Towards a Unified Framework for String Similarity Joins

Pengfei Xu
Department of Computer Science, University of Helsinki, Finland
pengfei.xu@helsinki.fi

Jiaheng Lu
Department of Computer Science, University of Helsinki, Finland
jiaheng.lu@helsinki.fi

ABSTRACT

A similarity join aims to find all similar pairs between two collections of records. Established algorithms utilise different similarity measures, either syntactic or semantic, to quantify the similarity between two records. However, when records are similar in forms of a mixture of syntactic and semantic relations, utilising a single measure becomes inadequate to disclose the real similarity between records, and hence unable to obtain high-quality join results.

In this paper, we study a unified framework to find similar records by combining multiple similarity measures. To achieve this goal, we first develop a new similarity framework that unifies the existing three kinds of similarity measures simultaneously, including syntactic (typographic) similarity, synonym-based similarity, and taxonomy-based similarity. We then theoretically prove that finding the maximum unified similarity between two strings is generally $\mathcal{NP}$-hard, and furthermore develop an approximate algorithm which runs in polynomial time with a non-trivial approximation guarantee. To support efficient string joins based on our unified similarity measure, we adopt the filter-and-verification framework and propose a new signature structure, called *pebble*, which can be simultaneously adapted to handle multiple similarity measures. The salient feature of our approach is that, it can judiciously select the best pebble signatures and the overlap thresholds to maximise the filtering power. Extensive experiments show that our methods are capable of finding similar records having mixed types of similarity relations, while exhibiting high efficiency and scalability for similarity joins. The implementation can be downloaded at `https://github.com/HY-UDBMS/AU-Join`.

PVLDB Reference Format:
Pengfei Xu and Jiaheng Lu. Towards a Unified Framework for String Similarity Joins. *PVLDB*, 12(11): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/3342263.3342268

1. INTRODUCTION

Given two collections of records, a similarity join aims to find all of which have similarities higher than a given threshold. Such operation is widely adopted in tasks such as data cleansing [2, 56], information retrieval [9], and data mining [32]. A plethora of similarity measures have been proposed to measure the similarity between two records, to name a few:

- Gram-based similarity measures two string's literal likeness by splitting each into a set of fixed-length grams, and then counts the intersected items between both sets. Many measures have been proposed, e.g. Jaccard coefficient [15, 31], Cosine similarity [10, 46], Hamming distance [59] and Dice similarity [9]. This kind of measure can be used to detect typographic mistakes, such as typos.
- Synonym similarity [32, 33, 58] measures the similarity of strings by leveraging a set of predefined synonym (or abbreviation) rules between strings. For example, "Bill" is a nickname of "William" and "DBMS" is an abbreviation of "Database Management System".
- Taxonomy similarity [47, 48, 55, 57] measures the similarity of strings by using a *knowledge hierarchy*, an abundant source of IS-A relations. For example, "kitten" is a kind of "pet", and "iPhone" is a type of "smartphone". Thanks to the public knowledge bases, e.g. Freebase [50], DBpedia [6], Wikipedia [21], and Yago [49], one can use these available taxonomies to enhance the effectiveness of string similarity matching.

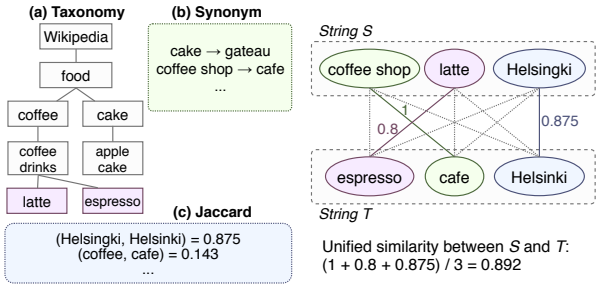


Figure 1: Example of similarity calculations on strings containing synonym, misspelling, and taxonomic-relevant terms.

