

Article

Generalized Sketches for Streaming Sets

Wenhua Guo ¹, Kaixuan Ye ^{2,*}, Yiyan Qi ^{3,*}, Peng Jia ³ and Pinghui Wang ³
¹ State Key Laboratory for Manufacturing Systems Engineering, Xi'an Jiaotong University, Xi'an 710049, China; markguo@xjtu.edu.cn

² School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China

³ MOE Key Laboratory for Intelligent Networks and Network Security, Xi'an Jiaotong University, Xi'an 710049, China; pengjia@sei.xjtu.edu.cn (P.J.); phwang@mail.xjtu.edu.cn (P.W.)

* Correspondence: yekaixuan@stu.xjtu.edu.cn (K.Y.); qiyiyan@stu.xjtu.edu.cn (Y.Q.); Tel.: +86-15906871991 (K.Y.)

Abstract: Many real-world datasets are given as a stream of user–interest pairs, where a user–interest pair represents a link from a user (e.g., a network host) to an interest (e.g., a website), and may appear more than once in the stream. Monitoring and mining statistics, including cardinality, intersection cardinality, and Jaccard similarity of users' interest sets on high-speed streams, are widely employed by applications such as network anomaly detection. Although estimating set cardinality, set intersection cardinality, and set Jaccard similarity, respectively, is well studied, there is no effective method that provides a one-shot solution for estimating all these three statistics. To solve the above challenge, we develop a novel framework, *SimCar*. *SimCar* online builds an *order-hashing* (OH) sketch for each user occurring in the data stream of interest. At any time of interest, one can query the cardinalities, intersection cardinalities, and Jaccard similarities of users' interest sets. Specially, using OH sketches, we develop maximum likelihood estimation (MLE) methods to estimate cardinalities and intersection cardinalities of users' interest sets. In addition, we use OH sketches to estimate Jaccard similarities of users' interest sets and build locality-sensitive hashing tables to search for users with similar interests with sub-linear time. We evaluate the performance of our methods on real-world datasets. The experimental results demonstrate the superiority of our methods.

Keywords: streaming algorithms; sketch; Jaccard similarity; cardinality



Citation: Guo, W.; Ye, K.; Qi, Y.; Jia, P.; Wang, P. Generalized Sketches for Streaming Sets. *Appl. Sci.* **2022**, *12*, 7362. <https://doi.org/10.3390/app12157362>

Academic Editor: Radu Godina

Received: 2 June 2022

Accepted: 16 July 2022

Published: 22 July 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Many real-world networks (e.g., computer networks, phone networks, and financial networks) generate data (e.g., packets, calling records, and transactions) in a stream fashion, where an element (e.g., a packet) records a link from a user (e.g., a source IP address) to an interest (e.g., a destination IP address). The data can be modeled as a data stream of user–interest pairs, including duplicates, because multiple data records (e.g., packets from a source IP address to a destination IP address) may relate to the same user–interest pair. Due to the large size and high-speed nature of these data streams, it is prohibitive to collect all the data, especially when the computation and memory resources of data-collection devices (e.g., network routers) are limited. To solve this challenge, considerable attention has been paid to designing fast and memory-efficient data streaming algorithms for monitoring and mining user behaviors. For example, Count-Min sketch [1] and its variants have been successfully used to detect heavy hitters (e.g., user–interest pairs frequently occurred).

In addition to statistics using the frequency of duplicates, significant effort has also been made to mine *users' interest sets*, such as:

- **User cardinality.** A user's cardinality is defined to be the number of *distinct* interests that the user links to, i.e., the cardinality of the user's interest set. Monitoring user cardinalities in computer networks is fundamental for applications such as network anomaly detection [2]. One can use a variety of sketch methods, such as LPC [3],

LogLog [4], HyperLogLog [5], MinCount [6], RoughEstimator [7], and their variants [2,8–13] to generate a compact data summary (or sketch) for each user's interest set, and then infer user cardinalities from the generated sketches.

- **Common-interest count.** Two users' common-interest count is defined as the number of interests that they both link to, i.e., the cardinality of the intersection of two users' interest sets. It is popularly used for applications such as friend recommendation [14] and network anomaly detection [15]. One can use sketch and sampling methods in [13–15] to estimate users' common-interest counts.
- **User Jaccard similarity.** The Jaccard similarity is a popular similarity measure. Two users' similarity can be evaluated as the Jaccard similarity of their interest sets, which is defined as their common-interest count divided by the number of distinct interests at least one of them links to. One can use sketch methods such as *MinHash* [16], *OPH* [17], and their variants [18–22] to estimate users' Jaccard similarities. Some of these sketch methods, such as Minhash, can be used for *locality-sensitive hashing* (LSH) indexing [16,23–26], which is capable of searching a query user's similar users with sub-linear time.

The above three *duplicate-irrelevant* statistics may all be desired for applications such as network anomaly detection. For example, given a detected abnormal user (e.g., the IP address of an attacker captured by Honeynet systems), network administrators may want to quickly search for similar users among all network users, which may also be abnormal users, and then estimate their cardinalities, Jaccard similarities, and common-interest counts for further inspection. Although many cardinality estimation methods can be easily extended to estimate the above three statistics because they generate *mergeable* sketches (mergeable cardinality estimation sketches of two sets S_1 and S_2 can be used to generate a sketch of $S_1 \cup S_2$, which is used to estimate the cardinality of $S_1 \cup S_2$), Cohen et al. [15] observed that their extensions exhibit large errors for estimating common-interest counts and Jaccard similarities. Even worse, the sketches generated by these cardinality methods cannot be used for LSH indexing, which results in a high computational cost for searching users with similar interests. One can combine multiple existing methods to generate different sketches for estimating user cardinalities, common-interest counts, and Jaccard similarities, respectively. This straightforward method increases the required memory and computational resources, which limits its usage for high-speed data streams.

To address the above challenges, we develop a novel framework, *SimCar*, which is shown in Figure 1. The framework consists of two modules: (1) the *online processing module*, which uses Giroire's algorithm [6] to build an *order-hashing* (OH) sketch of the interest set of each user occurring in the stream; (2) the *online query module*, which provides functions for estimating the cardinalities, intersection cardinalities, and Jaccard similarities of users' interest sets at any time of interest. Specially, using OH sketches, we develop maximum likelihood estimation (MLE) methods for estimating cardinalities and intersection cardinalities of users' interest sets. In addition, we use the technique *optimal densification* [22] to generate *densification OH* (DOH) sketches and then use DOH sketches to estimate Jaccard similarities of users' interest sets and build locality-sensitive hashing tables to search for users with similar interests with sub-linear time. Our main contributions are summarized as follows:

- For sets given in a stream fashion, our method, *SimCar*, builds only one type of sketches (i.e., OH sketch) to effectively mine a variety of their statistics, including cardinality, intersection cardinality, and Jaccard similarity, as well as fast search similar sets. It outperforms the straightforward method of combining existing sketch-based cardinality estimation methods and Jaccard similarity estimation methods.
- We develop maximum likelihood estimation (MLE) methods to estimate set cardinalities and intersection cardinalities, which are more accurate than the original OH-sketch-based cardinality estimation method [6].
- Compared with state-of-the-art methods, experimental results on real-world datasets demonstrate that our method, *SimCar*, is up to 1000 times faster for online data

streams processing, and reduces the memory usage by 13.5–23.8% to achieve the same performance accuracy.

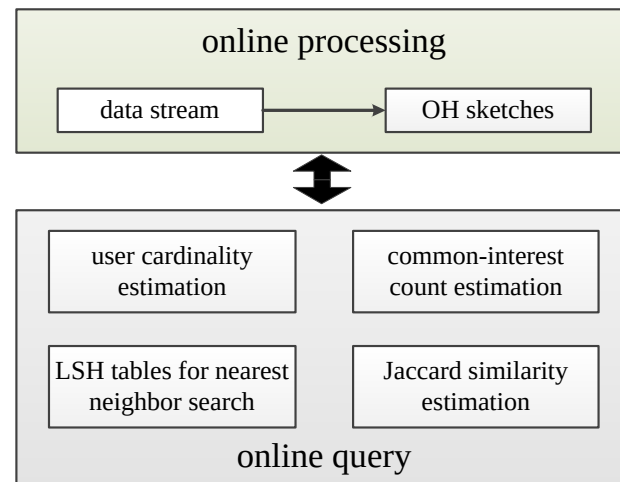


Figure 1. Framework of our framework SimCar.

The rest of this paper is organized as follows. The problem formulation is presented in Section 3. Section 4 introduces the preliminaries. Section 5 presents our framework, SimCar. The performance evaluation is presented in Section 6. Section 2 summarizes related work. Concluding remarks then follow.

2. Related Work

LSH for similarity estimation. LSH maps similar objects (e.g., sets and vectors) to the same hash value with higher probability than dissimilar ones [23], and has been widely used for applications such as similarity estimation and nearest neighbor search. *MinHash* [16] is a popular LSH technique for estimating the Jaccard similarity coefficient of sets. Recently, considerable attention has been paid to improving the performance of MinHash. For example, *b-bit minwise hashing* [18], *odd sketch* [19], and *MaxLogHash* [27] use probabilistic techniques such as sampling and sketching to further reduce the amount of memory used by MinHash. However, all these methods update each element with a high complexity of $O(k)$. To solve this problem, *OPH* [17] and its variants [20–22,28] were developed to accelerate the speed of processing each element in sets by orders of magnitude. Very recently, Raul et al. [29] used *HyperLogLog* [5] and *OPH* [17] sketches to estimate the Jaccard similarity, as well as the containment. In addition to sets (i.e., binary vectors), a variety of sketch methods have also been developed to estimate similarity between real-value weighted vectors. For example, Charikar [25] developed a method, *SimHash*, for approximating angle similarity (i.e., cosine similarity) between weighted vectors. Datar et al. [26] used p -stable distributions to estimate l_p distance between weighted vectors, where $0 < p \leq 2$. Refs. [30–38] were developed to approximate the weighted Jaccard similarity of weighted vectors. Ryan et al. [39] proposed a Gumbel-Max-Trick-based sketching method, *P-MinHash*, to estimate another novel Jaccard similarity metric, *probability Jaccard similarity*. They also demonstrated that the probability Jaccard similarity is scale invariant and more sensitive to changes in vectors. Qi et al. [40] proposed *FastGM* to further reduce the time complexity of *P-MinHash* by generating the hash values in order.

LSH for nearest neighbor search. To perform a nearest neighbor search, refs. [23,26] first, objects are hashed into a hash table. To perform a similarity search, a query object is hashed into a bucket of the hash table, and then the objects in the bucket are used as candidate objects, which need to be further verified. However, nearest neighbors may not appear as candidate objects because close objects may be hashed into different buckets. To solve this problem, refs. [41–43] proposed methods that look up more than one bucket of the hash table for a query, which significantly reduces the amount of storage. To further

improve the performance, many techniques, such as *C2LSH* [44], *SK-LSH* [45] and *LSB-Frest* [46], were developed to reduce memory usage and query cost. After gathering the candidate objects, but before verifying them, Satuluri et al. [47] developed a method *BayesLSH* that performs similarity estimation and candidate pruning using LSH sketches to accelerate the procedure of verifying candidate objects. Real-world datasets are usually not distributed uniformly over the space, which results in unbalanced LSH buckets and degrades the performance of LSH for nearest neighbor search. To solve this problem, Gao et al. [48] developed a method *DSH* leveraging data distributions to hash nearest neighbors together with a larger probability. Wang et al. [49] developed a system, FLASH, using OPH and reservoir sampling to overcome the computational and parallelization hurdles. Recently, several LSH-based methods [50–52] and transformation-based methods [53–55] were developed to solve the maximum inner product search (MIPS) problem, i.e., search vectors having maximum inner products with a given query vector.

Cardinality estimation. To estimate a large data set's cardinality, Whang et al. [3] developed the first fast sketch method LPC. Additionally, refs. [2,12] used different sampling methods to enlarge the estimation range of LPC. Flajolet and Martin [56] developed a sketch method, FM, which uses a register initialized to zero for cardinality estimation. To further improve the performance of FM, sketch methods LogLog [4], HyperLogLog [5], RoughEstimator [7], and HLL-TailCut+ [57] were developed to use a list of m registers and compress the register size from 32 bits to 5 or 4 bits. Giroire [6] developed a sketch method, *MinCount*, (also known as bottom- k sketch [58]) which stores the k minimum hash values of elements in the stream to estimate the stream cardinality. Lumbroso [59] developed another order-statistics-based cardinality estimation method which hashes the data stream's elements into k registers at random, and each register stores the minimum hash value of elements hashed into it. Ting [53] developed a martingale-based estimator to further improve the accuracy of the above sketch methods, such as LPC, HyperLogLog, and MinCount. Chen et al. [60] extended HyperLogLog to sliding windows. In addition to sketch methods, two sampling methods [61,62] were also developed for cardinality estimation. Recently, considerable attention [8–11] has been given to developing fast sketch methods to monitor the cardinalities of network hosts over high-speed links. Ting [13] developed methods to estimate the cardinality of set unions and intersections from MinCount sketches. Cohen et al. [15] developed a method combining MinHash and HyperLogLog to estimate set intersection cardinalities. These two methods fail to solve the problem studied in this paper, because MinCount does not belong to the LSH family, and cannot be used for sub-linear time similarity search, and MinHash fails to deal with high-speed data streams due to its high time complexity.

