A Two-Level Signature Scheme for Stable Set Similarity Joins

Daniel Schmitt
University of Salzburg, Austria
danielulrich.schmitt@plus.ac.at

Daniel Kocher
University of Salzburg, Austria
dkocher@cs.sbg.ac.at

Nikolaus Augsten
University of Salzburg, Austria
nikolaus.augsten@plus.ac.at

Willi Mann
Celonis SE, Germany
w.mann@celonis.com

Alexander Miller
University of Salzburg, Austria
alexander.miller@cs.uni-salzburg.at

ABSTRACT

We study the *set similarity join problem*, which retrieves all pairs of similar sets from two collections of sets for a given distance function. Existing exact solutions employ a signature-based filter-verification framework: If two sets are similar, they must have at least one signature in common, otherwise they can be pruned safely. We observe that the choice of the signature scheme has a significant impact on the performance. Unfortunately, choosing a good signature scheme is hard because the performance heavily depends on the characteristics of the underlying dataset.

To address this problem, we propose a *hybrid signature composition* that leverages the most selective portion of each signature scheme. Sets with an unselective primary signature are detected, and the signatures are replaced with a more selective secondary signature. We propose a generic framework called *TwoL* and a cost model to balance the computational overhead and the selectivity of the signature schemes. We implement our framework with two complementary signature schemes for Jaccard similarity and Hamming distance, resulting in effective two-level *hybrid indexes* that join datasets with diverse characteristics efficiently. TwoL consistently outperforms state-of-the-art set similarity joins on a benchmark with 13 datasets that cover a wide range of data characteristics.

PVLDB Reference Format:
Daniel Schmitt, Daniel Kocher, Nikolaus Augsten, Willi Mann, and Alexander Miller. A Two-Level Signature Scheme for Stable Set Similarity Joins. PVLDB, 16(11): 2686 - 2698, 2023.
doi:10.14778/3611479.3611480

1 INTRODUCTION

Similarity queries and joins based on set-valued objects have become increasingly popular in various domains [14] with different applications, for example, near-duplicate detection [26, 30], recommender systems [3], and query refinement for search engines [2]. In order to find all pairs of similar sets in two large collection of sets, R and S, different set similarity join algorithms have been proposed. A distance function $d(.,.)$ is used to assess whether two sets $r \in R$ and $s \in S$ are similar w.r.t. a user-defined threshold ϵ, i.e., the join result contains all pairs $(r, s) \in R \times S$ for which $d(r, s) \leq \epsilon$ holds.

Proceedings of the VLDB Endowment, Vol. 16, No. 11 ISSN 2150-8097.
doi:10.14778/3611479.3611480

Current set similarity join algorithms employ a signature-based filter-verification framework to join two collections of sets, R and S. A *signature scheme* is used to represent each set by one or more signatures: if two sets are similar, they must have at least one signature in common. Signatures are used in two ways: (i) The signatures of the sets $s \in S$ are *indexed* in an inverted list index. (ii) *Probing* the signatures of a set $r \in R$ against the index yields candidates. A candidate s shares at least one signature with r. *Verification* eliminates false positive candidates by computing the true similarity of the probing set r to each candidate s.

Prior Work. The signatures proposed in literature show very different performance depending on the characteristics of the joined datasets and the required similarity threshold, and there is no single signature that performs best in all settings. Relevant data characteristics include, for example, the frequency distribution of the set elements (called *tokens*), the number of distinct tokens in the sets (the *universe* size), and the size of the individual sets in the collection. The overall performance of a signature does not only depend on its selectivity but also on the overhead for indexing and probing.

The predominant signature schemes are either based (i) on a variation of the prefix filter [5] or use (ii) a partition-and-enumeration framework [1]. (i) The *prefix filter* examines the first π tokens of each set w.r.t. a global token order. Each prefix token is a signature, and sets without a common prefix token are pruned. Prefix-based approaches [3, 8, 14, 26, 28, 30] typically perform well on datasets with a large universe, sparsely populated set vector representations (any vector $x \in \{0, 1\}^d$ can be represented as a set $\{i \mid x_i = 1\}$), and a heavily skewed token distribution (i.e., enough infrequent tokens). (ii) *Partition-and-enumeration* frameworks split each set into disjoint partitions (by partitioning the universe) and enumerate all subsets within Hamming distance $\epsilon' \leq \epsilon$ for each partition to form the signatures; ϵ' is either fixed for all partitions or is computed using a cost model. Partition-enumeration frameworks [1, 7, 16, 17, 22, 31] incur a larger overhead than prefix-based techniques and require more memory. However, on datasets with a smaller universe or more uniformly distributed tokens, they may be much more selective and outperform prefix-based approaches.

Limitations. Choosing a good signature scheme is hard, and unfavorable signatures may lead to slowdowns of more than an order of magnitude or excessive memory footprints. A small universe or uniform token distributions render the prefix filter ineffective, resulting in expensive index lookups and many candidates. Partition-and-enumeration techniques are ill-suited for datasets that favor the prefix filter because the enumeration is expensive (due to a large universe) and sparsely populated binary vectors result in unselective partitions. Techniques to avoid unselective partitions [7, 22] incur high computational overhead and are often slower than the lightweight prefix filter [14]. Both approaches perform poorly on datasets that show mixed characteristics.

A *direct composition* of two signature schemes requires candidate pairs to share at least one signature in both schemes: The number of candidates is reduced at the cost of more expensive computations.

Our Solution & Contributions. We propose the *TwoL* framework (pronounced [tu:l]) for set similarity joins, which uses a novel *hybrid signature composition* technique to integrate signatures that are optimized for different dataset characteristics. TwoL employs a *hybrid index* that maintains two levels, one for each signature scheme. Our goal is to use the *most selective* signatures of each scheme, i.e., the signatures that generate the smallest number of candidates. If a *primary signature* at level 1 is not selective enough, we transfer the corresponding index entry to level 2, i.e., the affected sets are *reindexed* with the *secondary signature* scheme. TwoL uses a cost model to decide if migrating an index entry is beneficial. The cost model can be configured to *interpolate* between the two signature schemes and their direct composition. Our new hybrid signature composition gradually assumes properties of the primary scheme, the secondary scheme, and their direct composition.

We further introduce two novel optimizations for the self-join scenario that can be used independently of TwoL: (i) *EPEL* provides a tighter length bound on candidate sets than the standard PEL [13] filter. (ii) *Index filtering* avoids the creation of trivial singleton inverted lists, which substantially reduces the memory footprint of selective signatures, e.g., up to 99% of the fcLSH [20] lists are trivial.

In summary, our contributions are the following:

- We propose the *hybrid signature composition* for set similarity joins that enables "interpolation" between two signature schemes: our composition gradually assumes the properties of one of the two schemes and their direct composition.
- Our *TwoL* framework indexes hybrid compositions of two complementary signature schemes. We implement TwoL for the Jaccard similarity and the Hamming distance.
- A novel *cost model* allocates the two levels of our hybrid index. The optimal index allocation is shown to be computationally infeasible and we propose two heuristic strategies.
- For self-joins, we introduce *EPEL* to tighten the position-enhanced length filter, and *index filtering* that substantially reduces the memory cost for indexing selective signatures.
- In our experiments on 13 datasets with a wide range of characteristics, TwoL outperforms its competitors and is less sensitive to dataset characteristics even for datasets that do not favor one of the two selected signature schemes.

Outline. We discuss related work and preliminaries in Section 2 and 3, respectively. Section 4 generalizes prior set similarity techniques as different signature compositions, Section 5 proposes a new hybrid composition, and Section 6 discusses the index allocation cost. We instantiate TwoL with concrete signatures in Section 7, show experimental results in Section 8, and conclude in Section 9.

2 RELATED WORK

Most set similarity join algorithms follow the filter-verification framework: a filter generates candidates of possibly similar sets that must be verified. The two dominant filter categories are the prefix filter [5] and the partition-and-enumeration framework [1], and both apply additional filtering conditions like the set size [1].

Prefix Filter. AllPairs [2] uses both prefix and length filter in a main-memory setting. PPJoin(+) [30] extend AllPairs by utilizing positional information and suffix filtering. GroupJoin [3] extends PPJoin and groups sets with the same prefix during indexing and candidate generation to filter multiple sets at once. Numerous variations of prefix-filter algorithms [2, 3, 13, 26, 30] have been proposed and we refer to the survey by Mann et al. [14] for details. In their empirical evaluation, GroupJoin is the most robust algorithm with the smallest average and maximum gap factor compared to the winner in a given setting. SizeAware [8] uses overlap similarity and splits a dataset into small and large sets. Large sets are handled with ScanCount [11]. For small sets, subsets of size c (c-*subsets*) are used as signatures, but skipping and deduplication methods avoid enumerating all signatures. For other similarity functions like Jaccard, a prefix-filter based construction using AdaptJoin [26] is used. Rong et al. [23] use multiple prefix indexes with different token orderings to increase pruning effectiveness in a distributed setting. Wang et al. [28] propose SkipJoin and use grouping in the prefix index to build skippable blocks using the positional filter [30]. Additionally, given all similar sets for r, and a set s similar to r, a cost model either decides to compute the candidates of s from scratch or to leverage the similarity of s to r. Algorithms in this category typically perform well on heavily skewed datasets with a large universe (i.e., enough infrequent tokens). The prefix filter was also used in the context of top-k set similarity joins [29, 32], where the goal is to find the k pairs of sets with the highest similarity. Our goal is to solve the threshold-based set similarity join problem.

Partition-and-Enumeration. Arasu et al. [1] introduce the partition-and-enumeration framework. In a nutshell, algorithms in this framework split each set (or vector) into disjoint partitions $P_1, \ldots, P_n$ and enumerate all partitions within Hamming distance ϵ_k for each partition $P_k, 1 \leq k \leq n$. Two sets that agree on a partition form a candidate pair. Arasu et al. [1] use two tunable parameters for the partition count and the number of enumeration signatures. Norouzi et al. [16, 17] only enumerate on the query side and use a tight filtering condition. Zhang et al. [31] use an efficient verification algorithm for bit vectors and balance rare and common tokens using a dimension rearrangement method. In PartAlloc[1], Deng et al. [7] use a cost model to allocate $\epsilon_k \in \{-1, 0, 1\}$ efficiently. For the Hamming distance, sets are partitioned in $\epsilon + 1$ parts. Qin et al. [22] generalize ϵ_k allocations and propose a cost model and an uneven partitioning scheme. Liu et al. [12] generalize partitioning to allow overlaps and use machine learning for candidate estimations. Qin et al. [21] generalize the pigeonhole principle by considering chains of partitions and extend pkwise [27] for set similarity search with overlap constraint. Pagh's CoveringLSH [18] constructs bitmaps that cover all possible ways similar sets might differ and achieves total recall, whereas other LSH methods are approximations. fcLSH by Pham et al. [20] increases CoveringLSH's hashing performance by using the Fast Hadamard Transform. Algorithms in this category usually perform well on datasets with dense bitvectors of low dimensionality, where the prefix filter is ineffective. For skewed

[1]PartAlloc is also known as Greedy+ or PTJoin.

token distributions, however, the sophisticated filters and the large memory footprint of these algorithms do not pay off.