It is worth noting that, each of the above measures alone can handle only one specific type of similarity. For example, Jaccard coefficient is restricted to typographic (spelling) errors, while a synonym-based measure is unable identify IS-A relations. However, in real-world applications, two strings often involve more than one type of similarity [1, 22, 34, 43]. Consider the example in Figure 1, where we have two strings of *Points of Interests* (POIs): "coffee shop latte Helsingki" and "espresso cafe Helsinki". They are both referring to coffee shops in Helsinki thus highly relevant. Here, three different kinds of similarity relations are involved: (i) syntactic: "Helsingki" is a misspelt "Helsinki"; (ii) synonym: "cafe" is equivalent to "coffee shop"; (iii) taxonomy: both "latte" and "espresso" belong to coffee drinks. Note that a single measure can

Proceedings of the VLDB Endowment, Vol. 12, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3342263.3342268

only surface a portion of similarity and hence unfeasible to obtain the full knowledge of their relation. Naturally, an open question arises: *Is there a unified similarity measure that can capture multiple kinds of similarity relations simultaneously?* Note that this problem cannot be solved trivially by combing the results from different similarity measures, because such a method cannot apply multiple measures on different segments of one string simultaneously, as in the above POI example.

This paper addresses that open question by designing a new similarity measure combining all aforementioned measures. In particular, given two strings S and T, we calculate the unified similarity in two phases: (i) compute individual similarities between two *segments* (substrings) of S and T with different measures respectively, and then (ii) aggregate them together to compute the final similarity by finding the maximum matching in a bipartite graph, where the weight of each edge is the individual segment similarity. Note that the standard maximum matching in a bipartite graph admits polynomial algorithms, e.g., Hungarian algorithm [39] in $O(n^3)$ time, where n is the maximal number of vertices in one-side of a bipartite graph. Unfortunately, our careful investigation reveals that the unified similarity problem here is more complicated than the existing maximum matching, because one substring may participate in multiple synonym or taxonomy rules, leading to different options to segment the string and generate bipartite graphs:

EXAMPLE 1. *The purpose of this example is to show that a unified similarity measure is harder than the maximum matching problem in a bipartite graph. Recall two strings in Figure 1. One can either match the substring "coffee shop" of S with a synonym rule "coffee shop → cafe", or split it into two segments "coffee" and "shop", and "coffee" matches to an entity in the taxonomy hierarchy to obtain IS-A relation between "coffee" and "espresso". In the latter case, an edge exists between "coffee" and "espresso" in the corresponding bipartite graph, and hence we can obtain a different similarity. Therefore, in general, one has to enumerate all possible bipartite graph instances to obtain the maximum similarity, which may lead to the inherent intractability in the worst case.*

We theoretically prove that finding the *maximum similarity* between two strings can be intractable, i.e., $\mathcal{NP}$-hard, in light of the exponential combinations of segment partitions based on the synonym rules (or taxonomy hierarchy). However, one good news is that we develop a polynomial-time algorithm with an approximation bound, motivated by previous algorithms proposed for the *weighted Maximum Independent Set* (w-MIS) problem in a *k-claw-free* graph [5,7,26]. Further, our empirical experiments demonstrate the effectiveness and efficiency of our approximation algorithm, indicating its promising theoretical and practical significances.

Given a unified similarity measure, it is imperative to study efficient join algorithms for two large set of strings. To this end, we employ a *filter-and-verification* framework by pruning irrelevant string pairs and then verifying the similarity of survived ones. Since the computation of the similarity between strings is an expensive operation, it is critical to design an effective filtering strategy to prune away strings as many as possible. In literature, a widely-adopted filtering principle is the *prefix filtering* [4,9,15], which disqualifies dissimilar pairs by comparing their signatures (i.e. prefixes). However, this filter cannot be directly applied to our problem because the selection of signatures depends on a specific similarity measure, whereas our problem combines multiple measures. Therefore, we build a novel unified data structure, called *pebble*, which can be adaptive to different similarity measures: for Jaccard coefficient, pebbles are q-grams of strings; in synonym-based similarity, pebbles are the left-hand side of synonym rules; in taxonomy-based similarity, pebbles refer to sets of taxonomy entities. We further propose a family of algorithms by utilising heuristics and dynamic programming to select appropriate pebbles as signatures to maximise the overall filtering effectiveness.

