ($t = 5$). The probability of a successful comparison in terms of Jaccard similarity has a form as an S-curve (Figure 4). Figure 4 illustrates that several success curves are almost identical under different parameter settings. In general, the decrease of collision probability by increasing the number of hash functions can be compensated by the increase of iterations. However, since the increase of hash functions also decreases the expected sizes of candidate sets, it might reduce the number of comparisons in the merging process and increase the parallelism of multiple merging operations, which can lead to significant speed up in the summarization (Section 5.2).

*4.5.3 Salted Hash.* It is a simple hash function where each row is assigned a random integer. It aims at dividing similar rows when the candidate set becomes huge after LSH partition.

**Two Stage Summarization Using Salted Hashes.** For salted hashes, its usage lies on further dividing the rows in the candidate set in multiple subsets. The intuition is that within a candidate set, all rows should be similar to each other. Thus, it is just a matter of time for many of them to be merged together. Therefore, during the bottom-up merging routine, we can choose to compare a small subset inside the candidate set first, and then we can further merge the row partitions formed from the subsets. In Figure 3(b), we illustrate that in a distributed computing environment, since similar jobs (i.e. bottom up merging among candidate sets) needs to finish together in order to begin the next operation, the bottom neck of the summarization depends on the largest candidate set from the LSH partitions. In practice, many online visitors might exhibit similar behavior. Thus, it is not uncommon to have skewed distributions of rows among the LSH partitions. Salted hashes, therefore, provides a solution to dramatically reduce the number of candidates in each merging process (Figure 3(a)). We further illustrate that such strategy can result in magnitudes of speed up in the evaluation.

## 5 EVALUATIONS

In this section, our goal is to demonstrate:

(1) Our audience expansion method achieves at most around 10× better accuracies and interactive computing time (all within seconds and 27× faster) on real datasets.
(2) Our summarization pipeline contributes to meaningful performance improvement.
(3) The interactive audience expansion enables a new visual analytics system on real commercial datasets. Our market experts can identify meaningful potential target customers using the features provided by the system.

For dataset details and experimental setup see Appendix B-C.

Table 1: **The table shows the accuracy in terms of precision, recall, and F-score as well as run time using our *summary* method vs. some benchmark approaches of interactive audience expansion. The *improvement* and *speed up* are calculated against the best results from the benchmark approaches.**

| | | | Accuracy | | | | Run Time | |
|---|---|---|---|---|---|---|---|---|
| **Dataset** | Label | Methods | Precision | Recall | F-score | Improvement | Time (s) | Speed Up |
| Retail Rocket | Transaction (Item 1) | One Pass Logistic Regression | 0.0612 | 0.6 | 0.111 | - | 137 | - |
| | | K Means | 0.000235 | 0.933 | 0.00047 | - | 27.5 | - |
| | | Graph | 0.229 | 0.733 | 0.349 | - | 68.7 | - |
| | | **Summary (Ours)** | **0.4** | **0.533** | **0.457** | **1.31×** | **2.87** | **9.56×** |
| | Transaction (Item 2) | One Pass Logistic Regression | 0.0123 | 0.256 | 0.0235 | - | 395 | - |
| | | K Means | 0.000560 | 0.907 | 0.00112 | - | 60.4 | - |
| | | Graph | 1 | 0.0232 | 0.0455 | - | 88.7 | - |
| | | **Summary (Ours)** | **0.324** | **0.721** | **0.457** | **9.82×** | **2.87** | **11.6×** |
| | Transaction (Item 3) | One Pass Logistic Regression | 0.0104 | 0.423 | 0.020 | - | 941 | - |
| | | K Means | 8.85e-5 | 0.346 | 0.000177 | - | 112 | - |
| | | Graph | 0.25 | 0.0385 | 0.0667 | - | 105 | - |
| | | **Summary (Ours)** | **0.167** | **0.461** | **0.245** | **3.67×** | **2.87** | **27.1×** |
| AAM | Purchases (Cosmetics) | One Pass Logistic Regression | 0.563 | 0.655 | 0.605 | - | 15.5 | - |
| | | K Means | 0.00117 | 0.948 | 0.00234 | - | 1.58 | - |
| | | Graph | 0.140 | 0.353 | 0.200 | - | 44.1 | - |
| | | **Summary (Ours)** | **0.722** | **0.707** | **0.714** | **1.17×** | **0.661** | **2.40×** |
| | Purchases (Electronics) | One Pass Logistic Regression | 0.796 | 0.992 | 0.883 | - | 17.38 | - |
| | | K Means | 0.00301 | 1 | 0.006 | - | 1.70 | - |
| | | Graph | 0.442 | 0.798 | 0.569 | - | 50.6 | - |
| | | **Summary (Ours)** | **0.916** | **0.802** | **0.855** | **0.968×** | **0.958** | **1.77×** |