LSH for clustering and feature tracking. LSH can also support other analytical tasks, such as clustering and feature tracking. Mao et al. [63] proposed the MR-PGDLSH algorithm to reduce the overhead of frequent communication between nodes through LSH, so as to solve the problem of excessive communication overhead faced by the partition-based K-means clustering algorithm in the big data environment. Corizzo et al. [64,65] proposed the DENCAS algorithm to process large-scale high-dimensional data using LSH, which is more efficient than the most advanced distributed clustering algorithm. Cao et al. [66] used LSH-based k -nearest neighbor matching to generate feature correspondence, and then used a ratio test method to remove outliers from the previous matching set, and finally achieved better results. Ding et al. [67] proposed a perceptual hash algorithm based on deep learning and LSH to generate edge feature samples to improve the robustness of HRRS image authentication.

3. Problem Formulation

Let $\Gamma = e^{(1)}, \dots, e^{(t)}, \dots$ denote the stream of interest, where $e^{(t)} = (u^{(t)}, w^{(t)})$ is the t^{th} element of stream Γ , which represents a link from a user $u^{(t)}$ to an interest $w^{(t)}$. Note that a pair (u, w) may appear more than once in stream Γ . Denote by U the set of occurring users, and W the set of occurring interests. Let N_u be the interest set of user u , i.e., the

set of interests that user u links to. User u 's cardinality is defined as the number of its distinct interests, i.e., $d_u = |N_u|$. For two users u and v , we define their common-interest count as $d_{u \cap v} = |N_u \cap N_v|$, and their Jaccard similarity as $J_{u,v} = \frac{|N_u \cap N_v|}{|N_u \cup N_v|}$. In this paper, we aim to design a fast method to build a compact data summary (or, sketch) for each user in a single-pass stream Γ . For any specific query user u , generated sketches are effective for rapidly searching for nearest neighbors of user u (i.e., users $v \in U$ with large Jaccard similarity $J_{u,v}$) without enumerating each user in U . More importantly, they are also accurate for estimating the cardinality d_v of any user v , Jaccard similarity $J_{u,v}$, as well as the common-interest count $d_{u \cap v}$ of any two users u and v . For ease of reference, we list the notation used throughout the paper in Table 1.

Table 1. Table of notation.

Γ	The stream of interest
U, W	The sets of users and interests,
r_w	The random rank value of interest $w \in W$
i_w	$i_w = \lfloor r_w k \rfloor + 1$
k	The number of registers used for a MinHash, OPH, or OH sketch
k_1	The number of registers used for an HLL sketch
N_u	The set of interests of user $u \in U$
$N_{u \cap v}$	The set of common interests of users u and v in U , i.e., $N_{u \cap v} = N_u \cap N_v$
$N_{u \setminus v}$	The set of interests affiliated with user u but not user v , i.e., $N_{u \setminus v} = N_u \setminus N_v$
$N_{u,i}$	The set of interests w in N_u with $i_w = i$
$N_{u \cap v,i}$	The set of interests w in $N_{u \cap v}$ with $i_w = i$
$N_{u \setminus v,i}$	The set of interests w in $N_{u \setminus v}$ with $i_w = i$
d_u	The cardinality of user $u \in U$, i.e., $d_u = N_u $
$d_{u \cap v}$	The common-interest count of users u and v , i.e., $d_{u \cap v} = N_u \cap N_v $
$d_{u \setminus v}$	The number of elements in set $N_{u \setminus v}$ i.e., $d_{u \setminus v} = N_u \setminus N_v $
$d_{u,i}$	The number of interests w in N_u with $i_w = i$
$d_{u \cap v,i}$	The number of interests w in $N_{u \cap v}$ with $i_w = i$
$d_{u \setminus v,i}$	The number of interests w in $N_{u \setminus v}$ with $i_w = i$
$J_{u,v}$	The Jaccard similarity of sets N_u and N_v
$\vec{x}_u$	OH sketch of N_u , $u \in U$
$\vec{x}_u^*$	DOH sketch of N_u , $u \in U$
$x_{u,i}, x_{u,i}^*$	the i^{th} element of vectors $\vec{x}_u$ and $\vec{x}_u^*$, respectively
$\pi(\cdot)$	Random permutation from W to W
$\rho(\cdot)$	Random variable selected from $(0, 1)$ uniformly
b, c	Two parameters of LSH tables
$f(x y)$	PDF of random variable x given condition y
Φ_u	The set of elements in $\vec{x}_u$ that are smaller than 1
k_u	The number of elements in $\vec{x}_u$ that equal 1

Table 1. *Cont.*

$\lambda_u, \lambda_{u,v}$	Parameters of Poisson approximations
$\Phi_{u,v}^{(j)}$	The set of elements in $\vec{x}_u$ and $\vec{x}_v$ whose relations are Case j defined in Section 5.3, $1 \leq j \leq 6$
$k_{u,v}^{(j)}$	The cardinality of set $\Phi_{u,v}^{(j)}$, $1 \leq j \leq 6$
$g(\vec{\lambda}), H(\vec{\lambda})$	Gradient and Hessian matrix of likelihood function $\log \mathcal{L}(\vec{x}_u, \vec{x}_v)$ at $\vec{\lambda} = (\lambda_u, \lambda_v, \lambda_{u,v})$

4. Preliminaries

In this section, we present Giroire’s algorithm [6] for cardinality estimation, and the OPH method [17] as well as the *optimal densification* [22] for Jaccard similarity estimation, which are closely related with our framework, SimCar.

4.1. Giroire’s Algorithm

Giroire [6] developed a fast method, *CORE*, for estimating a stream’s cardinality, i.e., the number of distinct elements occurring in the stream. The basic idea behind *CORE* is that one can assign each element in the stream a rank that is randomly and uniformly selected from the range (0,1), and to some extent, the minimum of occurring elements’ ranks reflects the number of distinct occurring elements (i.e., the stream’s cardinality). To accelerate the performance, *CORE* splits elements in the stream into k subsets, and keeps track of the minimum rank for each subset, which is finally used to infer the stream’s cardinality. In detail, *CORE* uses k registers $x_1, \dots, x_k$ to generate a sketch of the stream, where $x_1, \dots, x_k$ are initialized to 1. Let r_e be the rank of an element e that is uniformly selected from range (0, 1) at random; that is, $r_e \sim \text{Uniform}(0, 1)$. For an element e arriving on the stream, *CORE* computes $i_e = \lfloor r_e k \rfloor + 1$ and then updates the i_e^{th} register x_{i_e} as

$$x_{i_e} \leftarrow \min(x_{i_e}, r_e k - i_e - 1). \quad (1)$$

At the end of the stream, the stream’s cardinality is estimated as $d^* = \frac{k(k-1)}{\sum_{i=1}^k x_i}$. Since d^* is severely biased for small cardinalities, *CORE* treats the list of registers $x_1, \dots, x_k$ as an LPC sketch (i.e., a bitmap of k bits) [3]. Let n_1 be the number of registers among $x_1, \dots, x_k$ that equal 1 at the end of the stream. When $\frac{n_1}{k} \leq 86\%$, *CORE* estimates the stream’s cardinality as $-k \log \frac{n_1}{k}$.

4.2. OPH

OPH [17] can be viewed as a sampling method, which samples (at most) k distinct elements from each set of interest without replacement and then estimates the Jaccard similarity of two sets based on their sampled elements. Compared with the well-known sketch method MinHash [16], which is used to estimate set Jaccard similarity, OPH reduces the time complexity of processing each element in set N_u from $O(k)$ to $O(1)$. In detail, to build a sketch of set $N_u \subseteq W$, OPH [17] uses a single permutation function $\pi(\cdot) : W \rightarrow W$ to process each element in a set N_u . Specifically, it evenly divides $W = \{1, \dots, p\}$ into k bins: $W_j = \left\{ \left\lfloor \frac{p(j-1)}{k} + 1 \right\rfloor, \dots, \left\lfloor \frac{pj}{k} \right\rfloor \right\}$, $1 \leq j \leq k$. The OPH sketch of set N_u is a vector $\vec{x}_u = (x_{u,1}, \dots, x_{u,k})^T$, where each element $x_{u,j}$, $1 \leq j \leq k$, is computed as

$$x_{u,j} = \begin{cases} \min_{w \in N_u \wedge \pi(w) \in W_j} \pi(w), & \exists w \in N_u, \pi(w) \in W_j \\ \emptyset, & \text{otherwise.} \end{cases}$$

Given that at least one of $x_{u,j}$ and $x_{v,j}$ does not equal $\emptyset$, Li et al.[17] observed that $x_{u,j} = x_{v,j}$ occurs with probability $J_{u,v}$, and proposed estimating $J_{u,v}$ based on this observation. Let $\mathbf{1}(\mathbb{P})$ be the indicator function that equals one when the predicate $\mathbb{P}$ is true, and

zero otherwise. Formally, OPH estimates the Jaccard similarity $J_{u,v}$ of two sets N_u and N_v based on their OPH sketches $\vec{x}_u$ and $\vec{x}_v$ as

$$\hat{J}_{u,v} = \frac{\sum_{i=1}^k \mathbf{1}(x_{u,i} = x_{v,i} \wedge x_{u,i} \neq \emptyset \wedge x_{v,i} \neq \emptyset)}{\sum_{i=1}^k \mathbf{1}(x_{u,i} \neq \emptyset \vee x_{v,i} \neq \emptyset)}.$$

As shown in Algorithm 1, one can apply the above OPH algorithm online to stream Γ and compute the OPH sketch of each user's interest set N_u , $u \in U$.

Algorithm 1: Online update procedure of OPH [17].

```

input : stream  $\Gamma$ ,  $k$ .
output:  $(x_{u,1}, \dots, x_{u,k})$ ,  $u \in U$ .

foreach user  $u$  in set  $U$  do
  for  $i = 1, \dots, k$  do
     $x_{u,i} \leftarrow \emptyset$ ;
  end
end

foreach  $e = (u, w)$  in  $\Gamma$  do
   $i_w \leftarrow \lfloor \frac{\pi(w)}{k} \rfloor + 1$ ;
  if  $x_{u,i_w} == \emptyset$  then
     $x_{u,i_w} \leftarrow \pi(w)$ ;
  end
  else
     $x_{u,i_w} \leftarrow \min(x_{u,i_w}, \pi(w))$ ;
  end
end

```

4.3. Optimal Densification

Clearly, the OPH method generates sketches (i.e., vectors) that may have empty elements (i.e., elements equal $\emptyset$), which hinders its application for LSH-based similarity search with sub-linear time, and also results in a large error in the Jaccard similarity estimation. To solve this problem, refs. [20–22] proposed several densification techniques, which reassign empty elements using the values of non-empty elements. The state-of-the-art densification technique, *optimal densification* [22], is shown in Algorithm 2. Specifically, when the i^{th} element of OPH sketch $\vec{x}_u$ is empty (i.e., $x_{u,i} = \emptyset$), optimal densification iteratively uses a hash function $j = h(i, l) : \{1, \dots, k\} \times \{1, 2, \dots\} \rightarrow \{1, \dots, k\}$, to find a non-empty element $x_{u,j}$ (i.e., $x_{u,j} \neq \emptyset$). Function $h(i, l)$ takes two arguments: (1) i , the current empty element that needs to be reassigned; (2) l , the number of failed attempts made so far to reach a non-empty element, which is initialized to 1. Let $\vec{x}_u^*$ denote the optimal densification of OPH sketch $\vec{x}_u$. For any sets N_u and N_v , their Jaccard similarity $J_{u,v}$ is estimated as

$$\hat{J}_{u,v}^* = \frac{\sum_{i=1}^k \mathbf{1}(x_{u,i}^* = x_{v,i}^*)}{k}.$$

Shrivastava [22] proves that $\hat{J}_{u,v}^*$ is an unbiased estimator of $J_{u,v}$ and is more accurate than the original OPH method, as well as other densification techniques in [20,21].

Algorithm 2: Optimal densification [22].

```

input :  $\vec{x}_u = (x_{u,1}, \dots, x_{u,k})^\top$ .
output:  $\vec{x}_u^* = (x_{u,1}^*, \dots, x_{u,k}^*)^\top$ .

foreach  $i = 1, \dots, k$  do
  if  $x_{u,i} \neq \emptyset$  then
     $x_{u,i}^* \leftarrow x_{u,i}$ ;
  end
  else
     $l \leftarrow 1$ ;
     $j \leftarrow h(i, l)$ ;
    while  $x_{u,j} == \emptyset$  do
       $l \leftarrow l + 1$ ;
       $j \leftarrow h(i, l)$ ;
    end
     $x_{u,i}^* \leftarrow x_{u,j}$ ;
  end
end

```

5. Our Framework SimCar

In this section, we use Giroire’s algorithm [6] (Algorithm 3) to build an OH sketch for each occurring user. Based on generated OH sketches, we first present our methods for estimating the Jaccard similarity of users’ interest sets and quickly searching users with similar interests. Then, we propose methods for estimating the cardinalities and common-interest counts of users’ interest sets.