Our hybrid signature composition is orthogonal to the development of new signature schemes: New signatures can replace less efficient ones in our hybrid composition and the TwoL framework.

Approximate Methods. Compared to exact set similarity join techniques, algorithms based on Locality-Sensitive Hashing (LSH) [9] or Mapping (LSM) [6] generally do not find all similar pairs, but allow time-recall trade-offs. Recent advances include approximate candidate pruning [4], LSM families with better query times compared to predecessors [6], and efficient implementations thereof [15]. In this paper, we focus on exact set similarity joins.

3 PRELIMINARIES

The core of a set similarity join algorithm is the signature scheme that is used to prune dissimilar pairs of sets. In this section, we define our problem of interest, formally introduce signature schemes, revisit signature-based set similarity joins, and specify our goal.

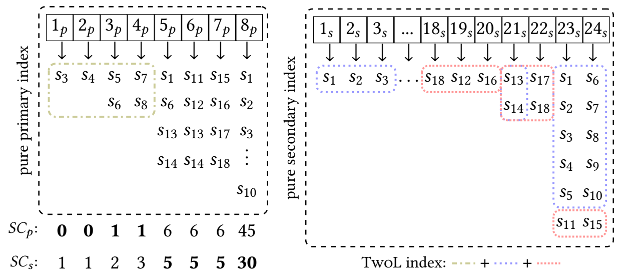
Set Similarity Joins and Signature Schemes. For two collections of sets R and S, a set similarity join reports all pairs of sets that are similar w.r.t. some distance function. A set r has n unique *tokens*, $|r| = n$ denotes the *size* of r, sets in R can differ in size, and the *token universe* of R is the set of all tokens over all sets in R. In this work, we focus on the Hamming distance as a dissimilarity measure; we discuss the extension to other distances in Section 7.6. The Hamming distance of two sets r and s is the number of tokens that only appear in one set, i.e., $r \triangle s = |r \cup s| - |r \cap s|$. Two sets are similar if their Hamming distance is within a given distance threshold ϵ. The set similarity join of R and S using the Hamming distance is defined as $R \tilde{\bowtie} S = \{(r, s) \in R \times S \mid r \triangle s \le \epsilon\}$. We study self joins, i.e., $R = S$, but an extension to foreign joins is straightforward.

Arasu et al. [1] used the notion of signatures as a conceptual framework to compare different set similarity join algorithms. A *signature* can be interpreted as a hash value for a set, and existing set similarity joins primarily differ in the way they generate signatures, i.e., their signature scheme. For a set r, a *signature scheme* generates one or more signatures, and $Sign(r)$ denotes the finite *set of signatures* of r. Moreover, a signature scheme must guarantee that any two similar sets r and s share at least one signature, i.e., $r \triangle s \le \epsilon \Rightarrow Sign(r) \cap Sign(s) \neq \emptyset$, where the distance threshold ϵ is a hidden parameter of *Sign*. *Asymmetric schemes* [7, 22, 30] construct different signatures for sets that are stored in an index and sets that are used to probe the index. We denote them as $Sign^I(r)$ and $Sign^P(r)$, respectively. Superscripts I and P are omitted if they are obvious from the context, the signature scheme is symmetric, or an extension to asymmetric signature schemes is straightforward.

Algorithmic Framework. To avoid computing $Sign^I(r) \cap Sign^P(s)$ for all set pairs, an inverted list index L^I is built on $Sign^I$. List L^I_t stores all sets r with $t \in Sign^I(r)$ and L^P_t denotes all sets containing the probing signature t. When set s is probed, we generate all signatures $Sign^P(s)$ and look up the lists L^I_t of all probing signatures $t \in Sign^P(s)$. All sets in these lists form a candidate pair with s and undergo verification. We use the term *signature* for the signature itself and its lists. In self joins, sets are typically processed in increasing size order. Algorithm 1 shows the corresp. framework.

Algorithm 1: Signature-based Framework

```
Input: Collection $R$, distance $\epsilon$
Result: All similar pairs in $R \times R$
$L^I \leftarrow \emptyset, M \leftarrow \emptyset, C \leftarrow \emptyset$        // inv. index, result, candidates
forall $r \in R$ do                                  // indexing
    forall signatures $t \in Sign^I(r)$ do $L^I_t \leftarrow L^I_t \cup \{r\}$
forall $s \in R$ do                                  // probing
    forall signatures $t \in Sign^P(s)$ do $C \leftarrow C \cup \{(r, s) \mid r \in L^I_t\}$
forall candidate pairs $(r, s) \in C$ do             // verification
    $M \leftarrow M \cup (r, s)$ if $r \triangle s \le \epsilon$
return $M$
```


pure primary index
1_p 2_p 3_p 4_p 5_p 6_p 7_p 8_p
s_3 s_4 s_5 s_7 s_1 s_{11} s_{15} s_1
s_6 s_8 s_6 s_{12} s_{16} s_2
s_{13} s_{13} s_{17} s_3
s_{14} s_{14} s_{18} ⋮
s_{10}
SC_p: 0 0 1 1 6 6 6 45
SC_s: 1 1 2 3 5 5 5 30
pure secondary index
1_s 2_s 3_s ... 18_s 19_s 20_s 21_s 22_s 23_s 24_s
s_1 s_2 s_3 . . . s_{18} s_{12} s_{16} s_{13} s_{17} s_1 s_6
s_{14} s_{18} s_2 s_7
s_3 s_8
s_4 s_9
s_5 s_{10}
s_{11} s_{15}
TwoL index: + +

(a) $Sign_p$ **(b)** $Sign_s$

Figure 1: Running example.

Example 3.1. Consider the self join of $R = \{s_1, s_2, \ldots, s_{18}\}$. We probe set s_3 in the inverted index of the symmetric signature $Sign_p$ in Figure 1(a). To avoid symmetric pairs (s_j, s_i) and reflexive pairs (s_i, s_i), we only consider candidate pairs (s_i, s_j) with $i < j$. With $Sign_p(s_3) = \{1_p, 8_p\}$, we get 0 non-reflexive, non-symmetric candidates from list 1_p and 9 candidate pairs from list 8_p: (s_1, s_3), (s_2, s_3), (s_3, s_4), …, (s_3, s_{10}). For signature $Sign_s$ shown in Figure 1(b), s_3 with $Sign_s(s_3) = \{3_s, 23_s\}$ forms candidate pairs with 0 sets from list 3_s and 5 sets from list 23_s: (s_1, s_3), (s_2, s_3), (s_3, s_4), (s_3, s_5), (s_3, s_{11}).

Objective. Our goal is to develop a new signature composition technique that enables *stable* set similarity joins, i.e., joins that are time and space efficient across a wide range of dataset characteristics.

4 SIGNATURE COMPOSITION

Typically, set similarity join algorithms do not use a single "atomic" signature scheme but combine multiple signature schemes to build more selective signatures. In this section, we abstract from different set similarity join techniques and formalize three signature compositions: (1) *direct composition*, (2) *partition-based composition*, and (3) *dependent composition*. We omit proofs due to space constraints.

Direct Composition. Given two signature schemes $Sign_p$ and $Sign_s$, their *direct composition* $Sign_p \times Sign_s$ is the Cartesian product $Sign_p \times Sign_s(r) = Sign_p(r) \times Sign_s(r)$. Intuitively, a probing set r must match both signatures $Sign_p$ and $Sign_s$ to be a candidate. This generates fewer candidates as dissimilar sets are less likely to match on both signatures, but requires additional computations. For example, AllPairs [2] directly composes prefix and length filter.

Lemma 4.1. *The* direct composition $Sign_p \times Sign_s$ *of signature schemes,* $Sign_p$ *and* $Sign_s$*, is a signature scheme, i.e.,* $r \triangle s \leq \epsilon \Rightarrow Sign_p \times Sign_s(r) \cap Sign_p \times Sign_s(s) \neq \emptyset$*. The candidates of* $Sign_p \times Sign_s$ *are a subset of the candidates generated by* $Sign_p$ *resp.* $Sign_s$*.*

Example 4.2. Consider the direct composition $Sign_p \times Sign_s$ depicted in Figure 1 and probing set s_3: $Sign_p \times Sign_s(s_3) = \{(1_p, 3_s), (1_p, 23_s), (8_p, 3_s), (8_p, 23_s)\}$. Signature $(8_p, 23_s)$ generates four candidates: s_1, s_2, s_4, s_5. No other signature generates a candidate.

Partition-based Composition. This composition uses a partition signature *Part* that partitions the token universe into n disjoint subsets $P = \{P_1, \dots, P_n\}$: $Part(r) = \{r \cap P_k \mid k \in \{1, \dots, n\}\}$. Given a signature scheme $Sign_s$ and a partition signature *Part*, the partition-based composition is the direct image $Sign_s \circ Part(r) = \{(k, t_s) \mid k \in \{1, \dots, n\}, t_s \in Sign_s(r \cap P_k)\}$. The hidden parameters ϵ_k for $Sign_s$ can differ for each partition P_k as long as $\epsilon \leq \sum_{k=1}^{n} \epsilon_k + n - 1$. This composition typically decreases the signature generation cost for schemes $Sign_s$ that are expensive to compute for large sets or high ϵ, but does *not* generate fewer candidates in general. PartAlloc [7] is an example for a partition-based composition.

Lemma 4.3. *The* partition-based composition $Sign_s \circ Part$ *of a signature scheme* $Sign_s$ *and a partition scheme Part is a signature scheme, i.e.,* $r \triangle s \leq \epsilon \Rightarrow Sign_s \circ Part(r) \cap Sign_s \circ Part(s) \neq \emptyset$*.*

Dependent Composition. A *primary* signature scheme $Sign_p$ forms a *dependent composition* with a *secondary* signature scheme $Sign_s$ if $Sign_s(r)$ depends on both r and $Sign_p(r)$. This composition is often based on a direct composition $Sign_p \times Sign_s$ but leverages signature-specific opportunities to reduce the number of required signatures w.r.t. the direct composition. The correctness is not given by construction and requires dedicated proofs. An example is the position-enhanced length filter (PEL) [13], a composition of prefix and positional filter, that leverages information of the primary prefix signature to reduce the number of required length signatures.