## 5.1 Accuracy

This section evaluates the accuracy through comparisons against alternatives using ground truth labels and assesses the trade-offs between summaries' quality and core pipeline parameters. We use the F-score over the positive class (i.e., expanded audiences) to measure audience expansion's accuracy with ground truth labels.

**Benchmark Methods for Interactive Audience Expansion.** We describe different approaches that can suit our purposes of *interactive* audience expansion. We referenced from the survey related to audience expansion [2] and neglect methods that require more than hours to return results. These methods all target at predicting visitors as for potential targets for advertisement, and they are, in theory, able to provide results within a reasonable time:

(1) **One Pass Logistic Regression**: A straightforward approach to predict potential audiences for advertisement is to train a binary classifier using the seed customers and some samples of general visitors. For interactivity purposes, we use the One Pass Logistic Regression (OLR) [35], an efficient approach to perform logistic regression on a dataset, as the benchmark for the machine learning approach for audience expansion.

(2) **K Means Clustering**: Another approach to address the problem is to cluster the data [38]. The clustering approach works by first cluster the seed customers and remaining dataset separately. Then, for the remaining dataset, each data is classified as the expanded audience if it is closer to a cluster from the seed set than the clusters from the dataset. Among all clustering methods, k means clustering is the most efficient one. We choose the number of clusters $k$ (1-10) that reaches the best accuracy and converges within 300 iterations.

(3) **User to User Similarity Graph**. The graph-based approach for audience expansion is to select potential customers through a user to user similarity graph [32]. The potential customers are queried if they are similar to the seed customers. For these potential customers, they are scored with a scoring function derived from the seed customers. LSH is used to connect the users by their approximated similarities.

**Accuracy and Online Running Time.** The accuracies and run time on the real datasets are shown in Table 1. Overall, our summary audience expansion method achieves better F-scores (at most 10×) than other approaches and significantly faster computations (at most 27×). The good accuracy may come from two reasons. First, given a set of seed customers, our method results in multiple score functions that capture different representative behavior, which avoids "averaging" the discriminative features between the seed customers and the total population. Such benefit avoids the algorithm from being over-selective (i.e. using LSH to select extremely small samples that results in a low recall in Graph) or not distinguishing the seed set from the whole dataset (i.e. low precision in the K Means). Second, the summary captures the important attributes (i.e., traits) from a high dimensional space of online visitor behavior, which avoids underfitting like logistics regression when constructing a meaningful score function or classifier. For the run time, our approach's time complexity is based on the number of data summaries, which is drastically smaller than the number of rows and columns in the original matrix. It allows us to return results within seconds from millions of visitors or traits in the datasets.

**Effect on LSH parameters.** To demonstrate that the accuracy of audience expansion is not sensitive to the parameter choice, we measure the accuracy of `AAM` audience expansion with varied

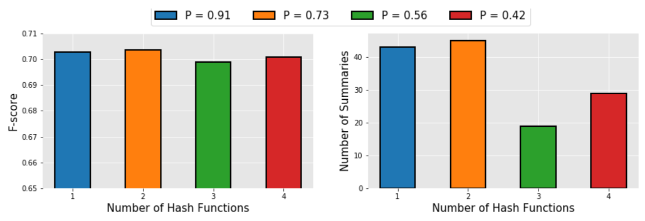


**Figure 5: The effect of increasing number of hash functions in terms of accuracies (left) and number of summaries resulted (right). The labels show the theoretical collision probabilities for items with similarity = 0.7. Also, all summaries provide audience expansion results within 1 second.**