Algorithm 3: The CORE algorithm [59].

```

input : stream  $\Gamma, k$ .
output:  $d_u, u \in U$ .

 $U \leftarrow \emptyset$ ;
foreach  $e = (u, w)$  in  $\Gamma$  do
  if  $u \notin U$  then
     $U \leftarrow U \cup \{u\}$ ;
    foreach  $i = 1, \dots, k$  do
       $x_{u,i} \leftarrow 1$ ;
    end
  end
   $i_w \leftarrow \lfloor r_w k \rfloor + 1$ ;
   $x_{u,i_w} \leftarrow \min(x_{u,i_w}, r_w k - i_w - 1)$ ;
end

return  $d_u^* \leftarrow \frac{k(k-1)}{\sum_{i=1}^k x_{u,i}}, u \in U$ ;

```

5.1. Jaccard Similarity Estimation and Nearest Neighbor Search

Jaccard Similarity Estimation. Let $\vec{x}_u^{(*)}$ be the DOH sketch of OH sketch $\vec{x}_u, u \in U$, which is computed using the optimal densification technique introduced in Section 4. Similar to optimal densification [22], we find that $P(x_{u,i}^* = x_{v,i}^*) = J_{u,v}$ for any $1 \leq i \leq k$. Then, we estimate the Jaccard similarity of $J_{u,v}$ as

$$\hat{J}_{u,v}^* = \frac{\sum_{i=1}^k \mathbf{1}(x_{u,i}^* = x_{v,i}^*)}{k}.$$

Following the error analysis in [22], we find that $\hat{J}_{u,v}^*$ is an unbiased estimate of $J_{u,v}$ and has the same accurate performance as the optimal densification of OPH.

Nearest Neighbor Search. A variety of LSH methods [16,23–26] have been developed to solve the task of nearest neighbor search with sub-linear time. The basic idea behind these LSH methods is to place similar users into the same bucket of a hash table with high probability. Following the principle of LSH, our nearest neighbor search method consists of two basic phases:

- **Pre-processing phase.** We first construct b hash tables $\mathbb{H}_1, \dots, \mathbb{H}_b$ for all users in U . Let $k = bc$, where c is an integer parameter used to determine the performance of LSH. Let $\vec{x}_u^{(*,i,c)}$, $1 \leq i \leq b$, denote the vector consisting of the $((i-1)c+1)^{\text{th}}$ to $(ic)^{\text{th}}$ elements of DOH sketch $\vec{x}_u^*$, i.e., $\vec{x}_u^{(*,i,c)} = (x_{u,(i-1)c+1}^*, \dots, x_{u,ic}^*)^T$. Then, we insert each user u and its densification sketch $\vec{x}_u^*$ into hash tables $\mathbb{H}_1, \dots, \mathbb{H}_b$ as

$$\mathbb{H}_i[\vec{x}_u^{(*,i,c)}] \leftarrow \mathbb{H}_i[\vec{x}_u^{(*,i,c)}] \cup \{u\}, \quad 1 \leq i \leq b.$$

- **Retrieving phase.** Given a specific user u , we search for its similar users. Instead of scanning all users in set U , we only probe b different buckets $\mathbb{H}_1[\vec{x}_u^{(*,i,c)}], \dots, \mathbb{H}_b[\vec{x}_u^{(*,i,c)}]$ from b hash tables, respectively, and report the users in any of these buckets as the potential candidates. Last, nearest neighbors of user u are detected by enumerating all users in set $\mathbb{H}_1[\vec{x}_u^{(*,i,c)}] \cup \dots \cup \mathbb{H}_b[\vec{x}_u^{(*,i,c)}]$, and computing and sorting their Jaccard similarity estimations.

5.2. User Cardinality Estimation

In this section, we estimate the cardinality of any queried user. Specially, we first build a probabilistic model according to the state of the OH sketch, and then solve it using MLE methods.

Exact probabilistic model. For any user $u \in U$, let $N_{u,i}$ denote the set of its interests whose hash values equal i with respect to hash function f ; that is,

$$N_{u,i} = \{w : i_w = i, w \in N_u\}, \quad 1 \leq i \leq k.$$

At any time of stream Γ , from Equation (1), we have

$$x_{u,i} = \min_{w \in N_{u,i}} r_w k - \lfloor r_w k \rfloor.$$

Define $d_{u,i} = |N_{u,i}|$. Let $f(x_{u,i} = x | d_{u,i} = d)$ denote the probability density function (PDF) of random variable $x_{u,i}$ at x given $d_{u,i} = d$. Then, we have

$$f(x_{u,i} = x | d_{u,i} = d) = \begin{cases} d(1-x)^{d-1}, & d > 0 \wedge 0 < x < 1 \\ 1, & d = 0 \wedge x = 1 \\ 0, & \text{otherwise.} \end{cases}$$

Our algorithm randomly splits users in set N_u into k subsets $N_{u,1}, \dots, N_{u,k}$. Using the classical balls-in-urns model, we derive the probability distribution of vector $(d_{u,1}, \dots, d_{u,k})$ as

$$P(d_{u,1}, \dots, d_{u,k} | d_u) = \frac{\binom{d_u}{d_{u,1}, \dots, d_{u,k}}}{k^{d_u}},$$

where $\binom{d_u}{d_{u,1}, \dots, d_{u,k}} = \frac{d_u!}{d_{u,1}! \dots d_{u,k}!}$. Therefore, the PDF of $\vec{x}_u$ given d_u is computed as

$$f(\vec{x}_u | d_u) = \sum_{d_{u,1} + \dots + d_{u,k} = d_u} \frac{\binom{d_u}{d_{u,1}, \dots, d_{u,k}}}{k^{d_u}} \prod_{i=1}^k f(x_{u,i} | d_{u,i}).$$

Poisson approximation model. Clearly, $d_{u,1}, \dots, d_{u,k}$ are not independent, which hinders us when it comes to obtaining the MLE of d_u . Similar to [5], we use the Poisson approximation technique to remove the dependence of $d_{u,1}, \dots, d_{u,k}$. Specifically, we assume that the value of d_u is distributed according to a Poisson distribution with parameter λ_u , i.e., $d_u \sim \text{Poisson}(\lambda_u)$. Then, the PDF of $\vec{x}_u$ given λ_u is

$$\begin{aligned} f(\vec{x}_u | \lambda_u) &= \sum_{d_u=0}^{+\infty} \mathcal{L}(\vec{x}_u | d_u) \frac{e^{-\lambda_u} \lambda_u^{d_u}}{d_u!} \\ &= \sum_{d_u=0}^{+\infty} \sum_{d_{u,1}+\dots+d_{u,k}=d_u} \frac{\binom{d_u}{d_{u,1}, \dots, d_{u,k}}}{k^{d_u}} \prod_{i=1}^k f(x_{u,i} | d_{u,i}) \frac{e^{-\lambda_u} \lambda_u^{d_u}}{d_u!} \\ &= \sum_{d_{u,1}=0}^{+\infty} \dots \sum_{d_{u,k}=0}^{+\infty} \prod_{i=1}^k f(x_{u,i} | d_{u,i}) \frac{e^{-\frac{\lambda_u}{k} \lambda_u^{d_{u,i}}}}{d_{u,i}! k^{d_{u,i}}} \\ &= \prod_{i=1}^k \sum_{d_{u,i}=0}^{+\infty} f(x_{u,i} | d_{u,i}) \frac{e^{-\frac{\lambda_u}{k} \lambda_u^{d_{u,i}}}}{d_{u,i}! k^{d_{u,i}}} \\ &= \prod_{i=1}^k f(x_{u,i} | \lambda_u). \end{aligned}$$

Given $d_u \sim \text{Poisson}(\lambda_u)$, the above equation indicates that the values of $x_{u,1}, \dots, x_{u,k}$ are independent and identically distributed. In addition, the values of $d_{u,1}, \dots, d_{u,k}$ are independent and identically distributed according to a Poisson distribution $\text{Poisson}(\frac{\lambda_u}{k})$. Formally, we have

$$d_{u,i} \sim \text{Poisson}\left(\frac{\lambda_u}{k}\right), \quad i = 1, \dots, k. \quad (2)$$

For an estimator $\phi(\vec{x}_u)$ of d_u (e.g., Equation (5), derived later), we let $\mathbb{E}_{d_u}(\phi(\vec{x}_u))$ and $\text{Var}_{d_u}(\phi(\vec{x}_u))$ denote the expectation and the variance of $\phi(\vec{x}_u)$ under the exact probabilistic model (i.e., $f(\vec{x}_u | d_u)$), and let $\mathbb{E}_{\text{Poisson}(\lambda_u)}(\phi(\vec{x}_u))$ and $\text{Var}_{\text{Poisson}(\lambda_u)}(\phi(\vec{x}_u))$ denote the expectation and the variance of $\phi(\vec{x}_u)$ under the Poisson approximation model (i.e., $f(\vec{x}_u | d_u)$). The authors of [5,68,69] reveal that statistical properties of $\mathbb{E}_{d_u}(\phi(\vec{x}_u))$ and $\text{Var}_{d_u}(\phi(\vec{x}_u))$ are well approximated by $\mathbb{E}_{\text{Poisson}(\lambda_u)}(\phi(\vec{x}_u))$ and $\text{Var}_{\text{Poisson}(\lambda_u)}(\phi(\vec{x}_u))$ when setting $\lambda_u = d_u$, which is the *depoissonization step*.

Estimator of λ_u . Next, we elaborate our method to derive the MLE of λ_u under the Poisson approximation model. For any $0 < x < 1$, the following equation holds

$$\begin{aligned} f(x_{u,i} = x | \lambda_u) &= \sum_{d_{u,i}=0}^{+\infty} f(x_{u,i} = x | d_{u,i}) \frac{e^{-\frac{\lambda_u}{k} \lambda_u^{d_{u,i}}}}{d_{u,i}! k^{d_{u,i}}} \\ &= e^{-\frac{\lambda_u}{k}} + \sum_{d_{u,i}=1}^{\infty} d_{u,i} (1-x)^{d_{u,i}-1} \frac{e^{-\frac{\lambda_u}{k} \lambda_u^{d_{u,i}}}}{d_{u,i}! k^{d_{u,i}}} \\ &= \frac{\lambda_u}{k} e^{-\frac{\lambda_u x}{k}}, \end{aligned} \quad (3)$$

where the last equation holds because of the Taylor series of $\frac{\lambda_u}{k} e^{-\frac{\lambda_u x}{k}}$ at $x_0 = 1$. Then, we compute the PDF of $x_{u,i}$ at x given λ_u as

$$f(x_{u,i} = x | \lambda_u) = \begin{cases} \frac{\lambda_u}{k} e^{-\frac{\lambda_u x}{k}}, & 0 < x < 1 \\ e^{-\frac{\lambda_u}{k}}, & x = 1. \end{cases} \quad (4)$$

The last equation holds because we have $d_{u,i} = 0$ when $x = 1$, which occurs with probability $P(d_{u,i} = 0|\lambda_u) = e^{-\frac{\lambda_u}{k}}$. Let Φ_u denote the set of elements in vector $\vec{x}_u$ that are smaller than 1, i.e.,

$$\Phi_u = \{i : x_{u,i} < 1, 1 \leq i \leq k\}.$$

Denoted by k_u is the number of elements in vector $\vec{x}_u$ that equal 1. Then, the likelihood function of λ_u given by k independent observed variables $x_{u,1}, \dots, x_{u,k}$ is computed as

$$\begin{aligned} \mathcal{L}(\vec{x}_u|\lambda_u) &= \prod_{i=1}^k f(x_{u,i}|\lambda_u) \\ &= e^{-\frac{\lambda_u k_u}{k}} \prod_{i \in \Phi_u} \frac{\lambda_u}{k} e^{-\frac{\lambda_u x_{u,i}}{k}} \\ &= \left(\frac{\lambda_u}{k}\right)^{k-k_u} e^{-\frac{\lambda_u(k_u + \sum_{i \in \Phi_u} x_{u,i})}{k}} \\ &= \left(\frac{\lambda_u}{k}\right)^{k-k_u} e^{-\frac{\lambda_u \sum_{i=1}^k x_{u,i}}{k}}. \end{aligned}$$

Let $\hat{\lambda}_u = \arg \max_{\lambda_u} \log \mathcal{L}(\vec{x}_u|\lambda_u)$ denote the MLE of λ_u . We compute the derivative of $\log \mathcal{L}(\vec{x}_u|\lambda_u)$ as

$$\frac{\partial \log \mathcal{L}(\vec{x}_u|\lambda_u)}{\partial \lambda_u} = \frac{k - k_u}{\lambda_u} - \frac{\sum_{i=1}^k x_{u,i}}{k}.$$