5 HYBRID SIGNATURE COMPOSITION

Our goal is a stable signature scheme that performs well across a wide range of different dataset characteristics. To this end, we introduce a novel composition technique, *hybrid signature composition*, that generalizes the direct composition and leverages the highly selective portion of each of the composed signatures. Compared to the dependent composition (cf. Section 4), the hybrid composition does not rely on specifics of the composed signatures and generalizes to arbitrary signature schemes.

5.1 A Hybrid Signature Scheme

We propose the following *hybrid signature scheme* for a primary signature scheme $Sign_p$ and a secondary signature scheme $Sign_s$:

$$Sign_p \times_h Sign_s(r) = \bigcup_{t_p \in Sign_p(r)} \begin{cases} \{(t_p)\} & A[t_p] = 0 \\ \{(k, t_s) \mid t_s \in Sign_s(r)\} & A[t_p] = k \end{cases}$$

The hybrid signature scheme is configured by an *allocation vector* A with one entry per primary signature $t_p \in Sign_p(r)$: If $A[t_p]$ is zero, only the primary scheme is used; an integer $k > 0$ indicates the partition of the secondary scheme that is used for the sets with signature t_p. In our two-level hybrid index introduced below, a separate secondary index is created for each partition k.

Lemma 5.1. *The* hybrid composition $Sign_p \times_h Sign_s$ *of two signature schemes,* $Sign_p$ *and* $Sign_s$*, is a signature scheme:*

$$r \triangle s \leq \epsilon \Rightarrow Sign_p \times_h Sign_s(r) \cap Sign_p \times_h Sign_s(s) \neq \emptyset$$

Proof. Assume $r \triangle s \leq \epsilon$. As $Sign_p$ and $Sign_s$ are signature schemes, there exist t_p and t_s with $t_p \in Sign_p(r) \cap Sign_p(s)$ and $t_s \in Sign_s(r) \cap Sign_s(s)$. If $A[t_p] = 0$, r and s share at least the signature (t_p) in $Sign_p \times_h Sign_s$. If $A[t_p] = k \neq 0$, r and s share at least the signature (k, t_s) in $Sign_p \times_h Sign_s$. □

To illustrate the impact of the allocation vector A, we study three special cases of $Sign_p \times_h Sign_s$: (1) If $A[t_p] = 0$ for all t_p, then $Sign_p \times_h Sign_s(r) = \cup_{t_p \in Sign_p(r)} \{(t_p)\} \cong Sign_p(r)$, and we recover $Sign_p$. (2) If $A[t_p] = 1$ for all t_p, then $Sign_p \times_h Sign_s(r) = \cup_{t_s \in Sign_s(r)} \{(1, t_s)\} \cong Sign_s(r)$, and we recover $Sign_s$. (3) If all values in A are non-zero and $A[t_p] \neq A[t'_p]$ for any two primary signatures $t_p \neq t'_p$, then there is a one-to-one correspondence of values $k = A[t_p]$ and signatures t_p. Hence, we can replace (k, t_s) with (t_p, t_s) and derive the direct composition: $Sign_p \times_h Sign_s(r) = \cup_{t_p \in Sign_p(r), t_s \in Sign_s(r)} \{(t_p, t_s)\} = Sign_p \times Sign_s(r)$.

By choosing A carefully, we can *interpolate* between the three signature schemes $Sign_p$, $Sign_s$*, and* $Sign_p \times Sign_s$. This allows us to use highly selective portions of $Sign_p$ and $Sign_s$, and to fall back to $Sign_p \times Sign_s$ only if neither signature is sufficiently selective.

5.2 A Two-Level Index for Hybrid Compositions

This section introduces our two-level *hybrid index* to efficiently leverage the hybrid signature composition $Sign_p \times_h Sign_s$. To foster the intuition, we interpret the hybrid composition as a two-level index rather than an index with two different types of signatures, (t_p) and (k, t_s). The first level is built using $Sign_p$. An entry for a primary signature t_p either references an inverted list if $A[t_p] = 0$ or the k-th secondary index if $A[t_p] = k$. With this interpretation, we have a choice for each primary signature t_p: (a) Retain the inverted list L^I_p as-is in the primary index ($A[t_p] = 0$), or (b) transfer all sets in the inverted list L^I_p to a secondary index; we can freely choose to which secondary index the sets are transferred. Case (b) is also referred to as *reindexing*.

Figure 2 depicts the steps to build a two-level index for input collection R in Figure 1 (allocation vector A highlighted in gray). Step $\textcircled{\small I}$ indexes R using $Sign_p$ to create the primary index; all entries in A are zero. In step $\textcircled{\small II}$, a part of the primary index remains as-is ($A[t_p] = 0$), whereas all sets in the lists of other primary signatures are transferred to disjoint secondary indexes[2] using $Sign_s$. In our example, signatures 1_p to 4_p remain in the primary index, and a secondary index is built for each primary signature 5_p to 8_p. The last four (non-zero) integers in A refer to individual secondary indexes (numbered 1-4). Step $\textcircled{\small III}$ finally merges multiple secondary indexes to reduce the cost of building and probing additional indexes. In Figure 2, we merge the secondary indexes of 5_p and 8_p (blue) as well as 6_p and 7_p (red). A is updated accordingly: 1 and 2 refer to the blue and red secondary index, respectively. Steps $\textcircled{\small II}$ and $\textcircled{\small III}$ aim to reduce the overall set similarity join costs (cf. Section 7).

[2]This is an intermediate, conceptual step without any physical index building.

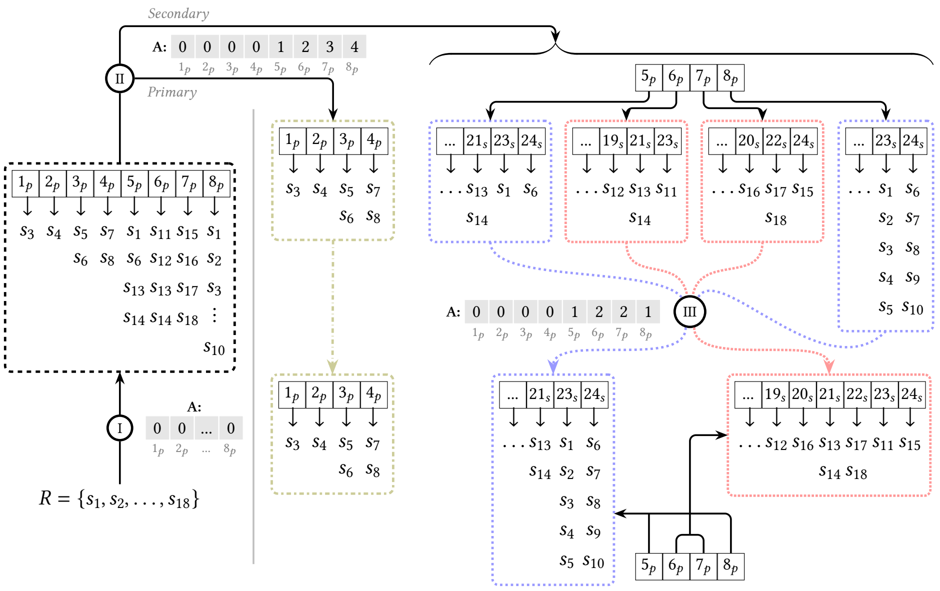


Figure 2: Overview of our two-level hybrid index for hybrid signature compositions.

5.3 TwoL: A Hybrid Composition Framework

The TwoL framework depicted in Algorithm 2 implements hybrid compositions as a two-level index. The computation of the allocation vector A is a black box in TwoL. Since our framework is still signature-based, it resembles Algorithm 1. During indexing, a full primary index is built using $Sign_p$ (lines 1-3). Then, the allocation vector A is computed (line 4) and signatures with non-zero allocation are transferred to a secondary index (lines 5-7). For probing a primary signature t_p, we either (a) ignore t_p if the signature is not in the domain of A (implicity in lines 11-12), (b) look up the signature (t_p) (line 11), or (c) store its secondary index $A[t_p]$ for later lookup (line 12). Multiple primary signatures may refer to the same secondary index, thus we postpone and batch-lookup all secondary indexes (line 14). Finally, candidates are verified (line 16).

Algorithm 2: The TwoL Framework

```
   Input: Collection $R$, distance $\epsilon$
   Result: All similar pairs in $R \times R$
1  $L^I \leftarrow \emptyset, M \leftarrow \emptyset, C \leftarrow \emptyset$          // inv. index, result, candidates
2  forall $r \in R$ do                                  // first-level indexing; step (I)
3      forall $t_p \in Sign_p^I(r)$ do $L^I_{(t_p)} \leftarrow L^I_{(t_p)} \cup \{r\}$
4  compute allocation vector $A$                        // steps (II) and (III)
5  forall $t_p$ with $A[t_p] = k \neq 0$ do                 // sec.-lvl indexing
6      forall $r \in L^I_{(t_p)}; t_s \in Sign_s^I(r)$ do $L^I_{(k,t_s)} \leftarrow L^I_{(k,t_s)} \cup \{r\}$
7      $L^I_{(t_p)} \leftarrow \emptyset$
8  forall $r \in R$ do
9      $K \leftarrow \emptyset$
10     forall $t_p \in Sign_p^P(r)$ do                  // first-level probing
11         if $A[t_p] = 0$ then $C \leftarrow C \cup \{(r,s) \mid s \in L^I_{(t_p)}\}$
12         else $K \leftarrow K \cup A[t_p]$;
13     forall $(k, t_s) \in K \times Sign_s^P(r)$ do         // sec.-lvl. prob.
14         $C \leftarrow C \cup \{(r,s) \mid s \in L^I_{(k,t_s)}\}$
15 forall candidate pairs $(r,s) \in C$ do               // verification
16     $M \leftarrow M \cup (r,s)$ if $r \triangle s \leq \epsilon$
17 return $M$
```

6 MINIMUM INDEX COST ALLOCATION

This section defines a cost model for the allocation vector A that considers the four main cost factors of signature-based set similarity joins: (1) signature generation, (2) indexing, (3) probing, and (4) verification. We show that finding an optimal allocation vector A is computationally infeasible and propose effective heuristics.