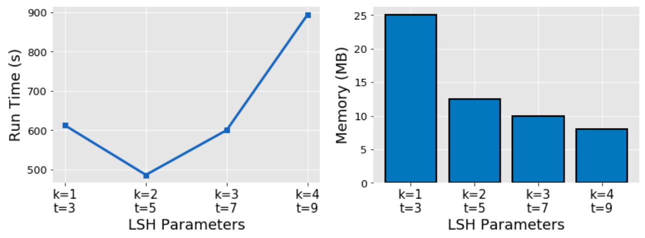


**Figure 6: Run time (left) and memory usage (right) with different numbers of hash functions ($k$) and iterations ($t$).**

collision probabilities ($P[s]$ in Equation (5)). In Figure 5, the number of data summaries is small when the number of hash functions is high since the more similar rows are allocated in the candidate set, the higher the quality the greedy merging algorithm will result. Still, the accuracy does not differ much since the representative behavior can still be acquired efficiently in Algorithm 1.

## 5.2 Scalability

In this section, we evaluate the effect of different LSH parameters to show that there is an optimal setup between the sizes and numbers of batch processing. Also, we demonstrate an end-to-end run time improvement using our hashing strategies and compression capability of our summarization algorithm.

**LSH parameters.** We report the effect of LSH parameters on run time and memory performance (Figure 6). We increase the number of hash functions and iterations together to maintain similar collision probability. For memory usage, increasing the number of hash functions reduces memory usage since reducing collision probability within an iteration can reduce the number of rows in a candidate set. However, the run time might not improve since there exists an overhead for each merge operation. Each time the rows are relocated to a different candidate set, it results in a *shuffle* in the distributed environment, where the data are transferred to different partitions, and a network cost occurs.

**End-to-end evaluation.** We report the run time breakdown of the merging operation (Algorithm 3) with the LSH and salting optimizations. We sample 10,000 visitor data from AAM dataset to demonstrate the run time improvement. Figure 7 shows the cumulative run time after applying each hashing strategy. In general, the hashing strategy allows the merging operation to scale well and remove 98% of the time. It shows that by partitioning the data and ensuring an even distribution of data in Spark, the whole merging operation achieves a much better concurrency and efficiency.

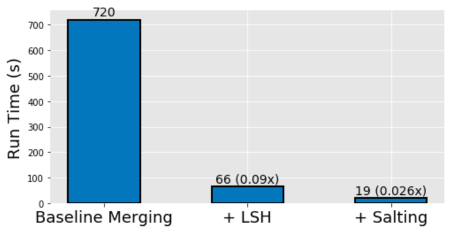


**Figure 7: Effect of LSH and Salting strategies.**

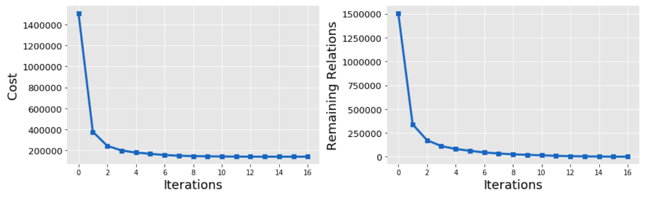


**Figure 8: Cost reduction throughout iterations (Algo. 2).**

**Cost Reduction and Summarization Over Iterations of Row Partitioning.** We demonstrate the effects of relation summarization through repeatedly summarizing the rows in the dataset (Algorithm 2). Figure 8 shows a “waterfall” structure of cost reduction after each iteration of summarization. Most relation summaries are quickly retrieved at the beginning of the iterations and rows remained for the merging operation are drastically reduced afterward. This provides intuition behind our cost function (Equation 4), as well as the effectiveness of the summarization strategy.

## 5.3 Use Case

Our highly interactive audience expansion algorithm opens new opportunities for developing interactive audience management systems at scale. For digital marketing, we implement a system to help marketers choose online visitors for advertisement (Figure 9).

For qualitative analysis of our system’s usefulness, we demonstrated our system to explore a real dataset on 2.3 million online visitors. First, market analysts would like to launch advertisements to online visitors who are likely to be video editors to purchase the company’s video editing software. Therefore, they initialized the seed set as the visitors who used the company’s video editing software (Figure 9(a)). Also, they removed the traits related to behavior about using the company’s photo editing software (Figure 9(b)) since online users typically had already purchased both software as a bundle. The algorithm then returned a list of line charts (Figure 9(c)), where each line chart represented the score distribution from a representative behavior. The line chart values were sorted so that audiences with scores greater than a threshold will be queried by brushing the line. By trying different thresholds and exploring the selected audience’s behavior, our analysts identify an interesting group of online visitors who searched for sample images on the company’s search engine website (Figure 9(d)). Thus, the marketers decided to launch advertisement about video editing software on the image search engine website, as video producers were likely to search for online images for their video productions.