Let $\frac{\partial \log \mathcal{L}(\vec{x}_u|\lambda_u)}{\partial \lambda_u} = 0$. Then, we obtain the MLE of λ_u as

$$\hat{\lambda}_u = \frac{k(k - k_u)}{\sum_{i=1}^k x_{u,i}}. \quad (5)$$

To analyze the estimation error of $\hat{\lambda}_u$, we first compute the Fisher information of λ_u as

$$\begin{aligned} \mathcal{I}(\lambda_u) &= -\mathbb{E}\left(\frac{\partial^2 \log \mathcal{L}(\vec{x}_u|\lambda_u)}{\partial \lambda_u^2}\right) \\ &= \mathbb{E}\left(\frac{k - k_u}{\lambda_u^2} \mid \lambda_u\right) \\ &= \frac{k(1 - e^{-\frac{\lambda_u}{k}})}{\lambda_u^2}. \end{aligned}$$

The last equation is obtained because $\mathbb{E}(k_u|\lambda_u) = ke^{-\frac{\lambda_u}{k}}$, which is easily derived from Equation (4). From [70], we find that the MLE $\hat{\lambda}_u$ is an asymptotically efficient unbiased estimator of λ_u , and its mean square error (MSE) asymptotically approaches the Cramér–Rao lower bound (CRLB) of λ_u , which is defined as $\frac{1}{\mathcal{I}(\lambda_u)}$. Formally, we have $\lim_{k \rightarrow +\infty} \hat{\lambda}_u \rightarrow \lambda_u$ and $\lim_{k \rightarrow +\infty} \text{MSE}(\hat{\lambda}_u) \rightarrow \frac{1}{\mathcal{I}(\lambda_u)}$. Lastly, at the dePoissonization step, we use $\hat{\lambda}_u$ to approximate d_u . In our experiments, we will show that the above MLE estimator is more accurate than the Giroire’s algorithm (Section 4.1), especially for small cardinalities.

5.3. Common-Interest Count Estimation

Similarly, we use the MLE and Poisson approximation techniques to estimate the common-interest count $d_{u \cap v}$ of any two users $u, v \in U$. Specifically, we assume

$$d_{u \cap v} \sim \text{Poisson}(\lambda_{u,v}).$$

Define $N_{u \setminus v} = N_u \setminus N_v$ and $N_{v \setminus u} = N_v \setminus N_u$. Let $d_{u \setminus v} = |N_{u \setminus v}|$ and $d_{v \setminus u} = |N_{v \setminus u}|$. We also assume

$$d_{u \setminus v} \sim \text{Poisson}(\lambda_u - \lambda_{u,v}),$$

$$d_{v \setminus u} \sim \text{Poisson}(\lambda_v - \lambda_{u,v}),$$

where $\lambda_u \geq \lambda_{u,v}$ and $\lambda_v \geq \lambda_{u,v}$. Similar to $N_{u,i}$, $1 \leq i \leq k$, we define

$$N_{u \cap v, i} = \{w : i_w = i, w \in N_{u \cap v}\},$$

$$N_{u \setminus v, i} = \{w : i_w = i, w \in N_{u \setminus v}\},$$

$$N_{v \setminus u, i} = \{w : i_w = i, w \in N_{v \setminus u}\}.$$

Let $d_{u \cap v, i} = |N_{u \cap v, i}|$, $d_{u \setminus v, i} = |N_{u \setminus v, i}|$, and $d_{v \setminus u, i} = |N_{v \setminus u, i}|$. Similar to Equation (2), we find that $d_{u \cap v, 1}, \dots, d_{u \cap v, k}, d_{u \setminus v, 1}, \dots, d_{u \setminus v, k}, d_{v \setminus u, 1}, \dots, d_{v \setminus u, k}$ are independent, and they are distributed according to the following Poisson distributions

$$d_{u \cap v, i} \sim \text{Poisson}\left(\frac{\lambda_{u,v}}{k}\right),$$

$$d_{u \setminus v, i} \sim \text{Poisson}\left(\frac{\lambda_u - \lambda_{u,v}}{k}\right),$$

$$d_{v \setminus u, i} \sim \text{Poisson}\left(\frac{\lambda_v - \lambda_{u,v}}{k}\right).$$

Define $x_{u \cap v, i} = \min_{w \in N_{u \cap v, i}} r_w$, $x_{u \setminus v, i} = \min_{w \in N_{u \setminus v, i}} r_w$, and $x_{v \setminus u, i} = \min_{w \in N_{v \setminus u, i}} r_w$. Then, we find that

$$x_{u, i} = \min(x_{u \cap v, i}, x_{u \setminus v, i}),$$

$$x_{v, i} = \min(x_{u \cap v, i}, x_{v \setminus u, i}).$$

Let $f(x_{u, i} = x, x_{v, i} = x' | \lambda_u, \lambda_v, \lambda_{u,v})$ denote the PDF of the random variable pair $(x_{u, i}, x_{v, i})$ at (x, x') , given λ_u, λ_v , and $\lambda_{u,v}$. In what follows, we omit the conditions λ_u, λ_v , and $\lambda_{u,v}$ for simplicity when no confusion is raised. We derive $f(x_{u, i} = x, x_{v, i} = x')$ for all possible relations between $x_{u, i}$ and $x_{v, i}$ as follows:

Case 1: $x_{u, i} = x_{v, i} = 1$. This case indicates $d_{u \cap v, i} = d_{u \setminus v, i} = d_{v \setminus u, i} = 0$. Therefore, we have

$$\begin{aligned} f(x_{u, i} = 1, x_{v, i} = 1) &= P(d_{u \cap v, i} = 0)P(d_{u \setminus v, i} = 0)P(d_{v \setminus u, i} = 0) \\ &= e^{-\frac{\lambda_{u,v}}{k}} e^{-\frac{\lambda_u - \lambda_{u,v}}{k}} e^{-\frac{\lambda_v - \lambda_{u,v}}{k}} \\ &= e^{-\frac{\lambda_u + \lambda_v - \lambda_{u,v}}{k}}. \end{aligned} \quad (6)$$

Case 2: $x_{u, i} = 1 \wedge 0 < x_{v, i} < 1$. This case indicates $d_{u \cap v, i} = d_{u \setminus v, i} = 0$ and $0 < x_{v \setminus u, i} = x_{v, i} < 1$. Similar to Equation (4), we have $f(x_{v \setminus u, i} = x) = \frac{\lambda_v - \lambda_{u,v}}{k} e^{-\frac{(\lambda_v - \lambda_{u,v})x}{k}}$. Then, we have

$$\begin{aligned} f(x_{u, i} = 1, x_{v, i} = x) &= f(x_{v \setminus u, i} = x)P(d_{u \cap v, i} = 0)P(d_{u \setminus v, i} = 0) \\ &= \frac{\lambda_v - \lambda_{u,v}}{k} e^{-\frac{\lambda_u + (\lambda_v - \lambda_{u,v})x}{k}}, \quad 0 < x < 1. \end{aligned} \quad (7)$$

Case 3: $0 < x_{u,i} < 1 \wedge x_{v,i} = 1$. Similar to Case 2, we have

$$\begin{aligned} f(x_{u,i} = x, x_{v,i} = 1) &= f(x_{u \setminus v,i} = x)P(d_{u \cap v,i} = 0)P(d_{v \setminus u,i} = 0) \\ &= \frac{\lambda_u - \lambda_{u,v}}{k} e^{-\frac{\lambda_v + (\lambda_u - \lambda_{u,v})x}{k}}, \quad 0 < x < 1. \end{aligned} \quad (8)$$

Case 4: $0 < x_{u,i} = x_{v,i} < 1$. This case indicates that $x_{u \cap v,i} < x_{u \setminus v,i}$ and $x_{u \cap v,i} < x_{v \setminus u,i}$. Similar to Equation (4), we obtain $f(x_{u \cap v,i} = x) = \frac{\lambda_{u,v}}{k} e^{-\frac{\lambda_{u,v}x}{k}}$, $f(x_{u \setminus v,i} > x) = e^{-\frac{(\lambda_u - \lambda_{u,v})x}{k}}$, and $f(x_{v \setminus u,i} > x) = e^{-\frac{(\lambda_v - \lambda_{u,v})x}{k}}$. Therefore, we have

$$\begin{aligned} f(x_{u,i} = x, x_{v,i} = x) &= f(x_{u \cap v,i} = x)f(x_{u \setminus v,i} > x)f(x_{v \setminus u,i} > x) \\ &= \frac{\lambda_{u,v}}{k} e^{-\frac{(\lambda_u + \lambda_v - \lambda_{u,v})x}{k}}, \quad 0 < x < 1. \end{aligned} \quad (9)$$

Case 5: $0 < x_{u,i} < x_{v,i} < 1$. This case indicates that $x_{u \setminus v,i} = x_{u,i}$ and $\min(x_{u \cap v,i}, x_{v \setminus u,i}) = x_{v,i}$. We find that $d_{v,i} = d_{u \cap v,i} + d_{v \setminus u,i} \sim \text{Poisson}\left(\frac{\lambda_v}{k}\right)$, and $f(\min(x_{u \cap v,i}, x_{v \setminus u,i}) = x') = \frac{\lambda_v}{k} e^{-\frac{\lambda_v x'}{k}}$. Therefore, we have

$$\begin{aligned} f(x_{u,i} = x, x_{v,i} = x') &= f(x_{u \setminus v,i} = x)f(\min(x_{u \cap v,i}, x_{v \setminus u,i}) = x') \\ &= \frac{(\lambda_u - \lambda_{u,v})\lambda_v}{k^2} e^{-\frac{(\lambda_u - \lambda_{u,v})x + \lambda_v x'}{k}}, \quad 0 < x < x' < 1. \end{aligned} \quad (10)$$

Case 6: $0 < x_{v,i} < x_{u,i} < 1$. Similar to Case 5, we have

$$\begin{aligned} f(x_{u,i} = x, x_{v,i} = x') &= f(\min(x_{u \cap v,i}, x_{u \setminus v,i}) = x)f(x_{v \setminus u,i} = x') \\ &= \frac{(\lambda_v - \lambda_{u,v})\lambda_u}{k^2} e^{-\frac{(\lambda_v - \lambda_{u,v})x' + \lambda_u x}{k}}, \quad 0 < x' < x < 1. \end{aligned} \quad (11)$$

Let $\Phi_{u,v}^{(j)}$ denote the set of integers $i \in \{1, \dots, k\}$, such that the relation between $x_{u,i}$ and $x_{v,i}$ is Case j , $1 \leq j \leq 6$. Define $k_{u,v}^{(j)} = |\Phi_{u,v}^{(j)}|$. Then, we have $k_u = k_{u,v}^{(1)} + k_{u,v}^{(2)}$, $k_v = k_{u,v}^{(1)} + k_{u,v}^{(3)}$, $\Phi_u = \Phi_{u,v}^{(3)} \cup \Phi_{u,v}^{(4)} \cup \Phi_{u,v}^{(5)} \cup \Phi_{u,v}^{(6)}$ and $\Phi_v = \Phi_{u,v}^{(2)} \cup \Phi_{u,v}^{(4)} \cup \Phi_{u,v}^{(5)} \cup \Phi_{u,v}^{(6)}$. Now, we obtain the likelihood function of $\vec{\lambda} = (\lambda_u, \lambda_v, \lambda_{u,v})$ given by k independent observed variable pairs $(x_{u,1}, x_{v,1}), \dots, (x_{u,k}, x_{v,k})$ as

$$\begin{aligned} \mathcal{L}(\vec{x}_u, \vec{x}_v) &= \prod_{i=1}^k f(x_{u,i}, x_{v,i}) = e^{-\frac{(\lambda_u + \lambda_v - \lambda_{u,v})k_{u,v}^{(1)}}{k}} \\ &\times \left(\frac{\lambda_v - \lambda_{u,v}}{k} \right)^{k_{u,v}^{(2)}} e^{-\frac{\lambda_u k_{u,v}^{(2)} + (\lambda_v - \lambda_{u,v}) \sum_{i \in \Phi_{u,v}^{(2)}} x_{v,i}}{k}} \\ &\times \left(\frac{\lambda_u - \lambda_{u,v}}{k} \right)^{k_{u,v}^{(3)}} e^{-\frac{\lambda_v k_{u,v}^{(3)} + (\lambda_u - \lambda_{u,v}) \sum_{i \in \Phi_{u,v}^{(3)}} x_{u,i}}{k}} \\ &\times \left(\frac{\lambda_{u,v}}{k} \right)^{k_{u,v}^{(4)}} e^{-\frac{(\lambda_u + \lambda_v - \lambda_{u,v}) \sum_{i \in \Phi_{u,v}^{(4)}} x_{u,i}}{k}} \\ &\times \left(\frac{(\lambda_u - \lambda_{u,v})\lambda_v}{k^2} \right)^{k_{u,v}^{(5)}} e^{-\frac{\sum_{i \in \Phi_{u,v}^{(5)}} (\lambda_u - \lambda_{u,v})x_{u,i} + \lambda_v x_{v,i}}{k}} \end{aligned}$$

$$\begin{aligned}
& \times \left(\frac{(\lambda_v - \lambda_{u,v})\lambda_u}{k^2} \right)^{k_{u,v}^{(6)}} e^{-\frac{\sum_{i \in \Phi_{u,v}^{(6)}} (\lambda_v - \lambda_{u,v})x_{v,i} + \lambda_u x_{u,i}}{k}} \\
& = e^{-\frac{\lambda_u \sum_{i=1}^k x_{u,i}}{k}} e^{-\frac{\lambda_v \sum_{i=1}^k x_{v,i}}{k}} e^{\frac{\lambda_{u,v} \sum_{i=1}^k \min(x_{u,i}, x_{v,i})}{k}} \\
& \times \left(\frac{\lambda_u - \lambda_{u,v}}{k} \right)^{k_{u,v}^{(3)} + k_{u,v}^{(5)}} \left(\frac{\lambda_v - \lambda_{u,v}}{k} \right)^{k_{u,v}^{(2)} + k_{u,v}^{(6)}} \\
& \times \left(\frac{\lambda_u}{k} \right)^{k_{u,v}^{(6)}} \left(\frac{\lambda_v}{k} \right)^{k_{u,v}^{(5)}} \left(\frac{\lambda_{u,v}}{k} \right)^{k_{u,v}^{(4)}}.
\end{aligned}$$