Motivation. Depending on the signature scheme and dataset, some primary signatures may appear more frequently than others during indexing and probing. Frequent signatures are problematic because the number of candidates is proportional to the size of the Cartesian product $L^I_t \times L^P_t$. In Figure 1(a), most candidates stem from list $L^I_{8_p}$ ($= L^P_{8_p}$), whereas many lists generate only a few candidates. Figure 3, which shows the number of candidates over prefix-based signatures, illustrates this behavior for two real-world datasets. At the cost of additional computational overhead, we can reduce the number of candidates by combining or substituting unselective primary signatures with a more selective secondary signature scheme $Sign_s$. Combination and substitution directly correspond to an interpolation towards $Sign_p \times Sign_s$ and $Sign_s$, respectively (cf. Section 5.1). Figure 1(b) illustrates this for a signature scheme $Sign_s$ that generates many but rather selective signatures (i.e., short lists).

6.1 Problem Definition

Our goal is to transfer an inverted list from the primary index to a secondary index such that the overall join costs are minimized. We distinguish between *primary* and *secondary index cost.*

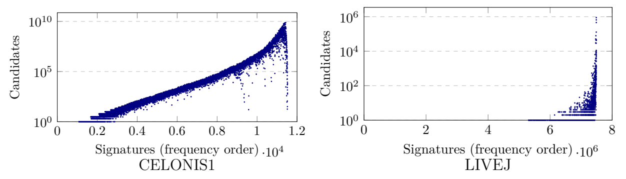


Figure 3: Candidate distribution over prefix-based signatures.

Primary Index Costs. The primary index costs include the costs (1) to verify all its candidates, (2) to index the sets, (3) to generate the probing signatures, and (4) to probe the index (cf. Table 1):

$$C_p = C_p^{verify} + C_p^{index} + C_p^{siggen} + C_p^{probe}$$

The verification costs are derived from the costs to verify a single set[3], c_p^{verify}, and the number of unique, unordered candidates: $c_p^{verify} \left|\bigcup_{A[t]=0}\{(r,s) \in L_t^I \times L_t^P\}\right|$. The probing costs depend on the lengths of all scanned lists: $l_p^{scan} = \sum_{A[t]=0} \left|\{(r,s) \in L_t^I \times L_t^P\}\right|$. Qin et al. [22] show that optimizing the number of candidates using L_t^I and L_t^P is NP-hard. We estimate the ratio of deduplicated candidates α_p and use the inverted list lengths to estimate both the number of candidates and the probing costs in a similar manner:

$$C_p^{verify} = \alpha_p c_p^{verify} l_p^{scan} \qquad C_p^{probe} = c_p^{probe} l_p^{scan}$$

Generating the indexing and probing signatures $Sign_p^I$ and $Sign_p^P$ is not part of the cost model. We generate these signatures to derive L_t^I and L_t^P, hence transferring a list does not reduce these costs.

Secondary Index Costs. The costs of secondary indexes resemble the primary index costs, but differ w.r.t. the cost constants c_s^x (cf. Table 1). Moreover, R is decomposed such that a single set may be indexed (and probed) in multiple secondary indexes. To minimize the number of replicated sets, we build at most K secondary indexes, i.e., $A[t] \leq K$. The secondary index costs consider the costs (1) to verify the candidates, (2) to transfer the inverted lists and to generate their signatures, (3) to generate the probing signatures, and (4) to scan the inverted lists in the secondary index:

$$C_s = C_s^{verify} + C_s^{index} + C_s^{siggen} + C_s^{probe}$$

We estimate the deduplicated verification costs using α_s and the sum over all K (possibly empty) secondary indexes. $R_k^I = \bigcup_{A[t]=k} L_t^I$ and $R_k^P = \bigcup_{A[t]=k} L_t^P$ denote the sets of all indexed and probed sets for the k-th secondary index, respectively. A pair (r, s) is a candidate if $Sign_s^I(r) \cap Sign_s^P(r) \neq \emptyset$. With the accumulated lengths $l_s^{scan} = \sum_{k=1}^{K} \left|\{(r,s) \in R_k^I \times R_k^P \mid Sign_s^I(r) \cap Sign_s^P(s) \neq \emptyset\}\right|$ of all scanned secondary index lists, we estimate the verification costs and the probing costs over all secondary indexes:

$$C_s^{verify} = \alpha_s c_s^{verify} l_s^{scan} \qquad C_s^{probe} = c_s^{probe} l_s^{scan}$$

Similarly, the indexing and signature generation costs are:

$$C_s^{index} = c_s^{index} \sum_{k=1}^{K} \left|R_k^I\right| \qquad C_s^{siggen} = c_s^{siggen} \sum_{k=1}^{K} \left|R_k^P\right|$$

[3]For some techniques, e.g., the prefix filter, verification is cheaper as the prefix overlap is computed during probing. Consequently, more easier-to-verify candidates may be faster than verifying fewer hard-to-verify candidates.

Table 1: Costs for primary (C_p) and secondary index (C_s).

Notation	Description
C_p^{verify}, C_s^{verify}	Verification costs (to verify all candidates).
C_p^{index}, C_s^{index}	Indexing costs.
C_p^{siggen}, C_s^{siggen}	Signature generation costs for probing.
C_p^{probe}, C_s^{probe}	Probing costs (to probe the index).
c_p^x, c_s^x	Single set cost, $x \in \{verify, index, siggen, probe\}$

Summarizing, we define the Minimum Index Cost Allocation (MICA) problem as follows:

Definition 6.1. Minimum Index Cost Allocation (MICA) Problem. Given a primary index with inverted lists L_t^I and L_t^P for the indexing and probing signatures, respectively, a secondary signature scheme $Sign_s^I$, $Sign_s^P$, non-negative costs c_p^{verify}, c_p^{probe}, c_s^{verify}, c_s^{siggen}, c_s^{index}, and c_s^{probe}, and a maximum number of secondary indexes K. Find an a-dimensional vector A with (1) $A[t] = 0$ if the signature t remains in the primary index and (2) $0 < A[t] = i \leq K$ if the elements in the inverted list of t are transferred to the i-th secondary index (i.e., reindexed), which minimizes:

$$C_p^{verify} + C_p^{probe} + C_s^{verify} + C_s^{siggen} + C_s^{index} + C_s^{probe}$$

6.2 Optimal Index Allocation

Finding an optimal solution to the Minimum Index Cost Allocation problem is NP-hard as shown in the next lemma.

Lemma 6.2. *The MICA problem is NP-hard.*

Proof Sketch. We show that the decision variant of the signature deletion problem is NP-hard, and consider a special case.

An instance $\left(\{L_t^I\}, Sign_s^I, Sign_s^P, c_p^{verify}, c_s^{verify}, c_s^{index}, C\right)$ is a true instance if there exists an allocation A that uses at most 1 secondary index while having costs of less than C. All other costs are 0, $L_t^I = L_t^P$, $\alpha_p = \alpha_s = 1$, and $Sign_s^I(r) = Sign_s^P(r) = constant$. We reduce from *CLIQUE*, where the goal is to decide if a graph $G = (V, E)$ has a clique of size m, i.e., a complete subgraph of G with size m. We set $\{L_t^I\} = E$, $C = \sum_t \frac{|L_t^I|^2 - |L_t^I|}{2} - \frac{1}{m}$, $c_p^{verify} = 1$, $c_s^{verify} = \frac{m-2}{m-1}$, and $c_s^{index} = \frac{m-2}{2(m-1)}$. As $|L_t^I|^2 - |L_t^I| = 2$ for all t and a given A, it suffices to check if $\sum_{A[t]=1} 1 - \frac{1}{m} \geq \frac{m-2}{m-1} \frac{|\bigcup_{A[t]=1} L_t^I|^2}{2}$. Consider the subgraph that contains all edges with $A[t] = 1$ and their nodes. The left side, $\sum_{A[t]=1} 1$, is (slightly less than) the number of edges in the subgraph. Since $|\bigcup_{A[t]=1} L_t^I|$ is the number of nodes in the subgraph, the right side computes the Turán number [25], i.e., the max. number of edges any graph with $n = |\bigcup_{A[t]=1} L_t^I|$ nodes and without a clique of size m can contain. If G contains a clique of size m, then selecting these edges for the secondary index yields $\frac{m^2-m}{2} - \frac{1}{m} \geq \frac{m-2}{m-1} \frac{m^2}{2}$, which is true for any $m > 1$. Conversely, if an allocation A with $\sum_{A[t]=1} 1 - \frac{1}{m} \geq \frac{m-2}{m-1} \frac{|\bigcup_{A[t]=1} L_t^I|^2}{2}$ exists, then a clique of size m that forms A must exist in the subgraph as this subgraph has more edges than allowed by any graph of equal size without a clique of size m. *CLIQUE* can be reduced to a special case of MICA, hence MICA is also NP-hard in the general case. □

6.3 Heuristic Index Allocation

Because an optimal solution is infeasible, we propose heuristic approaches to allocate our hybrid index.

Simple Decomposition. First, we want to identify *rewarding* signatures (i.e., lists) based on a conservative estimation.

Definition 6.3. The *standalone costs* $SC_p(L_t^I, L_t^P)$, $SC_s(L_t^I, L_t^P)$ of lists L_t^I, L_t^P of signature t are the costs of an index that only contains L_t^I and only probes L_t^P with $A[t] = 0$ and $A[t] = 1$, respectively:

$$SC_p(L_t^I, L_t^P) = (\alpha_p c_p^{verify} + c_p^{probe}) \left|\{(r,s) \in L_t^I \times L_t^P\}\right|$$
$$SC_s(L_t^I, L_t^P) = (\alpha_s c_s^{verify} + c_s^{probe}) \cdot \left|\{(r,s) \in L_t^I \times L_t^P \mid \ldots\}\right| + c_s^{index}|L_t^I| + c_s^{siggen}|L_t^P|$$

Lemma 6.4. *If $K \geq a$ and all lists L_t^I, L_t^P are pairwise disjoint, i.e., for all $t \neq u$, $L_t^I \cap L_u^I = \emptyset$, $L_t^P \cap L_u^P = \emptyset$, then we can solve the MICA problem by minimizing the standalone costs for each pair (L_t^I, L_t^P).*

By Lemma 6.4, we now only consider *rewarding* signatures t with $SC_p(L_t^I, L_t^P) > SC_s(L_t^I, L_t^P)$. The result is a *simple decomposition* heuristic that partitions all signatures into disjoint sets of rewarding and non-rewarding signatures to implement step Ⅱ in Figure 2.

To reduce the cost model overhead, we do no consider cases where non-rewarding signatures form a secondary index with a negative cost change, e.g., large list overlaps and high c_s^{index}, c_s^{siggen}.