Overall, our analytics system allows marketers to interactively discover potential profitable online visitors and identify the key attributes of these populations, providing new experience on audience management systems and commercial innovations.

## 6 CONCLUSION

This paper introduces a novel interactive audience expansion algorithm for a large matrix of online visitor behavior. Our approaches mainly focus on creating data summaries that can retrieve representative behavior among online visitors within seconds. While a straightforward graph summarization approaches can construct the summaries, we provide an efficient hashing strategy that significantly speeds up the computation in a distributed computing environment. The data summaries also provide a significant improvement in audience expansion accuracy and online run time on real commercial datasets. To demonstrate the interactive experience, we proposed an interactive system for utilizing our algorithm in business decision making. We believe that these results will inspire data scientist to design interactive recommendation systems for industry scale online visitor activities.

## ACKNOWLEDGMENTS

This work was supported in part by: NSF awards CNS-1229185, CCF-1533564, CNS-1544753, CNS-1730396, CNS-1828576, CNS-1626098; and the NVIDIA NVAIL at NYU. We also thank Nvidia Corporation for donating GPUs used in this research.

## REFERENCES

[1] Yong-Yeol Ahn, James P Bagrow, and Sune Lehmann. 2010. Link communities reveal multiscale complexity in networks. *Nature* 466, 7307 (2010), 761–764.

[2] Sihem Amer-Yahia. 2018. Human Factors in Data Science. In *ICDE*. 1–12.

[3] Domenica Arlia and Massimo Coppola. 2001. Experiments in parallel clustering with DBSCAN. In *ECPP*. 326–331.

[4] Abraham Bagherjeiran, Andrew Hatch, Adwait Ratnaparkhi, and Rajesh Parekh. 2010. Large-scale customized models for advertisers. In *ICDMW*. 1029–1036.

[5] Ashish Bindra, Srinivasulu Pokuri, Krishna Uppala, and Ankur Teredesai. 2012. Distributed big advertiser data mining. In *ICDMW*. 914–914.

[6] Christian Böhm, Robert Noll, Claudia Plant, and Bianca Wackersreuther. 2009. Density-based clustering using graphics processors. In *CIKM*. 661–670.

[7] Marco Cavallo and Çağatay Demiralp. 2018. Clustrophile 2: guided visual clustering analysis. *TVCG* 25, 1 (2018), 267–276.

[8] Gromit Yeuk-Yin Chan, Fan Du, Ryan A. Rossi, Anup B. Rao, Eunyee Koh, Cláudio T. Silva, and Juliana Freire. 2020. Real-Time Clustering for Large Sparse Online Visitor Data. In *WWW*. 1049âĂŞ1059.

[9] Gromit Yeuk-Yin Chan, Panpan Xu, Zeng Dai, and Liu Ren. 2018. ViBr: Visualizing Bipartite Relations at Scale with the Minimum Description Length Principle. *TVCG* 25, 1 (2018), 321–330.

[10] Randell Cotta, Mingyang Hu, Dan Jiang, and Peizhou Liao. 2019. Off-Policy Evaluation of Probabilistic Identity Data in Lookalike Modeling. In *WSDM*. 483–491.

[11] Weiwei Cui, Hong Zhou, Huamin Qu, Pak Chung Wong, and Xiaoming Li. 2008. Geometry-based edge clustering for graph visualization. *TVCG* 14, 6 (2008), 1277–1284.

[12] Stephanie deWet and Jiafan Ou. 2019. Finding Users Who Act Alike: Transfer Learning for Expanding Advertiser Audiences. In *KDD*. 2251–2259.

[13] Inderjit S Dhillon. 2001. Co-clustering documents and words using bipartite spectral graph partitioning. In *KDD*. 269–274.

[14] Inderjit S Dhillon and Dharmendra S Modha. 2002. A data-clustering algorithm on distributed memory multiprocessors. In *Large-scale parallel data mining*. 245–260.