We use the Newton–Raphson method [71] to compute the MLE of $\vec{\lambda} = (\lambda_u, \lambda_v, \lambda_{u,v})$, i.e., $\hat{\vec{\lambda}} = \arg \max_{\vec{\lambda}} \mathcal{L}(\vec{x}_u, \vec{x}_v)$. The Newton–Raphson method starts from an initial estimation, $\vec{\lambda}^{(0)}$, and then repeats the following procedure

$$\vec{\lambda}^{(l+1)} \leftarrow \vec{\lambda}^{(l)} - \left(H(\vec{\lambda}^{(l)}) \right)^{-1} g(\vec{\lambda}^{(l)}),$$

until a sufficiently accurate root of function $g(\vec{\lambda}) = (0, 0, 0)^T$ is reached, where $g(\vec{\lambda})$ and $H(\vec{\lambda})$ are the gradient and the Hessian matrix of $\log \mathcal{L}(\vec{x}_u, \vec{x}_v)$ at $\vec{\lambda}$ is defined as

$$\begin{aligned}
g(\vec{\lambda}) &= \left(\frac{\partial \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_u}, \frac{\partial \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_v}, \frac{\partial \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_{u,v}} \right)^T, \\
H(\vec{\lambda}) &= \begin{bmatrix} \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_u^2}, & \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_u \partial \lambda_v}, & \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_u \partial \lambda_{u,v}} \\ \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_v \partial \lambda_u}, & \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_v^2}, & \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_v \partial \lambda_{u,v}} \\ \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_{u,v} \partial \lambda_u}, & \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_{u,v} \partial \lambda_v}, & \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_{u,v}^2} \end{bmatrix}.
\end{aligned}$$

The closed formulas of $g(\vec{\lambda})$ and $H(\vec{\lambda})$ are given in Appendix A. To set $\vec{\lambda}^{(0)} = (\lambda_u^{(0)}, \lambda_v^{(0)}, \lambda_{u,v}^{(0)})^T$, we initialize $\lambda_u^{(0)}$ and $\lambda_v^{(0)}$ using the estimates of λ_u and λ_v given in Section 5.2 (i.e., $\lambda_u^{(0)} = \frac{k(k-k_u)}{\sum_{i=1}^k x_{u,i}}$ and $\lambda_v^{(0)} = \frac{k(k-k_v)}{\sum_{i=1}^k x_{v,i}}$) and initialize $\lambda_{u,v}^{(0)} = \frac{\hat{J}_{u,v}(\lambda_u^{(0)} + \lambda_v^{(0)})}{\hat{J}_{u,v} + 1}$, where $\hat{J}_{u,v} = \frac{\sum_{i=1}^k \mathbf{1}(x_{u,i}^* = x_{v,i}^*)}{k}$ is an estimate of Jaccard similarity $J_{u,v}$, which is introduced in Section 5.1.

Next, we analyze the error of obtained MLE $\hat{\lambda}_{u,v}^*$. The Fisher information matrix of $\vec{\lambda}$ is defined as

$$\mathcal{I}(\vec{\lambda}) = -\mathbb{E}(H(\vec{\lambda})).$$

The closed formula of $\mathbb{E}(H(\vec{\lambda}))$ is given in Appendix A. Let $(\mathcal{I}(\vec{\lambda}))_{3,3}^{-1}$ be the (3, 3) element of the inverse of matrix $\mathcal{I}(\vec{\lambda})$. From [70], we find that the MLE $\hat{\lambda}_{u,v}^*$ is an asymptotically efficient unbiased estimator of $\lambda_{u,v}$, and its MSE asymptotically approaches the CRLB of $\lambda_{u,v}$, i.e., $(\mathcal{I}(\vec{\lambda}))_{3,3}^{-1}$. Lastly, we use $\hat{\lambda}_{u,v}^*$ to approximate $d_{u \cap v}$.

5.4. Space and Time Complexities

Complexities of online generating OH sketches. We use k registers for each occurring user in stream Γ , and the time complexity of processing each element of stream Γ is $O(1)$.

Complexities of densification. No extra memory space is required for densification. From [22], we find that computing a user u 's DOH sketch from its OH sketch $\vec{x}_u$ requires time complexity $O(k(\frac{k_u}{k-k_u} + 2))$.

Complexities of user cardinality estimation. No extra memory space is required for user cardinality estimation. The time complexity of estimating a user's cardinality is $O(k)$.

Complexities of common-interest count estimation. No extra memory space is required for common-interest count estimation. We find that our method only requires several Newton–Raphson iterations to converge, where each iteration has a negligible computational cost. Therefore, the time complexity of estimating two users' common-interest counts is $O(k)$.

Complexities of Jaccard similarity estimation. No extra memory space is required for common-interest count estimation. Based on two users' DOH sketches, time $O(k)$ is required when estimating two users' Jaccard similarity.

Complexities of nearest neighbor search. Given DOH sketches of users in U , space $O(|U|b)$ and time $O(|U|b)$ are required for building LSH tables, and searching a query user's nearest neighbors requires time $O(k \log |U|)$.

6. Evaluation

In this section, we evaluate the runtime and accuracy of our framework, SimCar, compared to the state of the art. All experiments were run on the same machine with an Intel Xeon CPU E5-2620 v3 with 2.4 GHz, and all algorithms were implemented in Python.

6.1. Datasets

We performed our experiments on six real-world datasets from different areas. The dataset Flickr [72] is a bipartite network of Flickr users and their group memberships. The dataset Movie [73] is a bipartite network consisting of one million movie ratings from MovieLens (<http://movielens.umn.edu/>, accessed on 10 March 2022).

The dataset Wikipedia [74] is a bipartite network of excellent articles in the English Wikipedia (<http://en.wikipedia.org/wiki/Wikipedia>, accessed on 10 March 2022) which meet a core set of editorial standards, and the words they contain. The dataset Reuters [75] is a bipartite network of article-word inclusions in documents that appeared on Reuters newswire in 1987. The dataset Tropes [76] is a bipartite network of TV tropes (<http://tvtropes.org>, accessed on 10 March 2022), characterizing artistic works by their tropes. The dataset TREC [77] is a bipartite network of 1.7 million text documents from the Text Retrieval Conference's (TREC) (http://trec.nist.gov/data/docs_eng.html, accessed on 10 March 2022), containing 556,000 words of different languages. The detailed statistics of these real-world datasets are summarized in Table 2.

Table 2. Real-world datasets used in our experiments, where “size” refers to the number of elements in the stream.

	# Users	# Interests	Size
Movie	6040	9746	1,000,209
Reuters	21,557	60,234	1,464,182
Tropes	152,093	64,415	3,232,134
Wikipedia	2780	276,739	7,846,807
Flickr	499,610	395,979	8,545,307
TREC	556,077	1,729,302	151,632,178

6.2. Baselines

We compared our framework with the following state-of-the-art methods on all four tasks: user cardinality estimation, common-interest count estimation, Jaccard similarity estimation, and nearest neighbor search.

- **MinHash and HLL.** For common-interest count estimation, the baseline is the method [15], which estimates common-interest counts by generating both MinHash [16] and HyperLogLog [5] (short for HLL in the following paper) sketches for each user's interest set. Clearly, one can use generated HLL sketches to estimate user cardinalities, and

use generated Minhash sketches to estimate Jaccard similarities and build LSH tables for a nearest neighbor search.

- **OPH+HLL.** One can also solve all four tasks by combining OPH [17] and HLL, where OPH [17] is much faster than MinHash and exhibits comparable accuracy for applications such as Jaccard similarity estimation and nearest neighbor search [20–22]. In detail, HLL sketches were used to estimate user cardinalities, and optimal densified OPH sketches were used to estimate Jaccard similarity and build LSH tables for a rapid nearest neighbor search. To estimate the common-interest count of any two users u and v , we first easily obtained (1) estimations $\hat{d}_u$, $\hat{d}_v$, and $\hat{d}_{u \cup v}$ of d_u , d_v , and $d_{u \cup v} = |N_u \cup N_v|$ from HLL sketches and our OH sketches; (2) estimation $\hat{J}_{u,v}$ of Jaccard similarity $J_{u,v}$ from optimal densified OPH sketches and DOH sketches. As shown in [15], the common-interest count $d_{u \cap v}$ can be estimated by each of the following schemes: **(scheme 1)** $\hat{d}_{u \cap v} = \hat{d}_u + \hat{d}_v - \hat{d}_{u \cup v}$; **(scheme 2)** $\hat{d}_{u \cap v} = \hat{J}_{u,v} \hat{d}_{u \cup v}$; **(scheme 3)** $\hat{d}_{u \cap v} = \frac{\hat{J}_{u,v}}{\hat{J}_{u,v} + 1} (\hat{d}_u + \hat{d}_v)$. Similar to MinHash and HLL method [15], we initialize the parameter $\vec{\lambda}^{(0)} = (\lambda_u^{(0)}, \lambda_v^{(0)}, \lambda_{u,v}^{(0)})^T = (\hat{d}_u, \hat{d}_v, \hat{d}_{u \cap v})^T$ for our common-interest count estimation method in Section 5.3 and then run only a single Newton–Raphson iteration to obtain a more accurate estimation of $d_{u \cap v}$. In our experiments, common-interest count estimations given by OPH and HLL are computed by the above schemes 1, 2, and 3 in a direct manner.

In our experiments, by default, we let MinHash, OPH, and our sketch method OH use the same k , i.e., the number of registers used to generate one MinHash/OPH/OH sketch. We let k_1 denote the number of registers used for an HLL sketch. Therefore, our method, SimCar, uses k_1 fewer registers than baselines MinHash and HLL and OPH and HLL. Compared with MinHash, OPH, and OH using 32-bit registers, HLL uses 5-bit registers [5]; thus, our method, SimCar, reduces the memory usage by 13.5% and 23.8% when $k_1 = k$ and $k_1 = 2k$, respectively.

6.3. Metric

We evaluated both the efficiency and effectiveness of our method, SimCar, in comparison with the above two baseline methods. For efficiency, we evaluated the running time of all methods. Specially, we used the online sketching time to measure the time of online processing stream Γ . For user cardinality estimation, common-interest count estimation, and Jaccard similarity estimation, we evaluated the error of estimation $\hat{\mu}$ with respect to its true value μ using the normalized root mean square error (NRMSE), which is defined as $\text{NRMSE}(\hat{\mu}) = \sqrt{\text{MSE}(\hat{\mu})} / \mu$, where $\text{MSE}(\hat{\mu}) = \mathbb{E}((\hat{\mu} - \mu)^2) = \text{Var}(\hat{\mu}) + (\mathbb{E}(\hat{\mu}) - \mu)^2$. For the nearest neighbor search, given a query user, we report the average size of the retrieved user set, as well as the recall of the top 10 most similar users among that set. In our experiments, we computed the above metrics over 100 independent runs.

6.4. Runtime

As shown in Figure 2a, where we set $k_1 = k = 2^9$, our framework, SimCar, significantly outperforms MinHash and HLL and OPH and HLL on all datasets in Table 2. For example, for Reuters, the online sketching time of SimCar is about 2 seconds, while MinHash and HLL and OPH and HLL need about 764 and 5.3 seconds, respectively. On average, SimCar is about 400 and 2.5 times faster than MinHash and HLL and OPH and HLL, respectively. Figure 2b shows the online sketching time for different k . We can see that the online sketching time of SimCar and OPH+HLL is almost constant, while the online sketching time of MinHash and HLL increases linearly to k , because Minhash needs to update each of its k registers when updating each element that occurs in stream Γ .