Example 6.5. Consider the sets in Figure 1, and assume symmetric primary and secondary signature schemes, $\alpha_p = \alpha_s = c_p^{verify} = c_s^{verify} = c_s^{index} = 1$, and all other costs are 0. The standalone costs of signature 8_p are $SC_p = 45 + 0 = 45$ ($= \frac{10 \cdot 9}{2}$ candidates). Conversely, $SC_s = 10 + 20 = 30$ due to the transfer (10) and the verification costs of lists 23_s, 24_s ($10 + 10 = 20$). Since $SC_p > SC_s$, $(L_{8_p}^I, L_{8_p}^P)$ is rewarding and we transfer it to the second index level. In contrast, the standalone costs of signature 4_p are $SC_p = 1$ (number of candidates is not reduced) and $SC_s = 3$ (transfer costs), thus 4_p is non-rewarding. The lists of 5_p, 6_p, 7_p, and 8_p are all rewarding, and hence transferred to level 2. Figure 2 shows the partial primary index (green) and the simple decomposition after step Ⅱ.

Heuristical Index Merging. A simple decomposition may result in many secondary indexes and high transfer and signature generation costs (if inverted lists are not disjoint). To reduce these costs, we can merge multiple secondary indexes. However, this may result in higher candidate and probing costs due to more signature collisions.

Consider the standalone costs $SC_s(R_i^I, R_i^P)$ and $SC_s(R_j^I, R_j^P)$ of the i-th and j-th secondary index, respectively. The overall costs are reduced if $SC_s(R_i^I, R_i^P) + SC_s(R_j^I, R_j^P) > SC_s(R_i^I \cup R_j^I, R_j^P \cup R_j^P)$. The inclusion-exclusion principle shows that the cost change Δ is

$$\Delta = (\alpha_s c_s^{verify} + c_s^{probe}) \Big(\left|\{(r,s) \in (R_i^I \setminus R_j^I) \times (R_j^P \setminus R_i^P) \mid \ldots\}\right| + \left|\{(r,s) \in (R_j^I \setminus R_i^I) \times (R_i^P \setminus R_j^P) \mid \ldots\}\right| - \left|\{(r,s) \in (R_i^I \cap R_j^I) \times (R_i^P \cap R_j^P) \mid \ldots\}\right| \Big) - c_s^{index}\left|R_i^I \cap R_j^I\right| - c_s^{siggen}\left|R_i^P \cap R_j^P\right|$$

If $\Delta \leq 0$, the Δ-*check* succeeds and the indexes are merged. With the Δ-check, we construct a heuristic index merging technique, *MultiReassessment*, that enhances a given simple decomposition L (cf. Algorithm 3). We use two strategies: (1) We only merge indexes if their Δ-check succeeds, i.e., the total cost never increases. (2) All rewarding signatures of L should be part of a secondary index, even if only a single list forms the index. First, we order the rewarding signatures in L by descending standalone cost difference $SC_p - SC_s$. Δ-checks only succeed if two secondary indexes have a significant overlap. This order emphasizes the list lengths (which determine SC_p) and places them at the beginning to increase the probability of overlaps. We unconditionally[4] build an index on the first list. Then, we merge all other lists that pass the Δ-check into this index while we track skipped lists that failed the Δ-check. Finally, we build a new index on the skipped lists. To reduce the costs of a Δ-check, secondary signature collisions are estimated (cf. Section 7.7) on samples. For sampling, we select 1% of the smaller list and build random pairs with the larger list. Moreover, the number of Δ-checks is bounded by the number of rewarding lists $|L|$ and the number of indexes k built at the end of the loop. In the worst case, we need $O(k \cdot |L|)$ Δ-checks. In practice, the number of built indexes is low (≤12 in our experiments). Additionally, we can limit the max. number of indexes and use the simple decomposition for the remaining lists.

Example 6.6. Consider step Ⅱ and Ⅲ in Figure 2. MultiReassessment starts by building an empty index $k = 1$ (blue) and merging $L_{8_p}^I$ into the index. For $L_{7_p}^I$, the Δ-check $\Delta = 5 - 0 \nleq 0$ fails and we track $L_{7_p}^I$. Also the Δ-check for $L_{6_p}^I$ fails. For $L_{5_p}^I$, the Δ-check $\Delta = -2 \leq 0$ succeeds and we build the indexes. We continue by building a new empty index $k = 2$ (red) and merging in $L_{7_p}^I$. Since $L_{7_p}^I$ and $L_{6_p}^I$ are disjoint and do not suffer from collision in the secondary signatures, we merge them in because $\Delta = 0$. The overall costs are $0 + 0 + 1 + 1 + 33 + 10 = 45$.

In addition to MultiReassessment, we also evaluate a baseline called *SingleSimple* that merges all lists of the simple decomposition into a *single* secondary index. We expect good performance for datasets on which the secondary index severely outperforms the primary index. However, short inverted lists still remain in the primary index since a transfer does not pay off.

Example 6.7. SingleSimple indexes all rewarding inverted lists into one secondary index with overall costs of $0 + 0 + 1 + 1 + 50 = 52$.

6.4 Deletion under Memory Constraints

Different indexes typically differ in the number of indexing signatures generated for each set. The number of unique signatures is important for the memory usage of the index implementation as every signature needs a lookup-table entry, and almost empty lists incur a high overhead. Depending on the signature scheme, this number (a) is a user-defined constant [22], (b) grows in the size of the set's prefix [2], (c) grows in the set size [7], or (d) grows exponentially in the Hamming distance [18]. For datasets on which memory-efficient indexes perform poorly and fast indexes do not

[4] R_k^I, R_k^P are empty when the k-th index is built, hence $\Delta = 0$ and the Δ-check succeeds.

Algorithm 3: MultiReassessment

```
Input: All rewarding lists $L = \{(L_t^I, L_t^P) \mid t \text{ is rewarding}\}$
Result: Allocation vector $A$
$A \leftarrow a$-dimensional vector, initialized to 0; $Q \leftarrow L$; $k \leftarrow 0$
while $Q \neq \emptyset$ do
    $Q' \leftarrow \emptyset$; $k \leftarrow k + 1$; $R_k^I \leftarrow \emptyset$; $R_k^P \leftarrow \emptyset$
    forall $(L_t^I, L_t^P) \in Q$ in descending order by $SC_p - SC_s$ do
        if $\Delta$-check for $R_k^I, R_k^P$ and $L_t^I, L_t^P$ succeeds then
            $A[t] \leftarrow k$; $R_k^I \leftarrow R_k^I \cup L_t^I$; $R_k^P \leftarrow R_k^P \cup L_t^P$
        else $Q' \leftarrow Q' \cup (L_t^I, L_t^P)$
    $Q \leftarrow Q'$
return $A$
```

fit into main memory, we build a cheap primary index and transfer only the inverted lists with the highest benefit to the expensive secondary index. This allows for a time-space trade-off.

7 CONCRETE SIGNATURE SCHEMES

In this section, we represent well-known filters as signature schemes. We focus on the schemes that we use in our concrete TwoL implementation. Additionally, we propose an extended position-enhanced length filter, *EPEL*, and we adapt the CoveringLSH [9] signature scheme to be efficient in practice by filtering singleton lists.

7.1 Length Filter

The length filter by Arasu et al. [1] compares the sizes of two sets r and s to prune the pair (r, s) if their sizes differ too much. For the Hamming distance, the signature scheme *Len* is defined as:

$$Len^I(r) = \{|r|\} \quad Len^P(r) = \{i \mid l_{min} \leq i \leq l_{max}\}$$

where $l_{min} = |r| - \epsilon$ and $l_{max} = |r| + \epsilon$. For self joins, the upper bound l_{max} is replaced with $|r|$ to skip symmetric results.

7.2 Prefix Filter

The prefix filter by Chaudhuri et al. [5] considers the first π tokens as the signature of a set. The tokens of all sets are sorted w.r.t. a global token order, and π typically depends on the set size and the distance function. Xiao et al. [30] reduce the size of the indexing prefix π^I for self joins w.r.t. the probing prefix π^P (=π). For the Hamming distance, the signature scheme *Pre* is defined as:

$$Pre^I(r) = \{r_i \mid r = (r_1, \ldots, r_j) \text{ and } 1 \leq i \leq \epsilon/2 + 1\}$$

$$Pre^P(r) = \{r_i \mid r = (r_1, \ldots, r_j) \text{ and } 1 \leq i \leq \epsilon + 1\}$$

AllPairs [2] is a state-of-the-art set similarity join algorithm [14] that uses the direct composition of length filter and prefix filter $Len \times Pre$ in a main-memory setting.

7.3 CoveringLSH

Locality-sensitive hashing (LSH) for Hamming space [9] uses a set of hash functions H of the type $h(r) = r \wedge a$ where a is a bitmask and $\wedge$ is the bitwise AND. Choosing the bitmasks randomly results in false negatives if the bitmasks sample only tokens that differ between similar sets. CoveringLSH [18] constructs a correlated set of hash functions H that ensures that there is at least one hash function that maps two similar sets to the same hash value, i.e., there are no false negatives. Its signature scheme *cLSH* is the set of all hash values $cLSH(r) = \{h(r) \mid h \in H\}$. The correlated construction is not significantly slower than its uncorrelated counterpart; Fast CoveringLSH [20] (denoted FCLSH) uses the Fast Hadamard Transform to further reduce the signature computation time.

7.4 Extended PEL

The positional filter [30] states that r and s can only be similar if the overlap in their first p_r and p_s tokens (excl. the current match) is o, and $\frac{|r|+|s|-\epsilon}{2} \leq o + \min\{|r| - p_r, |s| - p_s\}$. If applied only to the first match of r and s, $o = 0$ can be assumed. The position-enhanced length filter (PEL) [13] advances the length filter and is a dependent composition on top of *Pre* that uses the token position in the probing set r to find a tighter upper bound $l_{max} = \min\{|r|, |r| - 2p_r + \epsilon\}$. For self joins, however, the upper bound often remains unchanged: $l_{max} = |r|$. Hence, some algorithms [28] reverse the probing order and only search for similar sets that are larger than the probing set r, i.e., $l_{min} = |r|$ and $l_{max} = |r| - 2p_r + \epsilon$. However, reversing the order requires us to use Pre^P for indexing and Pre^I for probing. Since Pre^P contains more signatures than Pre^I, this increases (a) the list lengths L_p^I and the associated transfer costs, and (b) the memory usage due to larger secondary indexes.

To this end, we propose an extended, two-sided variation of PEL, named *EPEL*, that retains the processing order but has higher pruning effectiveness than PEL. We keep the upper bound of PEL $l_{max} = \min\{|r|, |r| - 2p_r + \epsilon\}$ and improve the lower bound. If the position p_s is known, a tighter lower bound $l_{min} = |r| + 2p_s - \epsilon$ can be derived exactly like the upper bound. For each list L_t^I, we group all indexed sets s by p_s and apply l_{min} to each group. As datasets are processed in increasing size of $|r|$, l_{min} grows monotonically in each group. Therefore, an index entry that is skipped once due to l_{min} will be skipped by all further applications of the length filter. During probing, we maintain a monotonically increasing offset to the first set of every group that passed the length filter. After failing l_{max} for the first time, the rest of the group can be skipped.