[15] Khoa D Doan, Pranjul Yadav, and Chandan K Reddy. 2019. Adversarial factorization autoencoder for look-alike modeling. In *CIKM*. 2803–2812.

[16] Fan Du, Catherine Plaisant, Neil Spring, and Ben Shneiderman. 2018. Visual interfaces for recommendation systems: Finding similar and dissimilar peers. *TIST* 10, 1 (2018), 9.

[17] D Foti, D Lipari, Clara Pizzuti, and Domenico Talia. 2000. Scalable parallel clustering for data mining on multicomputers. In *IPDPS*. Springer, 390–398.

[18] Lichuan Gu, Yueyue Han, Chao Wang, Wei Chen, Jun Jiao, and Xiaohui Yuan. 2019. Module overlapping structure detection in PPI using an improved link similarity-based Markov clustering algorithm. *Neural Comp. and App.* 31, 5 (2019), 1481–1490.

[19] Jeffrey Heer and Maneesh Agrawala. 2008. Design considerations for collaborative visual analytics. *Information visualization* 7, 1 (2008), 49–62.

[20] Dong Hyun Jeong, Caroline Ziemkiewicz, Brian Fisher, William Ribarsky, and Remco Chang. 2009. iPCA: An Interactive System for PCA-based Visual Analytics. In *Computer Graphics Forum*, Vol. 28. 767–774.

[21] Bhargav Kanagal, Amr Ahmed, Sandeep Pandey, Vanja Josifovski, Lluis Garcia-Pueyo, and Jeff Yuan. 2013. Focused matrix factorization for audience selection in display advertising. In *ICDE*. 386–397.

[22] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. 2008. Visual analytics: Definition, process, and challenges. In *Information visualization*. Springer, 154–175.

[23] Daniel A Keim, Florian Mansmann, Jörn Schneidewind, Jim Thomas, and Hartmut Ziegler. 2008. Visual analytics: Scope and challenges. In *Visual data mining*. 76–90.

[24] Paul Kim and Sangwook Kim. 2015. Detecting overlapping and hierarchical communities in complex network using interaction-based edge clustering. *Physica A: Stat. Mech. App.* 417 (2015), 46–56.

[25] Jihoon Ko, Yunbum Kook, and Kijung Shin. 2020. Incremental Lossless Graph Summarization. In *KDD*. 317–327.

[26] Bum Chul Kwon, Ben Eysenbach, Janu Verma, Kenney Ng, Christopher De Filippi, Walter F Stewart, and Adam Perer. 2017. Clustervision: Visual supervision of unsupervised clustering. *TVCG* 24, 1 (2017), 142–151.

[27] Hanseung Lee, Jaeyeon Kihm, Jaegul Choo, John Stasko, and Haesun Park. 2012. iVisClustering: An interactive visual document clustering via topic modeling. In *Computer graphics forum*, Vol. 31. 1155–1164.

[28] Kyuhan Lee, Hyeonsoo Jo, Jihoon Ko, Sungsu Lim, and Kijung Shin. 2020. SSumM: Sparse Summarization of Massive Graphs. *arXiv:2006.01060* (2020).

[29] Gavin Li, Jaebong Kim, and Andy Feng. 2013. Yahoo audience expansion: migration from hadoop streaming to spark. *Proc.of the Spark Summit* (2013).

[30] Haishan Liu, David Pardoe, Kun Liu, Manoj Thakur, Frank Cao, and Chongzhe Li. 2016. Audience expansion for online social network advertising. In *KDD*. 165–174.

[31] Qiang Ma, Eeshan Wagh, Jiayi Wen, Zhen Xia, Robert Ormandi, and Datong Chen. 2016. Score Look-Alike Audiences. In *ICDMW*. 647–654.

[32] Qiang Ma, Musen Wen, Zhen Xia, and Datong Chen. 2016. A Sub-linear, Massive-scale Look-alike Audience Extension System A Massive-scale Look-alike Audience Extension. In *Big Data, Streams, & Heterogeneous Mining Workshop*. 51–67.

[33] Ashish Mangalampalli, Adwait Ratnaparkhi, Andrew O Hatch, Abraham Bagherjeiran, Rajesh Parekh, and Vikram Pudi. 2011. A feature-pair-based associative classification approach to look-alike modeling for conversion-oriented user-targeting in tail campaigns. In *WWW*. 85–86.