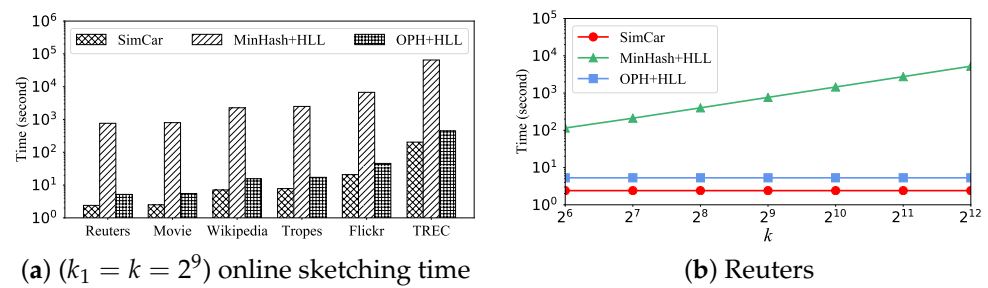


Figure 2. Online sketching time of our framework, SimCar, in comparison with MinHash and HLL and OPH and HLL. **(a)** Sketching time on different datasets with $k_1 = k = 2^9$. **(b)** Sketching time on dataset Reuters with different k .

6.5. Accuracy

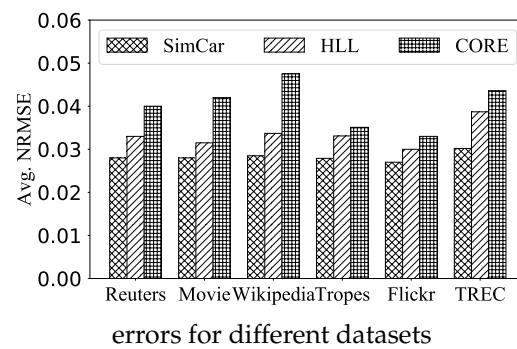
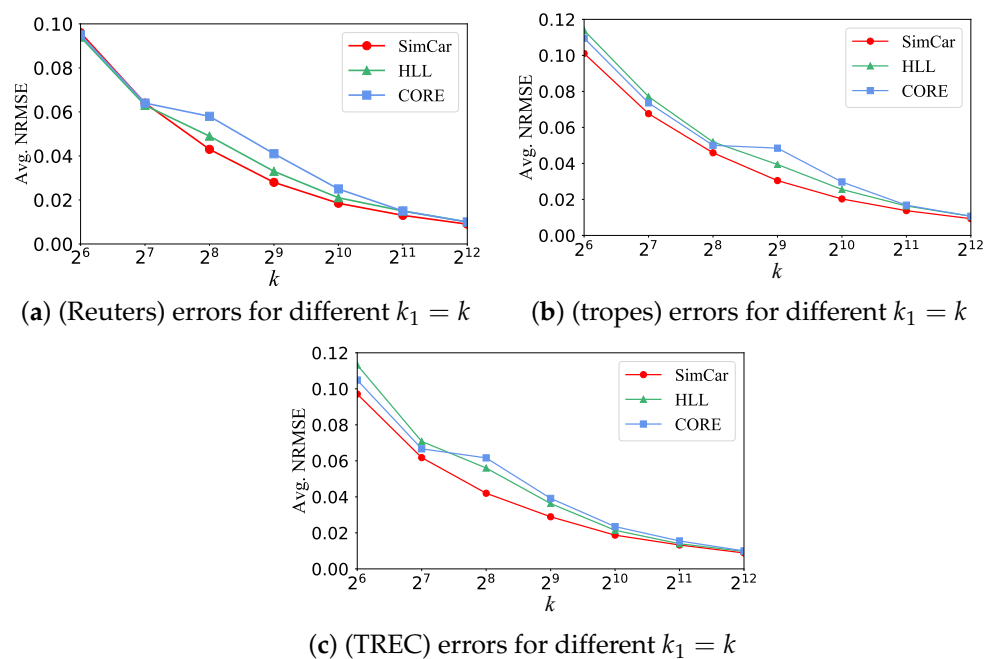
Results of cardinality estimation. We compared our method, SimCar, with HyperLogLog used by MinHash and HLL and OPH and HLL, as well as the CORE algorithm [59], which is introduced in Algorithm 3 in Section 5. In this experiment, all three sketch methods—HLL, CORE, and SimCar—use the same number of registers, i.e., $k_1 = k$. As shown in Figure 3, our method, SimCar, is significantly more accurate than HLL and CORE on these real-world datasets. To be more specific, SimCar gives a gain of 18%~40% (resp. 10%~23%) improvement to CORE (resp. HLL). For example, on Wikipedia, the average NRMSE of SimCar is 0.0285, while those of HLL and CORE are 0.0337 and 0.0475, respectively. Figure 4 further shows the average NRMSEs for different k . We can see that the average NRMSEs of all methods decrease as k increases, and our method is consistently more accurate than HLL and CORE. In addition, we compute the fine-grained NRMSEs with respect to the user cardinality ranging from 1 to 10,000 with $k = 2^8$ and $k = 2^9$. As shown in Figure 5, both HLL and CORE exhibit large estimation errors when user cardinality is small because they both use two different estimators for cardinalities within two different ranges, respectively. However, our method, SimCar, just uses one estimator, and we find that SimCar decreases the NRMSEs of HLL and CORE by 10% – 42% for different user cardinalities.

Results of common-interest count estimation. As shown in Figure 6a–c, where we set $k_1 = k$, we can see that our method, SimCar, is more accurate than the other two methods for all three schemes introduced in Section 6.2, and scheme 1 performs the worst among the three schemes, whose average NRMSEs are up to $8.7\times$ those of the other two schemes. We raised k_1 to $2k$, and the results are shown in Figure 6d–f. We can see that SimCar still outperforms MinHash and HLL and OPH and HLL for most of the datasets. Figure 7 show the average NRMSEs for different k on datasets Reuters, Tropes, and TREC, where we set $k_1 = k$. We can see that the average NRMSEs of all methods decrease as k increases. For scheme 1, our method, SimCar, is up to four and three times more accurate than MinHash and HLL and OPH and HLL, respectively. For schemes 2 and 3, we find that OPH and HLL gives results comparable to SimCar when k is medium. However, our method, SimCar, still outperforms OPH and HLL when $k \geq 2^{10}$.

Results of Jaccard similarity estimation and nearest neighbor search. We compared our method to the MinHash sketch and densified OPH sketch for Jaccard similarity estimation and nearest neighbor search. We used the technique discussed in Section 5.1 to build LSH for all three methods. The experimental results in Table 3 demonstrate that our method, SimCar, is comparable to MinHash and densified OPH on all the datasets.

Table 3. Performance of our framework, SimCar, compared with MinHash and OPH for Jaccard similarity estimation and nearest neighbor search with $k = 2^9$, $c = 2$.

	Jaccard Similarity Estimation			Top 10 Nearest Neighbor Search					
	NRMSE			Recall			the Number of Retrieved Users		
	SimCar	MinHash	OPH	SimCar	MinHash	OPH	SimCar	MinHash	OPH
Reuters	0.177	0.175	0.176	0.732	0.735	0.732	151	156	151
Movie	0.072	0.072	0.073	0.800	0.800	0.800	599	597	598
Wikipedia	0.085	0.085	0.085	0.799	0.800	0.800	2395	2398	2395
Tropes	0.168	0.171	0.170	0.555	0.561	0.556	220	222	220
Flickr	0.172	0.175	0.174	0.765	0.764	0.764	299	301	299
Actor	0.240	0.235	0.239	0.740	0.734	0.739	174	175	175
TREC	0.076	0.077	0.077	0.894	0.896	0.894	219	227	220

**Figure 3.** User cardinality estimation errors of our framework, SimCar, compared with HLL and CORE, $k_1 = k = 2^9$ by default.**Figure 4.** User cardinality estimation errors of our framework, SimCar, compared with HLL and CORE, $k_1 = k = 2^9$ by default.

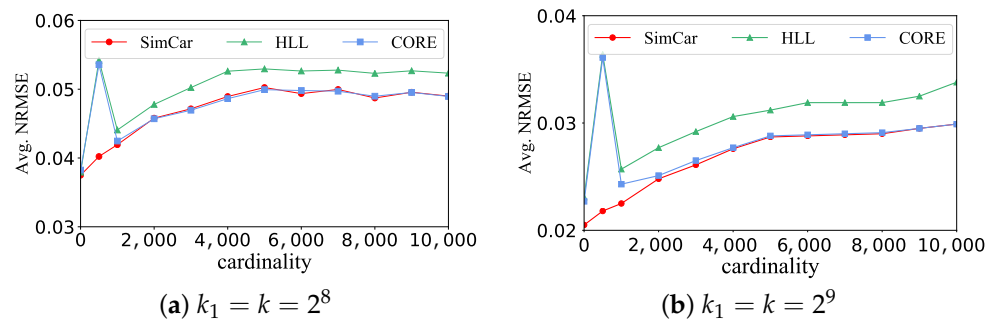


Figure 5. User cardinality estimation errors of our framework, SimCar, compared with HLL and CORE.

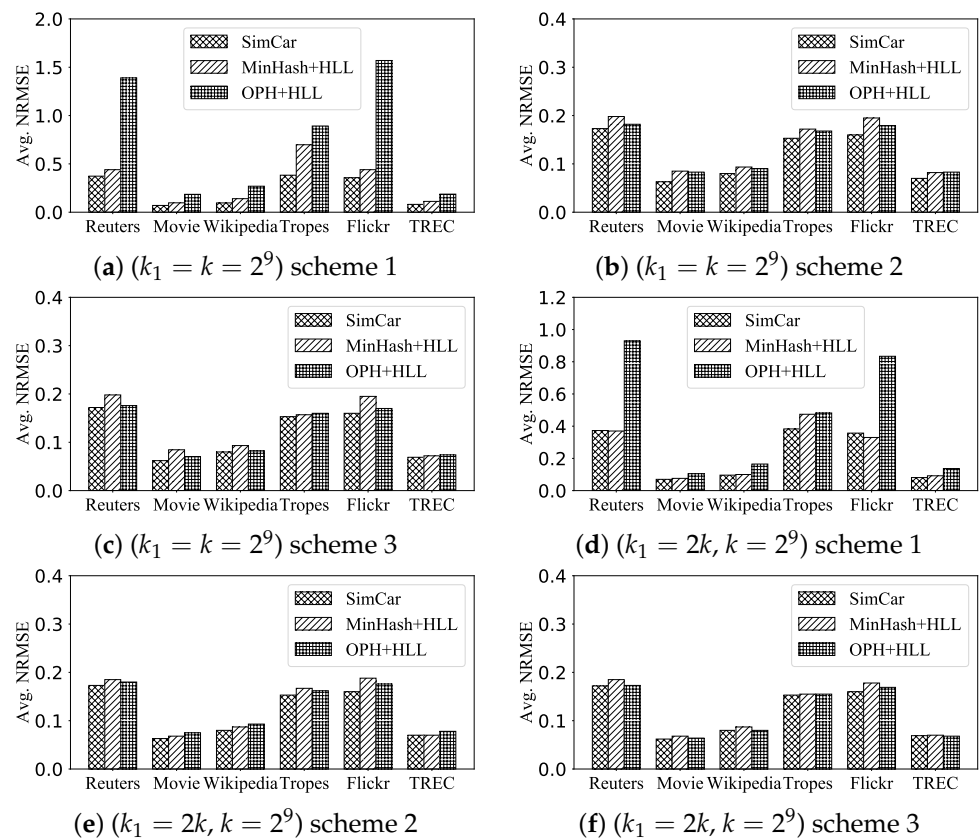


Figure 6. Average NRMSEs of SimCar compared with MinHash and HLL and OPH and HLL for estimating common-interest counts.