7.5 Self Join Optimizations

In the signature-based framework (cf. Algorithm 1), we index a set r by computing all signatures of r and inserting r into the respective inverted lists. In our experiments, we observe that between 90% and 99% of all lists are of length 1 for FCLSH (depending on the dataset). For a self join with symmetric signature scheme, singleton lists are called *trivial:* Only the set that is the sole element of the list will probe this list, thus we can disregard these reflexive pairs. In practice, a trivial list incurs high overhead because dynamic arrays (e.g., `std::vector`) typically consist of 3 pointers, i.e., a trivial list needs at least 4 memory words (e.g., 32 bytes on a 64-bit machine).

Index Filtering. Our index structure enhances the state of the art by only indexing signatures that appear more than once. This is particularly important for memory-heavy approaches like FCLSH, but also results in faster runtimes (no probing of trivial lists). For self joins, we process the sets in inverse processing order (i.e., largest to smallest). We build a Bloom filter over all signatures: If a signature is new, it is added to the Bloom filter, otherwise we insert it into the index. Since we only skip indexing for the first occurrence of a signature in inverse processing order, we only miss reflexive result pairs (r, r) during probing. Although index filtering does not reduce FCLSH's worst-case space complexity of $O(n(2^{\epsilon+1} - 1))$, the memory usage decreases sharply in practice. Index filtering is applicable to all self joins with symmetric signature schemes as indexing and probing access the same lists for a given set.

7.6 Choice of Primary and Secondary Index

The TwoL framework (cf. Algorithm 2) must be instantiated with concrete signatures and indexes. Our goal is to ensure that at least one of the two indexes performs well on a dataset, i.e., the signature schemes complement each other. Then, our hybrid index will handle no dataset significantly worse than the better of the two indexes.

We select the dependent composition of *EPEL* with *Pre* as our primary signature scheme. The prefix filter is space-efficient and performs well on many real-world datasets [14]. For some datasets with small universes, however, many candidates are generated. To this end, we choose *cLSH* (i.e., FCLSH) that works with dense bitvectors instead of sets to complement the primary signature scheme. TwoL (MR) and TwoL (SS) denote that we use MultiReassessment and SingleSimple to compute the allocation vector A, respectively.

Other Distances & Signatures. TwoL generalizes to other signature schemes and distances. Assuming the Jaccard distance, we choose a different secondary signature scheme that is complementary to AllPairs, e.g., PartAlloc , and use the equivalent Hamming distance [30]. The costs for the cost model are automatically estimated based on small samples (e.g., time to verify two sets). Our hybrid composition then adapts based on the new cost model. Notably, we need to compute the cost model only once per distance and signature, thus avoiding dataset-specific tuning.

7.7 Cost Estimation

We need an efficient way to estimate the costs for (a) the inverted lists in the primary index and (b) the union of multiple lists in the secondary index. As TwoL uses *EPEL*, $Sign_p^I$ and $Sign_p^P$ include the set's prefix tokens. $Sign_p^I$ further includes p_s and $|s|$, and $Sign_p^P$ includes all possible values of p_s and $l \in \{l_{min}, \ldots, l_{max}\}$. We estimate the length filter effectiveness for each list by sampling set pairs from each list and counting how often it would prune a pair.

To build a secondary index, the cost model requires the number of signature collisions in (step Ⅱ) and between (step Ⅲ) different inverted lists. For both PartAlloc and FCLSH, we estimate the number of collisions using the real Hamming distance of randomly sampled pairs in the indexing and probing lists. In the case of FCLSH, Corollary 1 in Pham et al. [20] states that for any set pair (r, s) we expect at most $2^{\epsilon+1-(r \triangle s)}$ hash collisions. For PartAlloc, we first observe that two sets partitioned into k parts have no common partition if at least one of the $r \triangle s$ mismatching tokens is mapped to each partition. Stirling numbers of the second kind $\left\{ {r \triangle s \atop k} \right\}$ correspond to the number of possible non-empty partitions and there are $k!$ possible permutations of each partitioning. Assuming that all tokens are uniformly assigned to partitions, the probability of a collision of a pair (r, s) is $1 - \frac{\left\{ {r \triangle s \atop k} \right\} k!}{k^{r \triangle s}}$. As PartAlloc uses a more sophisticated cost-based enumeration technique, the real probability is lower by a factor of α. In our experiments, we estimate α using sampling; depending on the dataset, $\alpha \in [0.003, 0.05]$.

8 EXPERIMENTAL EVALUATION

Algorithms. We compare our solution, TwoL, against AllPairs, GroupJoin, SizeAware, PartAlloc, FCLSH, and SkipJoin. Short descriptions of the algorithms are provided in Section 2. For AllPairs and GroupJoin, we use the efficient reimplementations of Mann et al. [14]. We reimplement PartAlloc with the greedy optimizer. The source code of SizeAware was provided by the authors and adapted to support Hamming distance joins; we use the fast verification algorithm by Mann et al. [14] for a fair comparison; prefix extensions c were chosen from $c \in \{2, \ldots, 10\}$; we only report the minimum time over all choices of c. TwoL uses *EPEL* with *Pre* as a primary signature, and *cLSH* and PartAlloc as secondary signature for the Hamming and the Jaccard distance, respectively. TwoL (MR)/(SS) denote the MultiReassessment/SingleSimple strategy of TwoL; we omit the strategy if they show similar performance.

Datasets. We evaluate all algorithms on 13 datasets (6 real-world; 7 synthetic), which cover a wide range of dataset characteristics. **Real-world** The datasets CELONIS1 and CELONIS2 [10] are from the process mining domain: A set contains the activity transitions of a process. A set in the DBLP12 dataset [24] is a publication, and a token is a 2-gram of the title. KOSARAK, LIVEJ, and ORKUT were previously used to benchmark set similarity join algorithms. For descriptions and preprocessing steps[5], we refer to Mann et al. [14]. **Synthetic** Based on CELONIS1, we create synthetic datasets to study TwoL's behavior w.r.t. dimensionality and skew. We take inspiration from Petersen et al. [19] and use the best fitting distribution of Exponential, Gamma, Generalized Extreme Value, Log-normal, and Yule-Simon as token frequency and Negative Binomial as set size. In their study, Generalized Extreme Value and Yule-Simon were the overall winners of continuous and discrete distributions, respectively. For comparatively less skewed datasets like CELONIS1, however, both distributions did not fit the dataset well due to the models' long tails. For CELONIS1, Log-normal was the best fit. We fitted token frequency and set size models using Maximum Likelihood Estimation for CELONIS1, called LNONIS1.

Then, we modified both token and set size distribution to generate synthetic datasets. LNONIS1-α/-β/-γ keep the size distribution of LNONIS1 but interpolate between the token distributions of CELONIS1 and ORKUT. NBIONIS1-α/-β/-γ keep the token distribution of LNONIS1, but interpolate between the size distributions of CELONIS1 and ORKUT. For NBIONIS1-α/-β/-γ, 25% of the sets have a max. size of 17, 13, 9, and 5, respectively. Table 3 shows the parameters for LNONIS1 and its variants. Figure 4 compares the original and the modeled distributions (token frequency and set size).

Table 2 summarizes the characteristics and Figure 4 shows the token frequencies of all datasets. KOSARAK/LIVEJ/ORKUT have large token universes and are highly skewed, whereas DBLP12/CELONIS1/CELONIS2 have a smaller token universe and are less skewed. We show the results for CELONIS1, DBLP12, KOSARAK, and ORKUT; trends for CELONIS2/LIVEJ are similar to CELONIS1/ORKUT, respectively.

[5]http://ssjoin.dbresearch.uni-salzburg.at/datasets.html

Table 2: Characteristics of datasets.

Dataset	Coll. Size	Set Size		Token Universe
		avg.	max.	
CELONIS1	$8.2 \cdot 10^6$	20.3	91	$1.2 \cdot 10^4$
CELONIS2	$6.5 \cdot 10^6$	22.5	326	$1.7 \cdot 10^4$
DBLP12	$4.6 \cdot 10^6$	75.5	562	$2.5 \cdot 10^4$
KOSARAK	$6.1 \cdot 10^5$	11.9	$2.5 \cdot 10^3$	$4.1 \cdot 10^4$
LIVEJ	$3.1 \cdot 10^6$	36.4	300	$7.5 \cdot 10^6$
ORKUT	$2.7 \cdot 10^6$	119.7	$4.0 \cdot 10^4$	$8.7 \cdot 10^6$
LNONIS1	$8.2 \cdot 10^6$	20.3	55	$4.2 \cdot 10^4$
LNONIS1-$\alpha/\beta/\gamma$	$8.2 \cdot 10^6$	20.3	55	$1.3/3.6/11 \cdot 10^5$
NBIONIS1-$\alpha/\beta/\gamma$	$8.2 \cdot 10^6$	20.3	121/175/300	$4.2 \cdot 10^4$

Table 3: Mean μ, standard deviation σ, successful trials n, and success probability p of the fitted log-normal (token frequency) and negative binomial distributions (set size).

	LN-/NBIONIS1	LNONIS1-			NBIONIS1-		
		α	β	γ	α	β	γ
μ	3.95	5.5	7	8.1	3.95		
σ	1.71	2.1	2.4	2.7	1.71		
n	50.0				4.46	2.01	0.96
p	0.711				0.18	0.09	0.045

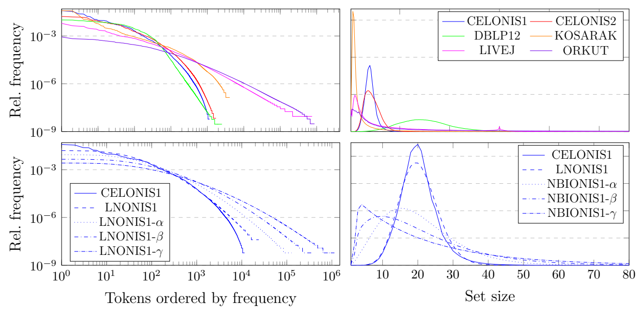


Figure 4: Characteristics of real-world and synthetic datasets.