[34] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. 2008. Graph summarization with bounded error. In *SIGMOD*. 419–432.

[35] Vu Nguyen, Tu Dinh Nguyen, Trung Le, Svetha Venkatesh, and Dinh Phung. 2016. One-pass logistic regression for label-drift and large-scale classification on distributed systems. In *ICDM*. 1113–1118.

[36] Artem Popov and Daria Iakovleva. 2018. Adaptive look-alike targeting in social networks advertising. *Procedia Computer Science* 136 (2018), 255–264.

[37] Anand Rajaraman and Jeffrey David Ullman. 2011. *Mining of massive datasets*.

[38] Archana Ramesh, Ankur Teredesai, Ashish Bindra, Sreenivasulu Pokuri, and Krishna Uppala. 2013. Audience segment expansion using distributed in-database k-means clustering. In *Workshop on Data Mining for Online Advertising*. 1–9.

[39] Jorma Rissanen. 1978. Modeling by shortest data description. *Automatica* 14, 5 (1978), 465–471.

[40] Michael T Schaub and Santiago Segarra. 2018. Flow smoothing and denoising: graph signal processing in the edge-space. In *GlobalSIP*. IEEE, 735–739.

[41] Tobias Schreck, Jürgen Bernard, Tatiana Von Landesberger, and Jörn Kohlhammer. 2009. Visual cluster analysis of trajectory data with interactive kohonen maps. *Information Visualization* 8, 1 (2009), 14–29.

[42] Jianqiang Shen, Sahin Cem Geyik, and Ali Dasdan. 2015. Effective audience extension in online advertising. In *KDD*. 2099–2108.

[43] Chuan Shi, Yanan Cai, Di Fu, Yuxiao Dong, and Bin Wu. 2013. A link clustering based overlapping community detection algorithm. *DKE* 87 (2013), 394–404.

[44] Ryan W Solava, Ryan P Michaels, and Tijana Milenković. 2012. Graphlet-based edge clustering reveals pathogen-interacting proteins. *Bioinfo.* 28, 18 (2012), i480–i486.

[45] Xufei Wang, Lei Tang, Huiji Gao, and Huan Liu. 2010. Discovering overlapping groups in social media. In *ICDM*. 569–578.

[46] John Wenskovitch, Ian Crandell, Naren Ramakrishnan, Leanna House, and Chris North. 2017. Towards a systematic combination of dimension reduction and clustering in visual analytics. *TVCG* 24, 1 (2017), 131–141.

[47] Weinan Zhang, Lingxi Chen, and Jun Wang. 2016. Implicit Look-Alike Modelling in Display Ads. In *ECIR*. 589–601.

[48] Weizhong Zhao, Huifang Ma, and Qing He. 2009. Parallel k-means clustering based on mapreduce. In *CLOUD*. 674–679.

[49] Hong Zhou, Xiaoru Yuan, Weiwei Cui, Huamin Qu, and Baoquan Chen. 2008. Energy-based hierarchical edge clustering of graphs. In *Pac. Vis. Sym.* 55–61.

[50] Chenyi Zhuang, Ziqi Liu, Zhiqiang Zhang, Yize Tan, Zhengwei Wu, Zhining Liu, Jianping Wei, Jinjie Gu, Guannan Zhang, Jun Zhou, et al. 2020. Hubble: An Industrial System for Audience Expansion in Mobile Marketing. In *KDD*. 2455–2463.

## Appendix

## A DETAILED ALGORITHMS

**ALGORITHM 2:** Relation Partitioning

```
Input : R  – input binary matrix
        c  – threshold of compactness
Output: P  – a list of relation partitions
M ← R = $U \times T$;
P ← [];
while M is not empty do
    $S \leftarrow \{\{u\} \forall u \in U_M\}$;
    t ← 1;
    // Section 4.5
    $P_S$ ← hash_and_merge($S$);
    // Section 4.4.4
    remove the relations in M that satisfy the compactness constraint in $P_S$;
    remove the relations in $P_S$ that do not satisfy the compactness constraint;
    P ← P + $P_S$;
end
```