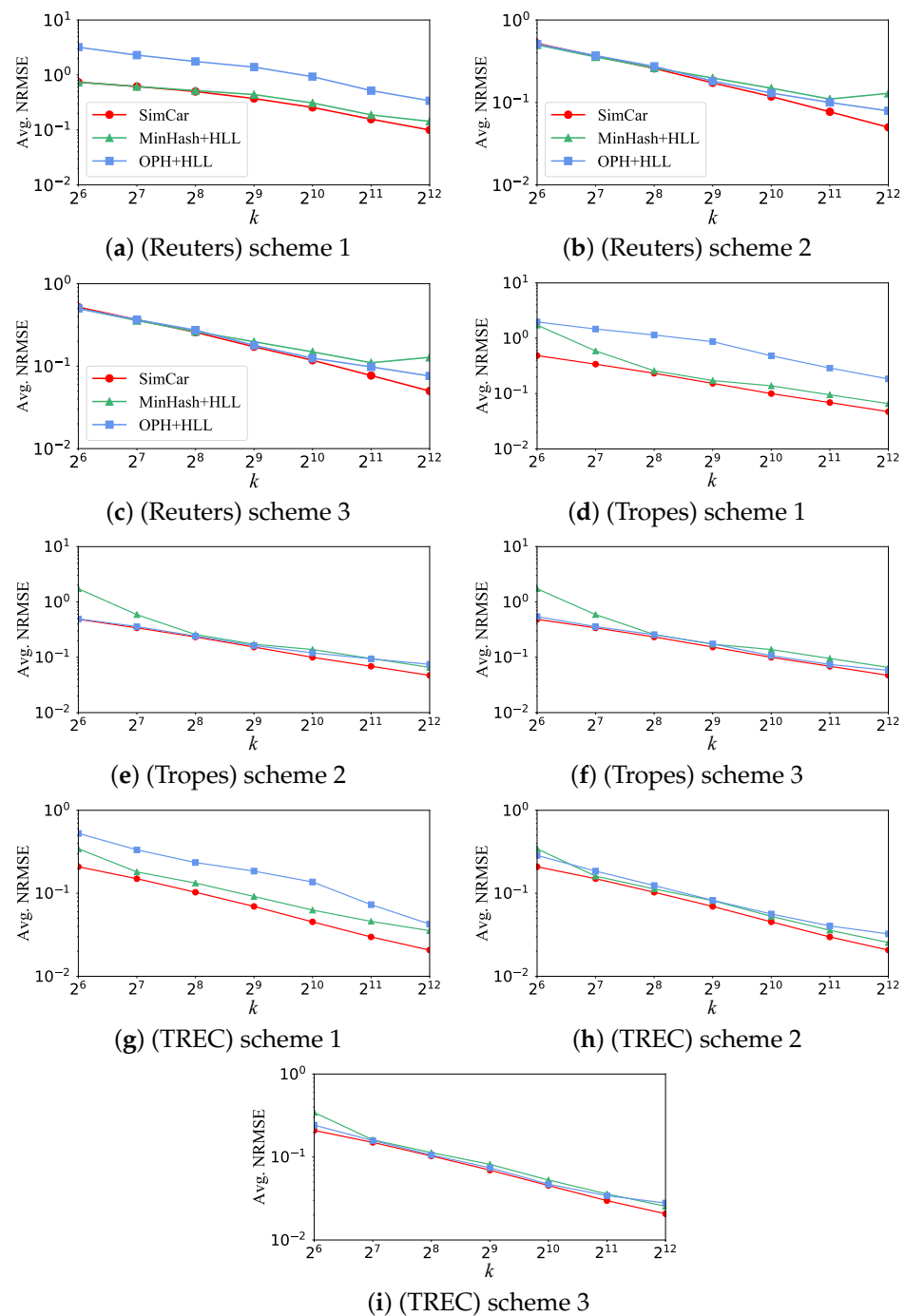


Figure 7. Average NRMSEs of SimCar compared with MinHash and HLL and OPH and HLL for estimating common-interest counts.

7. Conclusions and Future Work

In this paper, a framework—SimCar—was developed for mining users' interest sets. We built an OH sketch for each user that occurred in the data stream of interest, and one can query several mining results of users' interest sets at any time of the data stream. Specially, we developed accurate methods for estimating cardinalities, common-interest counts, and Jaccard similarities of users' interest sets. In addition, we used OH sketches to build LSH tables to quickly search for users with similar interests with sub-linear time. We evaluated the performance of our methods on real-world datasets. The experimental results demonstrated the effectiveness and efficacy of our methods. In future, we plan to

use techniques such as register sharing (i.e., compress the OH sketches of all users into a large register array) to further reduce the memory usage.

Author Contributions: Conceptualization, K.Y. and Y.Q.; methodology, K.Y. and Y.Q.; software, K.Y. and Y.Q.; validation, K.Y.; formal analysis, K.Y. and Y.Q.; investigation, K.Y. and Y.Q.; resources, K.Y. and P.J.; data curation, Y.Q.; writing—original draft preparation, K.Y.; writing—review and editing, W.G., Y.Q., P.W. and P.J.; visualization, K.Y.; supervision, W.G., P.W. and P.J.; project administration, W.G.; funding acquisition, W.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Key R&D Program of China (2021YFB1715600).

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: [<http://movielens.umn.edu/>; <http://en.wikipedia.org/wiki/Wikipedia>; <http://tvtropes.org>; <http://trec.nist.gov/data/docseng.html>].

Conflicts of Interest: The authors declare that they have no conflict of interest.

Appendix A

(1) **Formula of the gradient** $g(\vec{\lambda})$. Each element of $g(\vec{\lambda})$ is computed as

$$\begin{aligned}\frac{\partial \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_u} &= \frac{k_{u,v}^{(6)}}{\lambda_u} + \frac{k_{u,v}^{(3)} + k_{u,v}^{(5)}}{\lambda_u - \lambda_{u,v}} - \frac{\sum_{i=1}^k x_{u,i}}{k}, \\ \frac{\partial \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_v} &= \frac{k_{u,v}^{(5)}}{\lambda_v} + \frac{k_{u,v}^{(2)} + k_{u,v}^{(6)}}{\lambda_v - \lambda_{u,v}} - \frac{\sum_{i=1}^k x_{v,i}}{k}, \\ \frac{\partial \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_{u,v}} &= \frac{k_{u,v}^{(4)}}{\lambda_{u,v}} - \frac{k_{u,v}^{(3)} + k_{u,v}^{(5)}}{\lambda_u - \lambda_{u,v}} - \frac{k_{u,v}^{(2)} + k_{u,v}^{(6)}}{\lambda_v - \lambda_{u,v}} \\ &\quad + \frac{\sum_{i=1}^k \min(x_{u,i}, x_{v,i})}{k}.\end{aligned}$$

(2) **Formula of Hessian matrix** $H(\vec{\lambda})$. Each element of $H(\vec{\lambda})$ is computed as

$$\begin{aligned}\frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_u^2} &= -\frac{k_{u,v}^{(6)}}{\lambda_u^2} - \frac{k_{u,v}^{(3)} + k_{u,v}^{(5)}}{(\lambda_u - \lambda_{u,v})^2}, \\ \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_v^2} &= -\frac{k_{u,v}^{(5)}}{\lambda_v^2} - \frac{k_{u,v}^{(2)} + k_{u,v}^{(6)}}{(\lambda_v - \lambda_{u,v})^2}, \\ \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_{u,v}^2} &= -\frac{k_{u,v}^{(4)}}{\lambda_{u,v}^2} - \frac{k_{u,v}^{(3)} + k_{u,v}^{(5)}}{(\lambda_u - \lambda_{u,v})^2} - \frac{k_{u,v}^{(2)} + k_{u,v}^{(6)}}{(\lambda_v - \lambda_{u,v})^2}, \\ \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_u \partial \lambda_v} &= \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_v \partial \lambda_u} = 0, \\ \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_u \partial \lambda_{u,v}} &= \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_{u,v} \partial \lambda_u} = \frac{k_{u,v}^{(3)} + k_{u,v}^{(5)}}{(\lambda_u - \lambda_{u,v})^2}, \\ \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_v \partial \lambda_{u,v}} &= \frac{\partial^2 \log \mathcal{L}(\vec{x}_u, \vec{x}_v)}{\partial \lambda_{u,v} \partial \lambda_v} = \frac{k_{u,v}^{(2)} + k_{u,v}^{(6)}}{(\lambda_v - \lambda_{u,v})^2}.\end{aligned}$$

(3) **Formula of** $\mathbb{E}(H(\vec{\lambda}))$. From Equations (4)–(11), we derive expectations of variables $k_u, k_v, \mathbb{E}(k_{u,v}^{(1)}), \dots, \mathbb{E}(k_{u,v}^{(6)})$ as

$$\mathbb{E}(k_u) = ke^{-\frac{\lambda_u}{k}}, \quad \mathbb{E}(k_v) = ke^{-\frac{\lambda_v}{k}},$$

$$\begin{aligned}
\mathbb{E}(k_{u,v}^{(1)}) &= ke^{-\frac{\lambda_u + \lambda_v - \lambda_{u,v}}{k}}, \\
\mathbb{E}(k_{u,v}^{(2)}) &= k(e^{-\frac{\lambda_u}{k}} - e^{-\frac{\lambda_u + \lambda_v - \lambda_{u,v}}{k}}), \\
\mathbb{E}(k_{u,v}^{(3)}) &= k(e^{-\frac{\lambda_v}{k}} - e^{-\frac{\lambda_u + \lambda_v - \lambda_{u,v}}{k}}), \\
\mathbb{E}(k_{u,v}^{(4)}) &= \frac{k\lambda_{u,v}}{\lambda_u + \lambda_v - \lambda_{u,v}}(1 - e^{-\frac{\lambda_u + \lambda_v - \lambda_{u,v}}{k}}), \\
\mathbb{E}(k_{u,v}^{(5)}) &= \frac{k(\lambda_u - \lambda_{u,v})}{\lambda_u + \lambda_v - \lambda_{u,v}}(1 - e^{-\frac{\lambda_u + \lambda_v - \lambda_{u,v}}{k}}) - \mathbb{E}(k_{u,v}^{(3)}), \\
\mathbb{E}(k_{u,v}^{(6)}) &= \frac{k(\lambda_v - \lambda_{u,v})}{\lambda_u + \lambda_v - \lambda_{u,v}}(1 - e^{-\frac{\lambda_u + \lambda_v - \lambda_{u,v}}{k}}) - \mathbb{E}(k_{u,v}^{(2)}).
\end{aligned}$$

Based the above equations and the formula of $H(\vec{\lambda})$, we easily obtain the formula of each entry of matrix $\mathbb{E}(H(\vec{\lambda}))$.

References

1. Cormode, G.; Muthukrishnan, S. An Improved Data Stream Summary: The Count-min Sketch and Its Applications. *J. Algorithms* **2005**, *55*, 58–75. [\[CrossRef\]](#)
2. Estan, C.; Varghese, G.; Fisk, M. Bitmap algorithms for counting active flows on high speed links. In Proceedings of the SIGCOMM, Karlsruhe, Germany, 25–29 August 2003; pp. 182–209.
3. Whang, K.; Vander-zanden, B.T.; Taylor, H.M. A linear-time probabilistic counting algorithm for database applications. *IEEE Trans. Database Syst.* **1990**, *15*, 208–229. [\[CrossRef\]](#)
4. Durand, M.; Flajolet, P. *Loglog Counting of Large Cardinalities*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 605–617.
5. Flajolet, P.; Fusy, E.; Gandouet, O.; Meunier, F. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In Proceedings of the AOFA, Nice, France, 17–22 June 2007.
6. Giroire, F. Order statistics and estimating cardinalities of massive data sets. *Discret. Appl. Math.* **2009**, *157*, 406–427. [\[CrossRef\]](#)
7. Kane, D.M.; Nelson, J.; Woodruff, D.P. An Optimal Algorithm for the Distinct Elements Problem. In Proceedings of the PODS, Indianapolis, IN, USA, 6–11 June 2010; pp. 41–52.
8. Zhao, Q.; Kumar, A.; Xu, J. Joint data streaming and sampling techniques for detection of super sources and destinations. In Proceedings of the ACM SIGCOMM IMC 2005, Berkeley, CA, USA, 19–21 October 2005; pp. 77–90.
9. Yoon, M.; Li, T.; Chen, S.; Peir, J.K. Fit a spread estimator in small memory. In Proceedings of the IEEE INFOCOM 2009, Rio de Janeiro, Brazil, 19–25 April 2009; pp. 504–512.
10. Wang, P.; Guan, X.; Qin, T.; Huang, Q. A Data Streaming Method for Monitoring Host Connection Degrees of High-Speed Links. *IEEE Trans. Inf. Forensics Secur.* **2011**, *6*, 1086–1098. [\[CrossRef\]](#)
11. Xiao, Q.; Chen, S.; Chen, M.; Ling, Y. Hyper-Compact Virtual Estimators for Big Network Data Based on Register Sharing. In Proceedings of the SIGMETRICS, Portland, OR, USA, 15–19 June 2015; pp. 417–428.
12. Chen, A.; Cao, J.; Shepp, L.; Nguyen, T. Distinct counting with a self-learning bitmap. *J. Am. Stat. Assoc.* **2011**, *106*, 879–890. [\[CrossRef\]](#)
13. Ting, D. Towards Optimal Cardinality Estimation of Unions and Intersections with Sketches. In Proceedings of the SIGKDD, San Francisco, CA, USA, 13–17 August 2016.
14. Zhao, P.; Aggarwal, C.C.; He, G. Link prediction in graph streams. In Proceedings of the 32nd IEEE International Conference on Data Engineering, (ICDE 2016), Helsinki, Finland, 16–20 May 2016; pp. 553–564.
15. Cohen, R.; Katzir, L.; Yehezkel, A. A Minimal Variance Estimator for the Cardinality of Big Data Set Intersection. In Proceedings of the SIGKDD, Halifax, NS, Canada, 13–17 August 2017.
16. Broder, A.Z.; Charikar, M.; Frieze, A.M.; Mitzenmacher, M. Min-Wise Independent Permutations. *J. Comput. Syst. Sci.* **2000**, *60*, 630–659. [\[CrossRef\]](#)
17. Li, P.; Owen, A.B.; Zhang, C. One Permutation Hashing. In Proceedings of the NIPS, Lake Tahoe, NV, USA, 3–6 December 2012; pp. 3122–3130.
18. Li, P.; König, A.C. b-Bit minwise hashing. In Proceedings of the WWW, Raleigh, NC, USA, 26–30 April 2010; pp. 671–680.
19. Mitzenmacher, M.; Pagh, R.; Pham, N. Efficient estimation for high similarities using odd sketches. In Proceedings of the WWW, Doha, Qatar, 7–11 April 2014; pp. 109–118.
20. Shrivastava, A.; Li, P. Improved Densification of One Permutation Hashing. In Proceedings of the UAI, Quebec City, QC, Canada, 23–27 July 2014; pp. 732–741.
21. Shrivastava, A.; Li, P. Densifying One Permutation Hashing via Rotation for Fast Near Neighbor Search. In Proceedings of the ICML, Beijing, China, 21–26 June 2014; pp. 557–565.
22. Shrivastava, A. Optimal Densification for Fast and Accurate Minwise Hashing. In Proceedings of the ICML, Sydney, Australia, 6–11 August 2017; pp. 3154–3163.