Environment & Parameters. All experiments have been conducted on a 64-bit machine with 2 physical Intel Xeon E5-2603 v4 processors and 6 cores each, no hyperthreading. Each core has a 256kiB L2 cache, each processor has a 15MiB shared L3 cache. Our machine has 96GiB of main memory and runs Debian 10 Buster (Linux 4.19). We use clang 7.0.1 with optimization level O3. Runtime is measured with `clock_gettime` at process level (timeout: $15 \cdot 10^3$s); memory usage is the heap peak of Linux' `libmemusage` (with `LD_PRELOAD`). We ran single-core experiments with no additional load on the system. We vary the distance $\epsilon \in \{2, \ldots, 5\}$ (Hamming) and $\epsilon \in \{0.05, \ldots, 0.2\}$ (Jaccard) s.t. the join selectivity is ≤15% of the cross product.

8.1 Runtime Efficiency

We measure the total runtime to find all pairs of similar sets, excl. the time to load the dataset or perform typical preprocessing (tokenization, sorting, deduplication). The cost model overhead of TwoL is low, ranging from 1% (`ORKUT`) to at most 5% (`DBLP12`) of the total join time to find rewarding lists and perform Δ-checks (for $\epsilon = 5$ and $\epsilon = 0.2$, respectively). Figure 5 shows the results for varying ϵ. We focus on the Hamming distance, but discuss distinctive results for Jaccard. TwoL (MR) and (SS) perform similarly although TwoL (MR) may build multiple secondary indexes. For real-world datasets, TwoL (MR) never builds more than 12 secondary indexes.

Our two-level signature scheme consistently outperforms all competitors in almost all configurations for two reasons: (1) For datasets with high secondary index usage, e.g., `CELONIS1` (99% transferred) and `DBLP12` (91%), the primary index contains long lists that lead to many candidates. In those cases, all prefix-based algorithms (AllPairs, GroupJoin, SkipJoin, SizeAware) tend to perform poorly. Heavyweight signatures like fcLSH, PartAlloc, and TwoL's secondary signature scheme pay off because they are more selective (i.e., they consider multiple tokens). Interestingly, TwoL also outperforms fcLSH because TwoL disregards trivial lists (cf. Section 7.5). (2) For datasets with large universes and many small sets (`LIVEJ`, `KOSARAK`, `ORKUT`), prefix-based algorithms perform well, and the secondary index is hardly ever used (≤4% of the sets are transferred). For Jaccard, GroupJoin and AllPairs slightly outperform TwoL because the computation of the cost model does not amortize; but TwoL is still among the winning algorithms.

fcLSH and PartAlloc perform poorly on datasets with many small sets. fcLSH effectively prunes sets that are far away from the probing set. The max. Hamming distance of two sets r and s is $|r| + |s|$. For `KOSARAK`, ≈55% of the sets are of size ≤5, resulting in $\geq 2^{-4}$ expected candidates for any pair of small sets and $\epsilon = 5$, i.e., $\geq 3.5 \cdot 10^9$ over *all* small pairs (cf. Section 7.7). Conversely, TwoL avoids unfavorable transfers to its secondary signature fcLSH as it only considers lists with lower secondary standalone costs (which does not hold in this case). We observe a similar behavior for Jaccard.

SkipJoin uses a similar grouping idea as TwoL, but performs poorly even on datasets that favor the prefix filter. The reasons are twofold: (1) For Hamming, many small sets form result pairs because only a small overlap is required. This renders answer-level skipping (askip) ineffective: It is rarely used ($<10^{-4}$% over all configurations) while it still incurs the cost model overhead for each pair in the large join result. (2) Index-level skipping (iskip) is efficient if index entries are grouped based on token position (rather than set size; cf. Table 4). Overall, SkipJoin shows the best performance with disabled answer-level skipping (for small sets; size $<\epsilon$) and index-level skipping while keeping PEL with reversed processing order in place (cf. Table 5). For TwoL, the indexing cost outweighs the probing cost, justifying the need of our EPEL filter that avoids reversing the processing order (cf. Section 7.4). In the case of Jaccard and $\epsilon = 0.2$, SkipJoin times out for `DBLP12`.

For Hamming, SizeAware outperforms the other prefix-based algorithms on `CELONIS1` and `DBLP12`. Longer c-extended prefixes reduce the otherwise large number of candidates. For very small sets without a c-extended prefix, SizeAware combines the prefix filter and PEL with reversed processing order. Considering `KOSARAK`

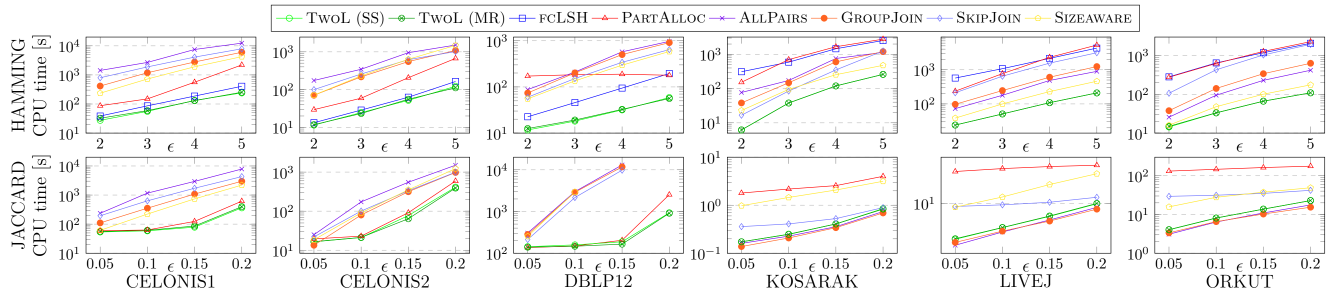


Figure 5: Runtime over ϵ for real-world datasets.

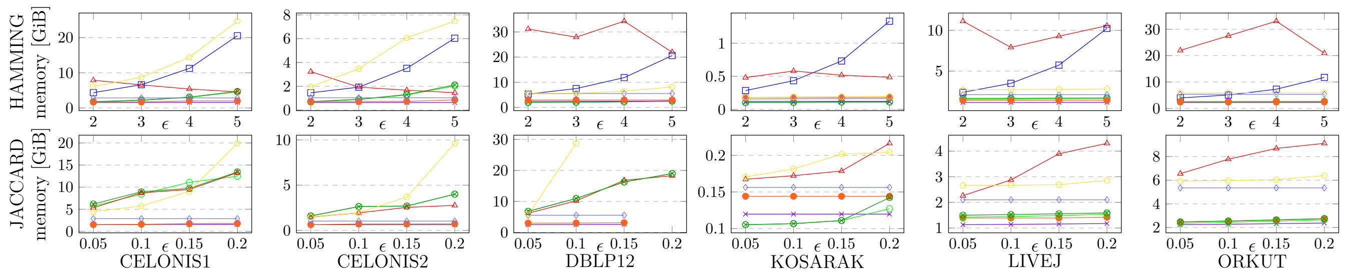


Figure 6: Main memory over ϵ for real-world datasets.

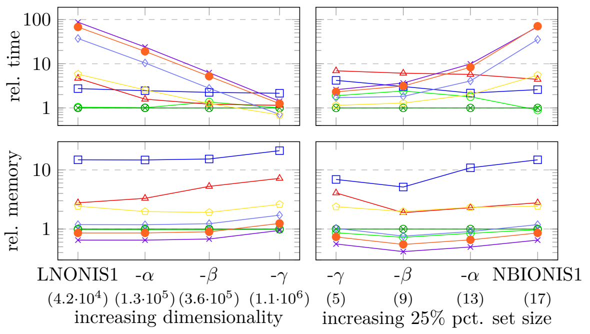


Figure 7: Relative runtime and memory w.r.t. TwoL (MR) for different synthetic datasets; $\epsilon = 5$.

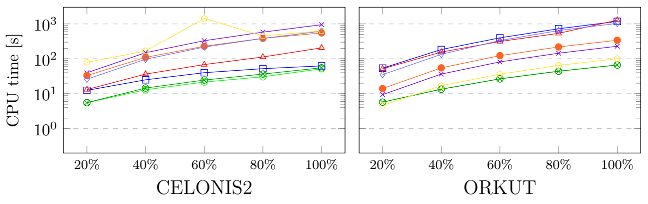


Figure 8: Runtime over varying dataset sizes; $\epsilon = 4$.

and ORKUT, SizeAware performs best with long prefix extensions when 99% and 92% of the respective total runtime was spent on very small sets. This shows that PEL with reversed processing order outperforms the length filter of AllPairs. For Jaccard, SizeAware is the slowest prefix-based algorithm on KOSARAK and runs out of memory for DBLP12 ($\epsilon > 0.1$).

Scalability. To study the scalability of our approach, we sample 20% to 80% of the records in CELONIS2 and ORKUT; Figure 8 shows the results. All algorithms scale well w.r.t. increasing dataset sizes. Only SizeAware has an outlier at 60% CELONIS2 due to consistently treating all records as "large" for this configuration. Quintupling the input size results in an increase in runtime by a factor of at most 11.6× for TwoL, whereas all other algorithms suffer from an increase by at least 20.5× on some dataset.

Concurrent Baseline. TwoL dynamically interpolates between two signature schemes and their direct composition. As a baseline, we execute *EPEL* with *Pre*, *cLSH*, and their direct composition concurrently on a single core, and terminate as soon as one of the three executions finishes. For Hamming, TwoL (executed on a single core) outperforms the concurrent baseline by a factor of 1.85× (CELONIS2) to 2.92× (DBLP12). Compared to TwoL, which uses the memory-intensive secondary signature only if necessary, the concurrent baseline requires between 4.2× (CELONIS2) and 43× (LIVEJ) more memory.

8.2 Memory Efficiency

In Figure 6, we study the memory usage of all competing algorithms for varying ϵ. For both Hamming and Jaccard, the memory usage of TwoL depends on the balance of the two signature schemes. Prefix-based algorithms are typically lightweight and require only a small number of index entries for each set. For datasets dominated by the primary signature scheme (KOSARAK and ORKUT), TwoL approximately matches the memory footprint of AllPairs, GroupJoin, and SkipJoin. SizeAware needs additional memory for storing

c-subsets. For Hamming, SizeAware requires even more memory than fcLSH on CELONIS1, CELONIS2, and ORKUT ($\epsilon < 4$). TwoL consumes significantly less memory as only a fraction of the sets are indexed with fcLSH. For Jaccard, SizeAware runs out of memory on DBLP12 ($\epsilon > 0.1$); TwoL nearly follows PartAlloc's memory footprint as dominating secondary signature scheme for CELONIS1, CELONIS2, and DBLP12. The number of PartAlloc's enumeration signatures depends on the avg. set size and results in high memory usage for DBLP12 and ORKUT.