In the filter-and-verification framework, there is a key parameter τ controlling the minimal number of overlapped signatures (i.e. pebbles in our paper) two strings must have to survive from filtering. Intuitively, a smaller τ leads to a faster filtering due to smaller index sizes, but more candidates for verification. On the other hand, a greater τ can lead to a longer filtering time but faster verification through reduced candidates. Therefore, it is crucial to select the best τ to strike a balance between filtering and verification time. In this paper, we propose a sampling-based framework to estimate the intermediate results and compute the optimal τ parameter, without *a priori* knowledge of dataset characteristics. Based on *independent Bernoulli sampling* and *Monte Carlo simulation* framework, our algorithm runs multiple iterations, each of which uses small samples to ensure accurate suggestions of τ with far fewer calculations. We show that our estimator is unbiased and analyse its variance theoretically. Experiments on real-world datasets show our estimator can suggest the best parameter with higher than 90% accuracy, by using only 0.003% samples out of 3.5 million records in each iteration of sampling, which finally occupies less than 1% of total join time.

To summarise, our technical contributions and the organisation of this paper are as follows:

- We propose a unified similarity measure that combines multiple similarity measures (Section 2). We prove the $\mathcal{NP}$-hardness to compute the maximum unified similarity between two strings and propose a polynomial-time algorithm to achieve a worst-case approximation guarantee.
- To support efficient similarity string joins based on our unified measure, we present a novel structure *pebble* to provide a unified solution for judiciously selecting high-quality signatures, which prunes away most of the dissimilar pairs to significantly improve the joining efficiency (Section 3).
- Since the threshold of overlapped signatures τ is a critical parameter affecting the join effectiveness, we design a sampling-based framework to dynamically estimate the join carnality of intermediate results, ensuring an accurate suggestion of the best τ to further optimise the joining efficiency (Section 4).
- Finally, comprehensive experiments in Section 5 show the effectiveness, efficiency, and scalability of our proposed algorithms, which strongly motives for its use in practice.

2. SIMILARITY MEASURES

This section explores a unified similarity measure and its computation algorithm. In particular, Section 2.1 provides preliminary definitions and notations, Section 2.2 proposes a generalised framework to unify the existing diverse measures, and finally, Section 2.3 provides a polynomial-time algorithm to approximate this unified similarity.

2.1 Preliminaries

An *alphabet* Σ is a finite non-empty set of symbols called *letters*. A *string* S over Σ is a finite sequence $S = s_1, \cdots, s_n$ of letters. There are many good reasons to define the similarity between strings [17]. For example, it enables users to capture the notion that one string is a typo of another, or to classify similar strings. We show here some of these similarities:

Gram-based similarity. Let $S[i, j]$ denotes the substring of S from its i-th to j-th letter. Then, given a positive integer q, a q-gram of

S is a q-letter substring of S, i.e., $g = S[i, i+q-1]$. The set of q-grams of S, denoted by $G(S,q)$, is the collection of all q-grams of S. The similarity of two strings is defined by the intersection of two gram collections. A common measure, called *Jaccard coefficient*, measures the typographic differences of two strings by counting their common q-grams:

$$sim_j(S,T) = \frac{|G(S,q) \cap G(T,q)|}{|G(S,q) \cup G(T,q)|} \tag{1}$$

Synonym similarity. One can identify the semantic similarity between strings given a set of synonym rules. Formally, let R denote a rule in the form of $lhs(R) \rightarrow rhs(R)$, where $lhs(R)$ and $rhs(R)$ are called *left-* and *right-hand side* of R, respectively. Let $C(R) \in (0,1]$ indicate the closeness between $lhs(R)$ and $rhs(R)$, and let $\mathcal{R}$ be a set of synonym rules. Given $\mathcal{R}$, the synonym similarity between two strings S and T is:

$$sim_s(S,T) = \begin{cases} C(R), & \text{if } \exists R \in \mathcal{R} \text{ s.t. } lhs(R) = S \text{ and } rhs(R) = T \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