23. Indyk, P.; Motwani, R. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In Proceedings of the STOC, Dallas, TX, USA, 23–26 May 1998; pp. 604–613.
24. Gionis, A.; Indyk, P.; Motwani, R. Similarity Search in High Dimensions via Hashing. In Proceedings of the PVLDB, Edinburgh, UK, 7–10 September 1999; pp. 518–529.
25. Charikar, M. Similarity estimation techniques from rounding algorithms. In Proceedings of the STOC, Montreal, QC, Canada, 19–21 May 2002; pp. 380–388.
26. Datar, M.; Immorlica, N.; Indyk, P.; Mirrokni, V.S. Locality-sensitive hashing scheme based on p-stable distributions. In Proceedings of the SOCG, Brooklyn, NY, USA, 8–11 June 2004; pp. 253–262.
27. Wang, P.; Qi, Y.; Zhang, Y.; Zhai, Q.; Wang, C.; Lui, J.C.S.; Guan, X. A Memory-Efficient Sketch Method for Estimating High Similarities in Streaming Sets. In Proceedings of the KDD, Anchorage, AK, USA, 4–8 August 2019; pp. 25–33.
28. Li, X.; Li, P. C-OPH: Improving the Accuracy of One Permutation Hashing (OPH) with Circulant Permutations. *arXiv* **2021**, arXiv:2111.09544.
29. Fernandez, R.C.; Min, J.; Nava, D.; Madden, S. Lazo: A Cardinality-Based Method for Coupled Estimation of Jaccard Similarity and Containment. In Proceedings of the 2019 IEEE 35th International Conference on Data Engineering (ICDE), Macao, China, 8–11 April 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1190–1201.
30. Manasse, M.; McSherry, F.; Talwar, K. *Consistent Weighted Sampling*; Technical Report; John Cappelen: Copenhagen, Denmark, 2010.
31. Haeupler, B.; Manasse, M.S.; Talwar, K. ling Made Fast, Small, and Easy. *CoRR* **2014**, abs/1410.4266.
32. Ioffe, S. Improved Consistent Sampling, Weighted Minhash and L1 Sketching. In Proceedings of the ICDM, Sydney, Australia, 13–17 December 2010; pp. 246–255.
33. Li, P. 0-Bit Consistent Weighted Sampling. In Proceedings of the KDD, Sydney, Australia, 10–13 August 2015; pp. 665–674.
34. Wu, W.; Li, B.; Chen, L.; Zhang, C. Canonical Consistent Weighted Sampling for Real-Value Weighted Min-Hash. In Proceedings of the ICDM, Barcelona, Spain, 12–15 December 2016; pp. 1287–1292.
35. Shrivastava, A. Simple and Efficient Weighted Minwise Hashing. In Proceedings of the NIPS, Barcelona, Spain, 5–10 December 2016; pp. 1498–1506.
36. Wu, W.; Li, B.; Chen, L.; Zhang, C. Consistent Weighted Sampling Made More Practical. In Proceedings of the WWW, Seville, Spain, 16–18 November 2017; pp. 1035–1043.
37. Ertl, O. BagMinHash-Minwise Hashing Algorithm for Weighted Sets. *CoRR* **2018**, abs/1802.03914.
38. Li, P.; Li, X.; Samorodnitsky, G.; Zhao, W. Consistent Sampling Through Extremal Process. In Proceedings of the Web Conference 2021, Ljubljana, Slovenia, 19–23 April 2021; pp. 1317–1327.
39. Moulton, R.; Jiang, Y. Maximally Consistent Sampling and the Jaccard Index of Probability Distributions. *arXiv* **2018**, arXiv:1809.04052.
40. Qi, Y.; Wang, P.; Zhang, Y.; Zhao, J.; Tian, G.; Guan, X. Fast Generating A Large Number of Gumbel-Max Variables. In Proceedings of the WWW, Taipei, Taiwan, 20–24 April 2020; pp. 796–807.
41. Panigrahy, R. Entropy based nearest neighbor search in high dimensions. In Proceedings of the SODA, Miami, FL, USA, 22–26 January 2006; pp. 1186–1195.
42. Lv, Q.; Josephson, W.; Wang, Z.; Charikar, M.; Li, K. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In Proceedings of the VLDB, Vienna, Austria, 23–27 September 2007; pp. 950–961.
43. Huang, Q.; Feng, J.; Zhang, Y.; Fang, Q.; Ng, W. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *PVLDB* **2015**, 9, 1–12. [[CrossRef](#)]
44. Gan, J.; Feng, J.; Fang, Q.; Ng, W. Locality-sensitive hashing scheme based on dynamic collision counting. In Proceedings of the SIGMOD, Scottsdale, AZ, USA, 20 May 2012; pp. 541–552.
45. Liu, Y.; Cui, J.; Huang, Z.; Li, H.; Shen, H.T. SK-LSH: An efficient index structure for approximate nearest neighbor search. *PVLDB* **2014**, 7, 745–756. [[CrossRef](#)]
46. Tao, Y.; Yi, K.; Sheng, C.; Kalnis, P. Quality and efficiency in high dimensional nearest neighbor search. In Proceedings of the SIGMOD, Providence, RI, USA, 29 June–2 July 2009; pp. 563–576.
47. Satuluri, V.; Parthasarathy, S. Bayesian locality sensitive hashing for fast similarity search. *PVLDB* **2012**, 5, 430–441. [[CrossRef](#)]
48. Gao, J.; Visvesvaraya Jagadish, H.; Lu, W.; Chin Ooi, B. DSH: Data Sensitive Hashing for high-dimensional k-NN search. In Proceedings of the SIGMOD, Snowbird, UT, USA, 22–27 June 2014.
49. ang, Y.; Shrivastava, A.; Ryu, J. Randomized Algorithms Accelerated over CPU-GPU for Ultra-High Dimensional Similarity Search. In Proceedings of the SIGMOD, Houston, TX, USA, 10–15 June 2018.
50. Ahle, T.D.; Pagh, R.; Razenshteyn, I.; Silvestri, F. On the complexity of inner product similarity join. In Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, San Francisco, CA, USA, 26 June–1 July 2016; pp. 151–164.
51. Neyshabur, B.; Srebro, N. On Symmetric and Asymmetric LSHs for Inner Product Search. In Proceedings of the International Conference on Machine Learning, Lille, France, 6 July–11 July 2015; pp. 1926–1934.
52. Shrivastava, A.; Li, P. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; pp. 2321–2329.

53. Ting, D. Streamed Approximate Counting of Distinct Elements: Beating Optimal Batch Methods. In Proceedings of the SIGKDD, New York, NY, USA, 24–27 August 2014; pp. 442–451.
54. Bachrach, Y.; Finkelstein, Y.; Gilad-Bachrach, R.; Katzir, L.; Koenigstein, N.; Nice, N.; Paquet, U. Speeding up the xbox recommender system using a euclidean transformation for inner-product spaces. In Proceedings of the 8th ACM Conference on Recommender Systems, Silicon Valley, CA, USA, 6 October 2014; pp. 257–264.
55. Ballard, G.; Kolda, T.G.; Pinar, A.; Seshadhri, C. Diamond sampling for approximate maximum all-pairs dot-product (MAD) search. In Proceedings of the ICDM, Atlantic City, NJ, USA, 14–17 November 2015; pp. 11–20.
56. Flajolet, P.; Martin, G.N. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci.* **1985**, *31*, 182–209. [\[CrossRef\]](#)
57. Xiao, Q.; Zhou, Y.; Chen, S. Better with fewer bits: Improving the performance of cardinality estimation of large data streams. In Proceedings of the INFOCOM, Atlanta, GA, USA, 1–4 May 2017; pp. 1–9.
58. Cohen, E.; Kaplan, H. Summarizing Data Using Bottom-k Sketches. In Proceedings of the PODC, Portland, OR, USA, 12–15 August 2007; pp. 225–234.
59. Lumbroso, J. An optimal cardinality estimation algorithm based on order statistics and its full analysis. In Proceedings of the AofA, Vienna, Austria, 28 June–2 July 2010; pp. 489–504.
60. Chen, W.; Liu, Y.; Guan, Y. Cardinality change-based early detection of large-scale cyber-attacks. In Proceedings of the INFOCOM, Turin, Italy, 14–19 April 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 1788–1796.
61. Flajolet, P. On Adaptive Sampling. *Computing* **1990**, *43*, 391–400. [\[CrossRef\]](#)
62. Gibbons, P.B. Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports. In Proceedings of the PVLDB, Roma, Italy, 11–14 September 2001; pp. 541–550.
63. Mao, Y.; Gan, D.; Mwakapesa, D.S.; Nanekharan, Y.A.; Tao, T.; Huang, X. A MapReduce-based K-means clustering algorithm. *J. Supercomput.* **2022**, *78*, 5181–5202. [\[CrossRef\]](#)
64. Corizzo, R.; Pio, G.; Ceci, M.; Malerba, D. DENCAST: Distributed density-based clustering for multi-target regression. *J. Big Data* **2019**, *6*, 1–27. [\[CrossRef\]](#)
65. Corizzo, R.; Dauphin, Y.; Bellinger, C.; Zdravevski, E.; Japkowicz, N. Explainable image analysis for decision support in medical healthcare. In Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 15–18 December 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 4667–4674.
66. Cao, M.; Jia, W.; Lv, Z.; Zheng, L.; Liu, X. Superpixel-Based Feature Tracking for Structure from Motion. *Appl. Sci.* **2019**, *9*, 2961. [\[CrossRef\]](#)
67. Ding, K.; Yang, Z.; Wang, Y.; Liu, Y. An improved perceptual hash algorithm based on u-net for the authentication of high-resolution remote sensing image. *Appl. Sci.* **2019**, *9*, 2972. [\[CrossRef\]](#)
68. Jacquet, P.; Szpankowski, W. Analytical Depoissonization and its Applications. *Theor. Comput. Sci.* **1998**, *201*, 1–62. [\[CrossRef\]](#)
69. Mitzenmacher, M.; Upfal, E. *Probability and Computing—Randomized Algorithms and Probabilistic Analysis*; Cambridge University Press: Cambridge, UK, 2005.
70. Bickel, P.J.; Doksum, K.A. *Mathematical Statistics: Basic Ideas and Selected Topics*, 2nd ed.; Prentice-Hall: Hoboken, NJ, USA, 2001; Volume 1.
71. Ypma, T.J. Historical Development of the Newton-Raphson Method. *SIAM Rev.* **1995**, *37*, 531–551. [\[CrossRef\]](#)
72. Mislove, A.; Marcon, M.; Gummadi, K.P.; Druschel, P.; Bhattacharjee, B. Measurement and analysis of online social networks. In Proceedings of the SIGCOMM, Kyoto, Japan, 27–31 August 2007; pp. 29–42.
73. GroupLens Research. MovieLens Data Sets. 2006. Available online: <http://www.grouplens.org/node/73> (accessed on 1 March 2022).
74. Wikimedia Foundation. Wikimedia Downloads. 2010. Available online: <http://dumps.wikimedia.org/> (accessed on 13 June 2021).
75. Lewis, D.D.; Yang, Y.; Rose, T.G.; Li, F. RCV1: A New Benchmark Collection for Text Categorization Research. *J. Mach. Learn. Res.* **2004**, *5*, 361–397.
76. Kunegis, J. KONECT: The Koblenz Network Collection. In Proceedings of the 22nd International Conference on World Wide Web, Rio de Janeiro, Brazil, 13–17 May 2013; pp. 1343–1350.
77. National Institute of Standards and Technology. Text REtrieval Conference (TREC) English Documents. 2010. Volume 4–5. Available online: http://trec.nist.gov/data/docs_eng.html (accessed on 11 June 2021).