TABLE 1
Distribution of Exact Similarity Values for Pairs in Last.fm Dataset

Similarity	0.0	0.05	0.1	0.15
No. of pairs	60432710	37947485	12031795	1938117
Similarity	0.2	0.25	0.3	0.35
No. of pairs	164246	7855	266	13
Similarity	0.4	0.45	0.5	≥ 0.55
No. of pairs	10	1	3	0

(0.45, 0.55) we deleted an item contained in row i with probability 1/3 and added a new item with probability $\frac{1}{19} \cdot \frac{1}{3} = \frac{1}{57}$. In the insertion-only case, the stream consists of the sequence of 1-entries of each row. We introduced deletions by randomly removing any non-zero entry immediately after insertion with probability $\frac{1}{10}$.

Last.FM Dataset. For an evaluation of our algorithm on real data we considered a dataset from [40] containing temporal data from the popular online (social) music recommendation system Last.fm. Users update their profiles in multiple ways: listening to their personal music collection with a music player supporting the Last.fm Audioscrobbler plugin, or by listening to the Last.fm Radio service, either with Last.fm official client application, or with the web player. Radio stations consist of uninterrupted audio streams of individual tracks based on the user's profile, its "musical neighbors" (i. e., similar users identified by the platform), or the user's "friends". All songs played are added to a log from which personalized top charts and musical recommendations are calculated, using a collaborative filtering algorithm. This automated track logging process is called *scrobbling*. Our dataset contains the full "scrobbled" listening history of a set of 44,154 users, covering a period of 5-years (May 2009-May 2014), containing 721M listening events and around 4.6M unique tracks, where each track is labeled with a score for a set of 700 music genres. To obtain a more granular feature space, we decided to map each track to the corresponding artist. To this end we queried the MusicBrainz DB[2] to obtain artist information for each of the unique tracks (total of 1.2M artists). We then processed the listening histories of each user i in chronological order to

$n = 15K$ users (sets), $|U| = 380K$ (items) and a stream length of 6.2M entries. Table 1 shows the distribution of exact similarity values for all pairs the Last.fm dataset.

6.1 Performance Evaluation

We evaluated the running time of our algorithm using the synthetic dataset, to understand its performance with respect to various dataset sizes, in two different scenarios, an insertion-only stream, and a *fully dynamic stream*, both obtained from our synthetic dataset. As a comparative benchmark, we compare our approach with an online implementation of a "vanilla" LSH scheme (later *Vanilla-MH*), where profile sketches are computed online using 2-wise independent hash functions (that is also our signature scheme).

We tested two versions of our algorithm. The first version henceforth called *DynSymSearch* (DSS) maintains the sketches of Algorithm 1 and computes fingerprints only at query time. The second, called *DynSymSearch Proactive* (or simply DSS *Proactive*), instead maintains a set of fingerprints online, with every update (that is, after line 5 of Algorithm 1, reflecting the most recent change from the stream).

The choice of the first or the second implementation depends on the use case, with a trade-off between query responsiveness and additional space required for computing and storing the signatures of sets.

Let us now focus on the algorithms that update signatures online. When inserting item j added to set i, both DSS *Proactive* and *Vanilla-MH* behave in the same way. When an element is added, all hash-functions are evaluated on the new element, and updated in case such value is the new minimum. Let $k = lsb(h(j))$. In case of deletions, both will have to recompute signatures, yet while *Vanilla-MH* has to do so for the full user profile, DSS *Proactive* has to recompute signatures only for the two compressed bucket sets $T_i^{k,\bullet}$ and $T_i^{0,\bullet}$. A further optimization that we implemented in DSS *Proactive*, is the *selective recomputation* of signatures in case of deletions. In case of deletions of an item j, we recompute a set of signatures for $T_{(i)}^{k}$ only if the bucket is *sensitive*, i. e., its corresponding set cardinality and similarity threshold are such that k is the range specified by line 5 of Algorithm 2. This allows to ignore many costly recomputations