**ALGORITHM 3:** Merging Rows in a Candidate Set

```
Input : R  – input binary matrix as a list of rows
Output: $P_{rows}$ – a list of row partitions
$P_{rows}$ ← [];
C ← $\{\{r\} | r \in R\}$;
current_cost ← cost(C);
while C is not empty do
    $c_0$ ← random_pop(C);
    $c_{best}$ ← undefined;
    best_cost ← infinity;
    foreach $c \in C$ do
        cost ← cost($C \cup \{c_0 \cup c\} \setminus \{c_0, c\}$);
        if cost < best_cost then
            $c_{best}$ ← $c$;
            best_cost ← cost;
        end
    end
    if $c_{best}$ < current_cost then
        $c_{best}$ ← $c_{best} \cup c_0$;
        current_cost ← best_cost;
    else
        $P_{rows}$.push ($c_0$);
    end
end
```

## B DATASET DETAILS

To evaluate scalability and accuracy, we experiment on two online commercial datasets. The first dataset `Retailrocket` is a public dataset which has 1,407,580 visitors' behavior logs. These logs consist of an action (*view*, *add to cart*, and *transaction*) on one of the 235,061 products. Therefore, there are 705,183 unique traits. The second dataset `AAM` is an internal dataset from our CRM platform, which has 178,349 visitors and 3,755 traits. The traits consist of visitors' logs and their demographics. For evaluating the accuracies of audience expansion, we label the visitors by whether they contain the trait that indicates purchasing behavior (e.g., *transaction* of an item in `Retailrocket`), which translates the problem into a binary classification. Then, we split the data into a training and testing set. Our objective is to see by learning the representative behavior from the seed set that contains the label, how is the accuracy of the score function in retrieving the visitors with the purchasing behavior in the test set. Notice that this is an extremely imbalanced classification problem. Among millions of visitors, only hundreds of them will lead to purchases.

