"The 18th ACM International Conference on Web Search and Data Mining (WSDM'25)"

Full version

Maintaining k-MinHash Signatures over Fully-Dynamic Data Streams with Recovery

Andrea Clementi **Luciano Gualà** **Luca Pepè Sciarria** **Alessandro Straziota**

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Abstract

We consider the task of performing Jaccard similarity queries over a large collection of items that are dynamically updated according to a streaming input model. An item here is a subset of a large universe U of elements. A well-studied approach to address this important problem in data mining is to design *fast-similarity data sketches*. In this paper, we focus on *global solutions* for this problem, i.e., a single data structure which is able to answer both *Similarity Estimation* and *All-Candidate Pairs* queries, while also dynamically managing an arbitrary, online sequence of element insertions and deletions received in input.

We introduce and provide an in-depth analysis of a dynamic, buffered version of the well-known k-MinHash sketch. This buffered version better manages critical update operations thus significantly reducing the number of times the sketch needs to be rebuilt from scratch using expensive recovery queries. We prove that the *buffered* k-MinHash uses $O(k \log |U|)$ memory words per subset and that its *amortized* update time per insertion/deletion is $O(k \log |U|)$ *with high probability*. Moreover, our data structure can return the k-MinHash signature of any subset in $O(k)$ time, and this signature is exactly the same signature that would be computed from scratch (and thus the quality of the signature is the same as the one guaranteed by the static k-MinHash).

Analytical and experimental comparisons with the other, state-of-the-art global solutions for this problem given in [Bury et al.,WSDM'18] show that the *buffered* k-MinHash turns out to be competitive in a wide and relevant range of the online input parameters.

Problem Statements

Input. We are given a finite universe U and a collection of m sets $A_1, \ldots, A_m \subseteq U$.

Static problem. Design a data structure over the collection, i.e., a structure $\mathcal{S}(A_i)$ for each set A_i, that can efficiently answer the following queries:

Similarity Estimation (SE) queries: given two $\mathcal{S}(A_i), \mathcal{S}(A_j)$ estimate the Jaccard similarity $J(A_i, A_j)$;

All Candidate Pairs (ACP) queries: given a input parameter $r > 0$, returns all pairs of sets (A_i, A_j) with Jaccard similarity at least r.

Dynamic problem. Design a data structure that supports SE and ACP queries, while also allowing the following update operations:

`Insert`$(\mathcal{S}(A), x)$: given $\mathcal{S}(A)$ and an element x, compute $\mathcal{S}(A \cup \{x\})$;

`Delete`$(\mathcal{S}(A), x)$: given $\mathcal{S}(A)$ and an element x, compute $\mathcal{S}(A \setminus \{x\})$.

Goal. The data structure should guarantee:

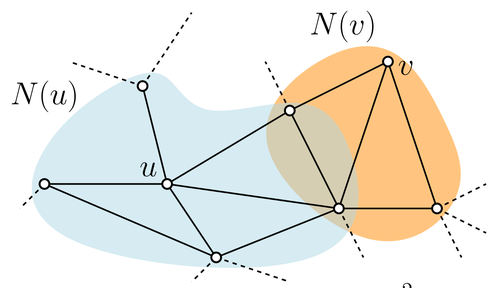
(i) sublinear space, i.e., PolyLog($|A_i|$) for each set A_i;

(ii) fast query time;

(iii) fast update time.

Use case: mining networks

A fundamental task in data mining is detecting "similar" nodes in a large network. Various works, i.e., [2,3,4], adopt Jaccard similarity between node's neighborhood as similarity measure: two nodes u, v are considered similar when the Jaccard similarity between $N(u)$ and $N(v)$ is high.



$$\mathrm{Sim}(u, v) = J\left(N(u), N(v)\right) = \frac{2}{8}$$

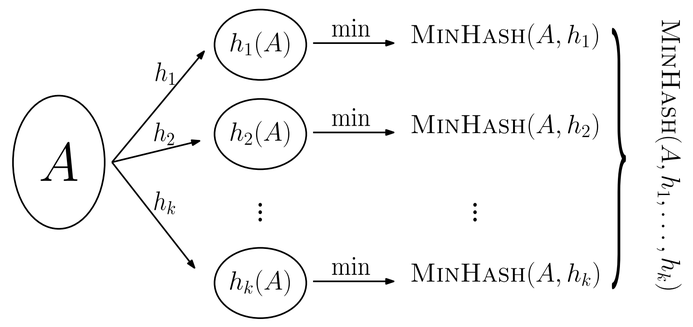
Challenge: dealing with dynamic graphs, where edges are inserted and removed.

k-MinHash Signature

Definition. Given k random hash function $h_1, \ldots, h_k : U \to U$ and a set $A \subseteq U$

$$\textsc{MinHash}(A, h_1, \ldots, h_k) := \langle \textsc{MinHash}(A, h_1), \ldots, \textsc{MinHash}(A, h_k) \rangle$$

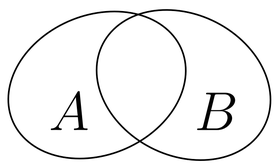
where $\textsc{MinHash}(A, h) = \min h(A)$.