Index Filtering. In the case of Hamming, we observe a significantly lower memory footprint for TwoL compared to fcLSH (cf. Figure 6). Using index filtering in TwoL's secondary signature scheme is beneficial for two reasons: (1) The high overhead of creating trivial lists is avoided. (2) For self joins, the last entry of each list (trivial result) is avoided. Table 6 shows the effectiveness of index filtering in terms of avoided list creations and list entries.

8.3 Synthetic Datasets

We study the impact of varying dimensionality and set size on both runtime performance and memory usage. Figure 7 shows the relative runtime performance and memory usage compared to TwoL (MR) for increasing dimensionality (LNONIS1 and its variants) and decreasing 25% percentile set size (NBIONIS1 and its variants). We notice two trends: (1) The runtime of prefix-based approaches decreases with increasing dimensionality. The performance of the prefix filter depends on the number of uncommon tokens. Uncommon tokens in the prefix result in fewer pairs of sets with overlapping prefixes and therefore fewer candidates. Figure 4 shows that the number of uncommon tokens increases for increasing dimensionality. (2) The runtime of fcLSH and PartAlloc increases with an increasing number of small sets. For the partition-based algorithm PartAlloc, partitioning small sets results in partitions with few or no tokens. These partitions are unselective and increase the number of candidates. The LSH-based algorithm fcLSH depends on the principle that set pairs with low distance often collide, and set pairs with high distance rarely have a hash value in common. Since the Hamming distance is bounded from above by the sum of the set sizes, small sets usually result in more collisions than large sets. Therefore, we observe a performance degradation when the number of small sets increases.

For LNONIS1 and its variants, TwoL gradually replaces its mostly fcLSH-based index (LNONIS1) with a mostly prefix-based index ($-\gamma$) with increasing dimensionality. This effect is also visible in the memory usage as TwoL's relative memory usage decreases with increasing dimensionality. For NBIONIS1 and its variants, TwoL (MR) replaces its mostly fcLSH-based index (NBIONIS1) with an index largely based on the direct composition of prefix filter and fcLSH. NBIONIS1$-\gamma$ with its small universe size and many small sets combines the "worst-case" scenarios of both prefix filter and fcLSH. As neither signature has high pruning power on its own, the direct composition is used to prune candidates more effectively. TwoL (SS) with its strategy of merging all secondary indexes into one index performs worse than TwoL (MR) because it cannot combine signatures. Over all settings, TwoL (MR) is the best or among the best performing algorithms and does not exhibit the shortcomings of techniques based only on a single signature.

Table 4: Average number of entries per block, $\epsilon = 5$.

	LIVEJ	KOSARAK	ORKUT
SkipJoin (block by size)	1.6	5.0	1.2
TwoL (block by position)	1.8	23.3	1.6

Table 5: Effect of iskip and askip on SkipJoin runtime, $\epsilon = 5$.

	LIVEJ	KOSARAK	ORKUT
SkipJoin	3144	1195	1967
disabled askip for small sets	848	584	410
disabled iskip and askip	586	463	250

Table 6: Index filtering effectiveness, $\epsilon = 5$.

Dataset	List entries avoided	List creations avoided
CELONIS1	79%	89%
CELONIS2	83%	94%
DBLP12	>99%	>99%

9 CONCLUSION

In this paper, we have studied the set similarity join problem. We could show that current solutions use signature schemes optimized for specific dataset characteristics. If the assumptions on the dataset characteristics do not hold, the performance of these approaches is unsatisfactory. To address this problem, we introduced the hybrid signature composition that allows for interpolation between different signature schemes and their direct composition, i.e., we use the highly selective portion of each signature scheme. We designed the TwoL framework that implements the hybrid signature composition using a hybrid index. In order to optimize the cost of the hybrid index, we developed a cost model and proposed heuristic index allocation strategies. For a concrete implementation of the TwoL framework, we chose two complementary signature schemes and enhanced them for their application in TwoL: *EPEL* with *Pre* as the primary signature scheme, and fcLSH and PartAlloc as secondary signature schemes for Hamming and Jaccard distance, respectively. Experimental results showed that TwoL outperforms its competitors on real-world datasets in most settings and is less sensitive to dataset characteristics. Experiments on synthetic datasets further demonstrated the ability of TwoL to perform well on datasets with mixed characteristics; on these datasets, schemes based on a single signature fail as none of the signature schemes is favored.

ACKNOWLEDGMENTS

This research was funded in whole, or in part, by the Austrian Science Fund (FWF) P 34962. For the purpose of open access, the authors have applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

REFERENCES

[1] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. 2006. Efficient exact set-similarity joins. In *Proceedings of the 32nd international conference on Very large data bases*. 918–929.

[2] Roberto J Bayardo, Yiming Ma, and Ramakrishnan Srikant. 2007. Scaling up all pairs similarity search. In *Proceedings of the 16th international conference on World Wide Web*. 131–140.

[3] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis. 2012. Spatio-textual similarity joins. *Proceedings of the VLDB Endowment* 6, 1 (2012), 1–12.

[4] Aniket Chakrabarti, Venu Satuluri, Atreya Srivathsan, and Srinivasan Parthasarathy. 2015. A bayesian perspective on locality sensitive hashing with extensions for kernel methods. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10, 2 (2015), 1–32.

[5] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. 2006. A primitive operator for similarity joins in data cleaning. In *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 5–5.

[6] Tobias Christiani and Rasmus Pagh. 2017. Set similarity search beyond minhash. In *Proceedings of the 49th annual ACM SIGACT symposium on theory of computing*. 1094–1107.

[7] Dong Deng, Guoliang Li, He Wen, and Jianhua Feng. 2015. An efficient partition based method for exact set similarity joins. *Proceedings of the VLDB Endowment* 9, 4 (2015), 360–371.

[8] Dong Deng, Yufei Tao, and Guoliang Li. 2018. Overlap set similarity joins with theoretical guarantees. In *Proceedings of the 2018 International Conference on Management of Data*. 905–920.

[9] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.

[10] Daniel Kocher, Nikolaus Augsten, and Willi Mann. 2021. Scaling Density-Based Clustering to Large Collections of Sets. In *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, Yannis Velegrakis, Demetris Zeinalipour-Yazti, Panos K. Chrysanthis, and Francesco Guerra (Eds.). OpenProceedings.org, 109–120. https://doi.org/10.5441/002/edbt.2021.11

[11] Chen Li, Jiaheng Lu, and Yiming Lu. 2008. Efficient merging and filtering algorithms for approximate string searches. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 257–266.

[12] Qiyu Liu, Yanyan Shen, and Lei Chen. 2022. HAP: An Efficient Hamming Space Index Based on Augmented Pigeonhole Principle. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 917–930. https://doi.org/10.1145/3514221.3517880

[13] Willi Mann and Nikolaus Augsten. 2014. PEL: Position-Enhanced Length Filter for Set Similarity Joins. In *Grundlagen von Datenbanken*. Vol. 1313. 89–94.

[14] Willi Mann, Nikolaus Augsten, and Panagiotis Bouros. 2016. An empirical evaluation of set similarity join techniques. *Proceedings of the VLDB Endowment* 9, 9 (2016), 636–647.

[15] Samuel McCauley, Jesper W Mikkelsen, and Rasmus Pagh. 2018. Set similarity search for skewed data. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 63–74.

[16] Mohammad Norouzi, Ali Punjani, and David J Fleet. 2012. Fast search in hamming space with multi-index hashing. In *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 3108–3115.

[17] Mohammad Norouzi, Ali Punjani, and David J Fleet. 2013. Fast exact search in hamming space with multi-index hashing. *IEEE transactions on pattern analysis and machine intelligence* 36, 6 (2013), 1107–1119.

[18] Rasmus Pagh. 2016. Locality-sensitive hashing without false negatives. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 1–9.

[19] Casper Petersen, Jakob Grue Simonsen, and Christina Lioma. 2016. Power law distributions in information retrieval. *ACM Transactions on Information Systems (TOIS)* 34, 2 (2016), 1–37.

[20] Ninh Pham and Rasmus Pagh. 2016. Scalability and total recall with fast CoveringLSH. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1109–1118.

[21] Jianbin Qin and Chuan Xiao. 2018. Pigeonring: A Principle for Faster Thresholded Similarity Search. *Proc. VLDB Endow.* 12, 1 (2018), 28–42. https://doi.org/10.14778/3275536.3275539

[22] Jianbin Qin, Chuan Xiao, Yaoshu Wang, Wei Wang, Xuemin Lin, Yoshiharu Ishikawa, and Guoren Wang. 2019. Generalizing the pigeonhole principle for similarity search in Hamming space. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[23] Chuitian Rong, Wei Lu, Xiaoli Wang, Xiaoyong Du, Yueguo Chen, and Anthony KH Tung. 2012. Efficient and scalable processing of string similarity join. *IEEE Transactions on Knowledge and Data Engineering* 25, 10 (2012), 2217–2230.

[24] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. ArnetMiner: Extraction and Mining of Academic Social Networks. In *KDD'08*. 990–998.

[25] Paul Turán. 1941. Eine Extremalaufgabe aus der Graphentheorie. *Mat. Fiz. Lapok* 48, 436-452 (1941), 61.

[26] Jiannan Wang, Guoliang Li, and Jianhua Feng. 2012. Can we beat the prefix filtering? An adaptive framework for similarity join and search. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. 85–96.

[27] Pei Wang, Chuan Xiao, Jianbin Qin, Wei Wang, Xiaoyang Zhang, and Yoshiharu Ishikawa. 2016. Local Similarity Search for Unstructured Text. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 1991–2005. https://doi.org/10.1145/2882903.2915211

[28] Xubo Wang, Lu Qin, Xuemin Lin, Ying Zhang, and Lijun Chang. 2019. Leveraging set relations in exact and dynamic set similarity join. *The VLDB Journal* 28, 2 (2019), 267–292.

[29] Chuan Xiao, Wei Wang, Xuemin Lin, and Haichuan Shang. 2009. Top-k set similarity joins. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 916–927.

[30] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. 2011. Efficient similarity joins for near-duplicate detection. *ACM Transactions on Database Systems (TODS)* 36, 3 (2011), 1–41.

[31] Xiaoyang Zhang, Jianbin Qin, Wei Wang, Yifang Sun, and Jiaheng Lu. 2013. Hmsearch: An efficient hamming distance query processing algorithm. In *Proceedings of the 25th international conference on scientific and statistical database management*. 1–12.

[32] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J Miller. 2019. Josie: Overlap set similarity search for finding joinable tables in data lakes. In *Proceedings of the 2019 International Conference on Management of Data*. 847–864.