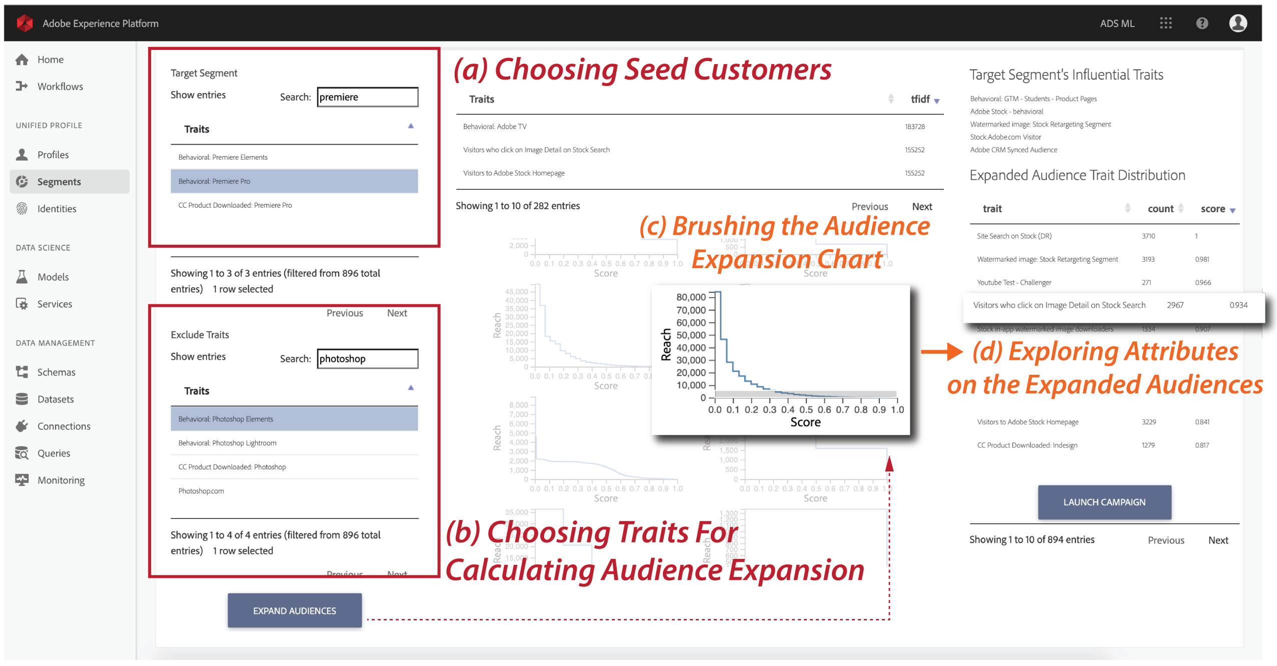
## C EXPERIMENTAL SETUP

For constructing the summaries, we report results on a Hadoop cluster with ten nodes, and each node has four AMD Opteron 6276 2.3 GHz CPUs with 256GB of RAM. We report runtime averages from multiple trials. Without any specifications, our LSH hash tables contain two bands with 1 MinHash signature in each band (i.e., two hash tables), and we have five iterations in the row merging operations. Also, we set all thresholds (i.e. *compactness* and *overlapping*) to 0.9. For the audience expansion using the summaries, we select visitors as the expanded audiences when they have at least one score greater than 0.9 (i.e., $max(\alpha_i) \geq 0.9$). The expansions are run in a MacBook Pro with 2.4 GHz 8-Core Intel Core i9 CPUs and 32GB RAM.

## D HASHING ALGORITHM

**ALGORITHM 4:** Overview of Hashing Strategies

```
Input : $S$  – input candidate set
        $T$  – number of iterations
Output: $P_S$ – a list of relation partitions
for $t = 0$; $t < T$; $t$++; do
    partition $S$ to $S_{LSH}$ by MinHash LSH;
    foreach $s_{LSH}$ in $S_{LSH}$ do
        partition $s_{LSH}$ to $s_{salt}$ by salted hashes;
        foreach $s$ in $s_{salt}$ do
            $s$ ← merge_rows($s$);
        end
        $s_{LSH}$ ← merge_rows($\cup s_{salt}$);
    end
    $S$ ← $\cup S_{LSH}$;
end
$P_S$ ← $S$;
```


Adobe Experience Platform
ADS ML
Home
Workflows
UNIFIED PROFILE
Profiles
Segments
Identities
DATA SCIENCE
Models
Services
DATA MANAGEMENT
Schemas
Datasets
Connections
Queries
Monitoring
(a) Choosing Seed Customers
(b) Choosing Traits For Calculating Audience Expansion
(c) Brushing the Audience Expansion Chart
(d) Exploring Attributes on the Expanded Audiences
Reach
Score
EXPAND AUDIENCES
LAUNCH CAMPAIGN

Target Segment

Show entries Search: premiere

| Traits |
|---|
| Behavioral: Premiere Elements |
| Behavioral: Premiere Pro |
| CC Product Downloaded: Premiere Pro |

Showing 1 to 3 of 3 entries (filtered from 896 total entries) 1 row selected

Previous Next

Exclude Traits

Show entries Search: photoshop

| Traits |
|---|
| Behavioral: Photoshop Elements |
| Behavioral: Photoshop Lightroom |
| CC Product Downloaded: Photoshop |
| Photoshop.com |

Showing 1 to 4 of 4 entries (filtered from 896 total entries) 1 row selected

Previous Next

| Traits | tfidf |
|---|---|
| Behavioral: Adobe TV | 183728 |
| Visitors who click on Image Detail on Stock Search | 155252 |
| Visitors to Adobe Stock Homepage | 155252 |

Showing 1 to 10 of 282 entries Previous Next

Target Segment's Influential Traits

Behavioral: GTM - Students - Product Pages
Adobe Stock - behavioral
Watermarked image: Stock Retargeting Segment
Stock.Adobe.com Visitor
Adobe CRM Synced Audience

Expanded Audience Trait Distribution

| trait | count | score |
|---|---|---|
| Site Search on Stock (DR) | 3710 | 1 |
| Watermarked image: Stock Retargeting Segment | 3193 | 0.981 |
| Youtube Test - Challenger | 271 | 0.966 |
| Visitors who click on Image Detail on Stock Search | 2967 | 0.934 |
| Stock in-app watermarked image downloaders | 1334 | 0.907 |
| Visitors to Adobe Stock Homepage | 3229 | 0.841 |
| CC Product Downloaded: Indesign | 1279 | 0.817 |

Showing 1 to 10 of 894 entries Previous Next

**Figure 9: An Audience Expansion interface that provides (a) choices of seed customers through defining segments (b) options of traits to be included for scoring functions (c) audience expansion results as an interactive line chart (d) attributes overview of a selection of expanded audiences.**