Main Property.

$$P\left(\textsc{MinHash}(A, h_i) = \textsc{MinHash}(B, h_i)\right) = \frac{|A \cap B|}{|A \cup B|} = J(A, B)$$

Jaccard Similarity



$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \in [0, 1]$$

Jaccard Similarity Unbiased Estimator using k-MinHash

$$\widetilde{J}(A, B) = \frac{1}{k} \sum_{i=1}^{k} \mathbb{1}\{\textsc{MinHash}(A, h_i) = \textsc{MinHash}(B, h_i)\}$$

Main Property. The Jaccard similarity estimator is unbiased, i.e.,

$$\mathbb{E}\left[\widetilde{J}(A, B)\right] = J(A, B)$$

Data Stream with Recovery

Due to some problematic update operation, the k-MinHash signature may get out of sync, i.e. `delete`$(\mathcal{S}(A), a)$ where $a = \arg\min h_i(A)$ for some $1 \le i \le k$. In order to maintain the k-MinHash signature in a fully-dynamic envirnment we have to access the current state of the sets performing a *recovery query*. We call this streaming model *"Data Stream with Recovery"*.

State of the art

The BSS sketch [1] is the *only* analytical work presenting a sketch scheme for fully-dynamic streams that supports both SE and ACP queries with provably good performance.

Pro:

- works on pure stream model, i.e., no recovery queries are needed;
- fast update time.

Con:

- strong assumption on input stream (only legal streams supported);
- slow query time;
- bad accuracy.

To overcome the slow query time, the authors proposed a **proactive version** of the BSS sketch, which accelerates query processing at the cost of slower updates.

Our Solution: ℓ-*buffered* k-MinHash

We present the ℓ-*buffered* k-MinHash, a data structure that makes the static k-MinHash fully dynamic at a reasonable, logarithmic space and amortized update time overheads. This implies that, by incurring these small additional costs, the ℓ-*buffered* k-MinHash achieves the same accuracy in SE and ACP queries as the (static) k-MinHash.

Pro:

- same k-MinHash quality;
- supports arbitrary (even non-legal) input streams;
- fast update and query time.

Con:

- requires recovery queries;
- update time slower than BSS.

Main Result

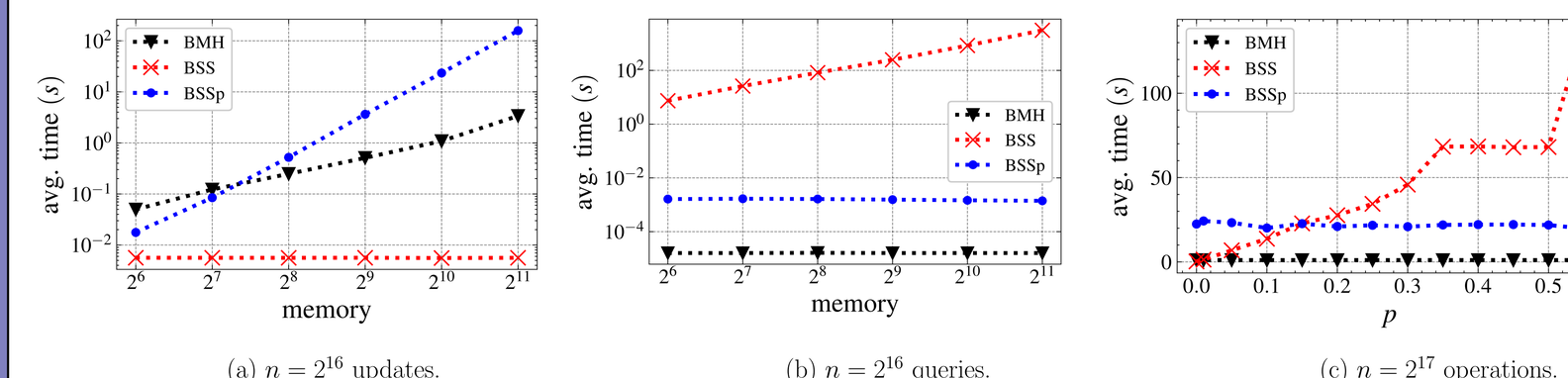
Main Theorem. The ℓ-buffered k-MinHash data structure is able to maintain the k-MinHash signatures of any fully-dynamic set collection of m sets over a universe U of size N with the following performance guarantees:

Memory usage: $O(k \log N)$ memory words per set;

Time: any sequence of update operations requires $O(k \log N)$ amortized time per operation with high probability w.r.t. the maximum size of the sets in the collection;

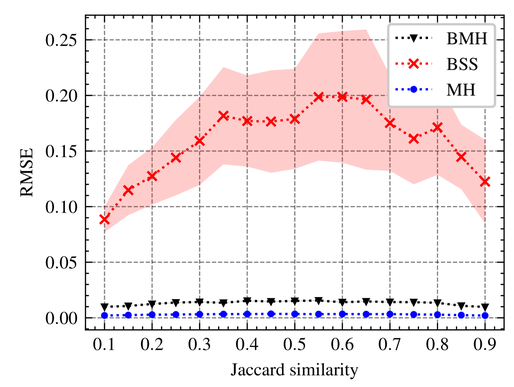
Response Quality: it can return the k-MinHash signature of each set of the collection in time $O(k)$.

Sketch	Space	Update	Query
k-MinHash	$O(k)$	$O(k\|A\|)$	$O(k)$
BSS	$O(c^2 \log N)$	$O(1)$	$O(c^2 k)$
BSS Proactive	$O(c^2 \log N)$	$O(c^2 k)$	$O(k)$
ℓ-buffered k-MinHash	$O(k \log N)$	$O^*(k \log N)$	$O(k)$



(a) $n = 2^{16}$ updates. (b) $n = 2^{16}$ queries. (c) $n = 2^{17}$ operations.

(a) comparison, as memory changes, of $n = 2^{16}$ insertions followed by n deletions. (b) comparison, as memory changes, of $n = 2^{16}$ queries. (c) time comparison for a sequence of $n = 2^{17}$ operations varying the fraction p of query, for 1024×17 memory words.



RMSE for different values of Jaccard similarity, from 0.1 to 0.9. The size of the sketches is $1024 \times \log |U|$ memory words, where $|U| = 2^{20}$. Thus static MinHash (MH in the figure) uses $k \times \log |U| = 1024 \times 17$ hash functions. The size of the sets are ≈ 6500. Each point is the RMSE for 1000 experiments. The shaded areas represent the standard deviation of the experiments.

n	static	ℓ-buffered	speedup
2^{12}	$96.376s$	$0.43s$	224x
2^{16}	$2049.953s$	$2.981s$	687x
2^{19}	$15279.49s$	$20.505s$	745x

Average running time comparison in executing $2n$ operations between static and ℓ-*buffered* k-MinHsah, for $k = 2000$ hash functions.

		Sketch	J	b	r	Precision	Recall	F_1	Effective Pairs	Returned Pairs
$d = 1$	`LiveJournal`	BMH	0.1	700	3	**0.808**	**0.955**	**0.875**	113, 117	133, 671
		BSS		420	5	0.439	0.753	0.555		193, 873
	`Orkut`	BMH	0.1	700	3	0.231	**0.752**	0.354	12, 604	40, 963
		BSS		350	6	**0.74**	0.4	**0.52**		6, 814
	`YouTube`	BMH	0.05	400	2	**0.247**	**0.832**	**0.382**	23, 763	79, 852
		BSS		160	5	0.13	0.30	0.18		53, 439
$d = 2$	`LiveJournal`	BMH	0.35	150	5	**0.254**	**0.849**	**0.391**	219, 826	735, 052
		BSS		125	6	0.176	0.449	0.253		561, 368
	`Orkut`	BMH	0.4	150	6	**0.239**	**0.63**	**0.346**	182, 285	490, 682
		BSS		150	6	0.232	0.303	0.263		243, 158
	`YouTube`	BMH	0.1	700	3	0.67	**0.38**	**0.485**	470, 216	587, 936
		BSS		420	5	**0.698**	0.135	0.227		91, 358

Our `C++` implementation on
`https://github.com/Alessandrostr95/DynamicMinHash`

[1] Marc Bury, Chris Schwiegelshohn, and Mara Sorella. Similarity search for dynamic data streams. IEEE Transactions on Knowledge and Data Engineering, 32:22412253, 2020.

[2] Peng Jia, Pinghui Wang, Jing Tao, and Xiaohong Guan. A fast sketch method for mining user similarities over fully dynamic graph streams. In 2019 IEEE 35th International Conference on Data Engineering (ICDE), pages 16821685, 2019.

[3] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive data sets. Cambridge university press, 2020.

[4] Qingjun Xiao, Shiwei Yang, Panpan Li, Kangying Li, and Lin Wen. Multi-resolution odd sketch for mining extended jaccard similarity of dynamic streaming sets. IEEE Transactions on Network Science and Engineering, pages 115, 2023.