Taxonomy similarity. The third category of similarity measure is to identify the semantic similarity between strings by leveraging knowledge from a taxonomy hierarchy, which can be modelled as a tree structure. Given two string S and T mapped to taxonomy nodes n_S and n_T, and let $|n_S|$ ($|n_T|$) denotes the depth of n_S (n_T). Then, the similarity between S and T can be measured based on the depth of their *lowest common ancestor* (LCA):

$$sim_t(S,T) = \frac{|LCA(n_S, n_T)|}{\max\{|n_S|, |n_T|\}} \tag{3}$$

Example 2. *We use this example to illustrate above similarity measures. (i) Given strings S: "Helsingki" and T: "Helsinki", where $G(S,2) = \{$He, el, ls, si, in, ng, gk, ki$\}$ and $G(T,2) = \{$He, el, ls, si, in, nk, ki$\}$, then $sim_j(S,T) = 6/9 = 2/3$. (ii) Given S: "coffee shop" and T: "cafe", by assuming $C(R) = 1$ in Equation (2), $sim_s(S,T) = 1$ since they match the lhs and rhs of a synonym rule R in Figure 1(b). (iii) Consider S: "latte" and T: "espresso". Their LCA node is "coffee drinks" according to Figure 1(a), and hence $sim_t(S,T) = 4/5 = 0.8$.*

Remark. The above three similarities can be divided two categories: typographic (i.e., gram-based Jaccard) and semantic (i.e., synonym and taxonomy). As an example, "California" (a state of U.S.) have several semantic variations: "Calif.", "CA USA", "Golden State", "Golden Bear State", etc., as well as "Callifornia" and "Calfornia" which occur unintentionally due to typographic errors. In this paper, we will propose a new framework to catch all above variations and errors by unifying both typographic and semantic similarities.

2.2 Unified String Similarity Measure

In this subsection, we study a generalised framework which captures all the above similarities. Intuitively, given two strings S and T, one can define a unified measure as in Equation (4) to select the maximum among all possible similarity measures:

$$msim(S,T) = \max_{\forall f} \{sim_f(S,T)\} \tag{4}$$

As an example, the Jaccard coefficient between "cake" and "apple cake" is 0.33, but their taxonomy similarity is 0.75 according to Figure 1(a). Based on Equation (4), their final similarity is max{0.33, 0.75}=0.75. However, this straightforward approach cannot handle a complicated case where multiple similarity measures are simultaneously applicable to different parts of strings. Therefore, a better idea is to split strings into multiple segments to match different similarity measures, and then compute the final similarity through the maximum matching in a bipartite graph [47]. Further, since there are multiple ways to split each string and generate bipartite graph (as illustrated in Example 1 in Section 1), the final similarity could be defined as the maximum match among all possible bipartite graphs. In the following, we formalise the above intuitions with several definitions.

A string can be tokenised into multiple *tokens* with respect to a delimiter, e.g. empty space. For example, "coffee shop" can be tokenised into two tokens "coffee" and "shop".

Definition 1 (Well-Defined Segment). *Given a string S, a well-defined segment P_S is a sequence of consecutive tokens of S which (i) can map to the lhs or rhs of a synonym rule, or (ii) can match a corresponding taxonomy entity, or (iii) contains exactly one token.*

Using the example in Figure 1, "coffee shop", "latte", and "Helsingki" are three well-defined segments of string S, since they match the *lhs* of a synonym rule, a taxonomy node, or contains a single token. In contrast, "shop latte" is not a well-defined segment.

Definition 2 (Well-Defined Partition). *Given a string S, a well-defined partition $\mathcal{P}_S$ is a collection of well-defined segments of S such that (i) any two well-defined segments shares no common token and (ii) each token of S is in exactly one well-defined segment.*

Definition 3 (Unified Similarity). *Given two strings S and T, their unified similarity, denoted by $USIM(S,T)$, is defined as:*

$$USIM(S,T) = \max_{\forall(\pi_S, \pi_T)} \{SIM(\mathcal{P}_S, \mathcal{P}_T)\} \quad \text{where } \pi_S : S \rightarrow \mathcal{P}_S,\ \pi_T : T \rightarrow \mathcal{P}_T \tag{5}$$

$$SIM(\mathcal{P}_S, \mathcal{P}_T) = \frac{\max \sum_{i=1}^{|\mathcal{P}_S|} \sum_{j=1}^{|\mathcal{P}_T|} I_{ij} \cdot msim(P_{S_i}, P_{T_j})}{\max\{|\mathcal{P}_S|, |\mathcal{P}_T|\}} \quad \text{where } I_{ij} = 0 \text{ or } 1, \sum_i I_{ij} \leqslant 1, \text{and } \sum_j I_{ij} \leqslant 1 \tag{6}$$

Let π_S denote a partition function which transforms a string S into a well-defined partition $\mathcal{P}_S$. The unified similarity *USIM* in Equation (5) is defined as the maximum similarity among all pairs of partitions $(\mathcal{P}_S, \mathcal{P}_T)$. Equation (6) finds the maximum weight matching in a bipartite graph by selecting (i.e. $I_{ij} = 1$) (or not selecting: $I_{ij} = 0$) the similarity between each segment pair (P_{S_i}, P_{S_j}) and make sure that each segment is selected at most once. The numerator of Equation (6) can be solved by using *Hungarian algorithm* [39] in polynomial $O(n^3)$ time, where n denotes the maximum number of vertices at one side of a bipartite graph.

Example 3. *We now illustrate Definition 3. Recall Strings S and T in Figure 1. There are two well-defined partitions for S: (i) $\{$coffee shop, latte, Helsingki$\}$ which has similarity with T equals to $(1 + 0.8 + 0.875)/3 = 0.892$ by Equation (6). (ii) $\{$coffee, shop, latte, Helsingki$\}$, which leads to $(0.333 + 0.8 + 0.875)/4 = 0.502$ by selecting (a) Jaccard: "coffee" vs "cafe", (b) Taxonomy: "latte" vs "espresso", and (c) Jaccard "Helsingki" vs "Helsinki". Finally, $USIM(S,T) = \max\{0.892, 0.502\} = 0.892$.*

Unfortunately, computing the unified similarity is in general $\mathcal{NP}$-hard, due to the potentially exponential ways to partition strings in the worst case.

Theorem 1. *$USIM \in \mathcal{NP}$-hard.*

The proof of Theorem 1 can be found in the Appendix. Briefly speaking, we present a polynomial-time reduction from the *Maximum Independent Set* problem, which is a well-known $\mathcal{NP}$-hard problem [5,7,26]. In light of the inherent intractability, we design an efficient polynomial-time algorithm with a bounded approximation guarantee as follows.

2.3 Approximation Algorithm

In this section, we present an algorithm that solves the *USIM* problem in polynomial time. As a high-level overview, we convert this problem to another problem called *weighted Maximum Independent Set* (*w*-MIS) on *k-claw-free* graphs, which admits an efficient approximation algorithm.

We first describe how to construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ based on two given strings S and T in three steps: (i) find all pairs of possible well-defined segments P_S, P_T (of S and T, respectively) such that (a) $P_S \rightarrow P_T$ or $P_T \rightarrow P_S$ is a synonym rule, or (b) P_S and P_T match two taxonomy entities, or (c) each of P_S and P_T contains a single token. (ii) Corresponding to each segment pair (P_S, P_T), add a vertex to $\mathcal{V}$ and assign it a weight equals to $msim(P_S, P_T)$ (see Equation 4). (iii) Add an edge to $\mathcal{E}$ between any two vertices that *conflicts*, meaning that their corresponding segments have a non-empty token intersection. In other words, two vertices $v = (P_S, P_T)$ and $v' = (P'_S, P'_T)$ conflicts if $(P_S \cap P'_S) \neq \emptyset$ or $(P_T \cap P'_T) \neq \emptyset$. If there is an edge between two vertices, then their corresponding rules cannot be applied simultaneously, as illustrated in Example 4.

Example 4. *The purpose of this example is to illustrate the construction of Graph $\mathcal{G}$. Given two strings S, T and six synonym rules in Figure 2(a), we can construct $\mathcal{G}$ as in Figure 2(b). There is an edge between any conflict vertex pair. For example, R_3 and R_5 are connected because they are sharing the token "d" and thus they cannot be applied on S simultaneously. Note that R_6 does not appear in $\mathcal{G}$ because it is not applicable on S.*

Recall that, in graph theory, a d-claw C is an induced subgraph that consists of d independent vertices called *talons* (denoted by $\mathcal{T}_C$), and one centre node connected to all d talons [11, 26]. We claim that the graph $\mathcal{G}$ from the above construction is a k+1-claw-free graph, where k is the maximal number of tokens in both sides of any synonym rule or taxonomy entity pair. This is because any vertex in $\mathcal{G}$ can only connect to at most k *mutually non-adjacent* vertices (though it can connect to more than k vertices). In other words, $\mathcal{G}$ poses no k+1-claw because talons must be non-adjacent by definition. Hence, $\mathcal{G}$ is k+1-claw-free.

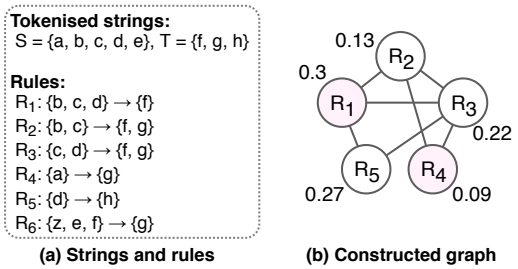
The *w*-MIS problem aims to find an independent vertex set $\mathcal{A} \subseteq \mathcal{E}$ which maximises $W(\mathcal{A}) = \sum_{u \in \mathcal{A}} w(u)$, where $w(u)$ is the weight of vertex u. MIS solutions can correspond to a pair of well-defined partitions on S and T, which, in turn, constructs a bipartite graph instance for computing the unified similarity. For example, recall Figure 2 where $\{R_1, R_4\}$ is an instance of MIS, which corresponds to two partitions: $\mathcal{P}_S$=$\{\{a\}, \{b, c, d\}, \{e\}\}$ and $\mathcal{P}_S$=$\{\{f\}, \{g\}, \{h\}\}$.

The *w*-MIS problem is known to be $\mathcal{NP}$-hard, and it cannot be approximated within any constant factor for general graphs unless $\mathcal{P} = \mathcal{NP}$ [25, 38]. However, when it is restricted to k-claw-free graphs, the polynomial-time approximation of *w*-MIS becomes possible by treating k as a constant. This problem has been researched intensively in the past few decades [5, 7, 11, 13, 26]. The state-of-art method is SquareImp [11], which can achieve the approximation ratio $\frac{t}{t-1} \cdot \frac{k+1}{2}$ in time that is polynomial in $t \cdot n$, where $t > 1$ is a tunable parameter that enables trading off running time and approximating accuracy, and n is the number of vertices in $\mathcal{G}$. In this paper, we extend SquareImp to solve our problem to achieve a non-trivial approximation guarantee.

Algorithm 1: Approximation algorithm

Input: two strings S,T, and their associated graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Output: the unified similarity between S and T

1 compute the maximum independent set $\mathcal{A}$ of $\mathcal{G}$ with SquareImp [11]
2 define $N(\mathcal{R}, \mathcal{A}) = \{u \in \mathcal{R} : \exists v \in \mathcal{A}$ such that $(u, v) \in \mathcal{E}$ or $u = v\}$
3 **while** *there exists a claw $C \subset \mathcal{G}$ s.t. $\mathcal{T}_C$ improves the unified similarity at least $1/t$* **do**
4 $\quad \mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{T}_C \setminus N(\mathcal{T}_C, \mathcal{A})$, where C is a claw that improves the most GetSim($\mathcal{A}$)
5 **return** *GetSim*($\mathcal{A}$)
6 **Function** GetSim($\mathcal{A}$):
7 $\quad \mathcal{P}_S, \mathcal{P}_T \leftarrow$ partitions of S and T constructed from $\mathcal{A}$
8 **return** *SIM*($\mathcal{P}_S, \mathcal{P}_T$)



(a) Strings and rules **(b) Constructed graph**

Figure 2: Illustration of Example 5. Numbers beside each vertex is its weight. SquareImp selects R_2 and R_5, whereas Alg. 1 finally selects R_1 and R_4.

Algorithm 1 shows the process of the unified similarity approximation. Given two strings S, T and their associated graph $\mathcal{G}$, the algorithm first computes the *w*-MIS solution using SquareImp algorithm (Line 1). Then, in Line 3, we say talons $\mathcal{T}_C$ "improves" the similarity if adding all $u \in \mathcal{T}_C$ and removing all neighbourhoods of u (i.e. $N(\mathcal{T}_C, \mathcal{A})$) can bring a higher similarity (i.e. GetSIM($\mathcal{A}$)). Note that, by specifying the size of the minimal allowed improvement to $1/t$ ($t > 1$), Lines 3-4 can run in polynomial time, ensuring the termination within $\lfloor t \rfloor$ iterations. Finally, Line 5 constructs two partition functions (π_S, π_T) based on solution $\mathcal{A}$ and returns the final unified similarity.

Example 5. *Given two strings and synonym rules in Figure 2(a). We construct a 5-claw-free graph in Figure 2(b). When applying Algorithm 1 on this graph, Line 1 (i.e.* SquareImp*) selects R_2, R_5, and hence two partitions $\mathcal{P}_S = \{\{a\}, \{b, c\}, \{d\}, \{e\}\}$, $\mathcal{P}_T = \{\{f, g\}, \{h\}\}$ that lead to the similarity* $(0.13 + 0.27)/4 = 0.1$. *Then, Lines 3-4 of Algorithm 1 find another claw with centre R_2 and talons R_1, R_4 can improve the similarity by partitioning two strings as $\mathcal{P}_S = \{\{a\}, \{b, c, d\}, \{e\}\}$, $\mathcal{P}_T = \{\{f\}, \{h\}, \{g\}\}$ that has similarity* $(0.3 + 0.09)/3 = 0.13$. *Therefore, the final returned result is 0.13.*

Analysis. The approximate bound of Algorithm 1 is different from SquareImp, because the unified similarity is computed by the solution of maximum bipartite graph match divided by the maximal number of segments of S and T (see Equation (6)). The optimisation of *w*-MIS can only achieve the best approximation for the bipartite graph match, but cannot determine the number of segments. Therefore, the following theorem shows the adjusted worst-case bound. The proof can be found in the Appendix.

Theorem 2. *Given two strings S and T, Algorithm 1 computes a solution for the unified similarity between S and T with a tight approximation ratio $\frac{t}{t-1} \frac{k^2-1}{2}$ in time that is polynomial on $t \cdot n$, where $t > 1$, n is the number of vertices in $\mathcal{G}$ constructed by S and T, and k is the maximal number of tokens in any applicable synonym rule or taxonomy entity pair.*

Table 1: Table of notations used in Section 3.

Notation	Description
B,$w(B)$	a pebble and its associated weight
$\mathcal{B}$	a sorted list of pebbles
$\mathcal{B}[i, j]$	the subsequence of the i-th to j-th pebbles of $\mathcal{B}$
$\mathcal{B}_{P,f}[i, j]$	all pebbles in $\mathcal{B}[i, j]$ that are generated from P by f
$MP(S)$	the min number of well-defined segments for any partition of S
$W(\mathcal{B})$	the weight sum of all pebbles in $\mathcal{B}$. See Eq. (7)
$TW_k(\mathcal{B})$	the weight sum of top-k heaviest pebbles in $\mathcal{B}$. See Eq. (8)

3. THE UNIFIED JOIN FRAMEWORK

In this section, we study the string similarity join problem based on the proposed unified similarity measure. Given two collections of strings $\mathcal{S}$ and $\mathcal{T}$, we want to find all string pairs $(S, T) \in \mathcal{S} \times \mathcal{T}$ such that $USIM(S, T) \geqslant \theta$, where $\theta \in [0, 1]$ is a predefined join threshold and $USIM$ is the similarity measure defined in Definition 3.

Roughly speaking, our solution is based on the *filter-and-verification* framework. That is, in the filtering step, the algorithm generates candidate string pairs by identifying all pairs (S, T) such that signatures of S and T overlap. In the verification step, we check if $USIM(S, T) \geqslant \theta$ for each candidate and outputs those satisfying the similarity predicate. In the rest of this section, we will propose a family of signature-based filtering and join algorithms.

3.1 U-Filter: Unified Signature Filtering

One efficient filtering technique is *prefix filtering*: given a string S, generate its prefix signature as its first $(1 - \theta)|S| + 1$ tokens sorted by a predefined global order, such as by *inverse document frequency* (IDF) [44, 47]. Then, any two similar strings must share at least one token within their signatures. This condition is necessary but not sufficient for two strings being similar, thus verification is required to check real similarities of survivors.

The above existing filtering principle, unfortunately, cannot be straightforwardly applied to our problem due to two challenges. First, *how to represent different types of similarity measures in a unified, coherent way*? Second, *how to select signatures of strings such that the overlapping requirement of prefix filtering still holds*? We address these two challenges by proposing a new structure as well as a new filtering strategy.

Pebbles. We introduce an abstract concept called *pebble*, to represent various similarity types in a unified yet elegant manner. Intuitively, given a segment P, we can generate pebbles B and their associative weight $w(B)$ w.r.t. different similarity measures:

- Jaccard coefficient: each q-gram of P is a pebble B. The weight of each B is $w(B) = 1/|G(P, q)|$.
- Synonym similarity: assume that the applicable synonym rule of P is R, then the pebble B of P is $lhs(R)$, and $w(B) = C(R)$.
- Taxonomy similarity: let n be the matching taxonomy entity of P. The pebbles of P include n and all its ancestors, each has weight $w(B) = 1/|n|$.

Table 2 exemplifies pebbles of segments "coffee" and "cafe" for three similarity measures in Figure 1. Given a string S, we first generate all pebbles and sort them by a global frequency order. Then, we remove as many pebbles (those having higher frequencies) as possible and select the remaining pebbles as the signature of S. Below, we precisely quantify the contribution of pebbles to compute the final similarity.

Given a sorted pebble list $\mathcal{B}$, let $\mathcal{B}[i, j]$ $(1 \leqslant i \leqslant j \leqslant |\mathcal{B}|)$ denotes a sublist from its i-th to j-th pebbles, and let $\mathcal{B}_{P,f}[i, j]$ denotes a list containing pebbles from $\mathcal{B}[i, j]$ that are generated from segment P by measure f. Further, let

Table 2: Illustrating pebbles and their associative weights.

Measure		Jaccard	Synonym	Taxonomy
Pebble type		q-grams	*lhs* of the rule	ancestor nodes
"coffee"	Pebbles	{co, of, ff, fe, ee}	-	{Wikipedia, food, coffee}
	Weight	1/5	-	1/3
"cafe"	Pebbles	{ca, af, fe}	{coffee shop}	-
	Weight	1/3	1	-

$$W(\mathcal{B}) = \sum_{\forall B \in \mathcal{B}} w(B) \tag{7}$$

$$TW_k(\mathcal{B}) = \sum_{\forall B \in topk(\mathcal{B},k)} w(B) \tag{8}$$

Intuitively, $W(\mathcal{B})$ returns the sum of weights of pebbles in $\mathcal{B}$, while $TW_k(\mathcal{B})$ returns the sum of weights of top-k heaviest pebbles (in terms of weight) in $\mathcal{B}$. As an example, for string "cafe" in Table 2, $W(\mathcal{B})$=(1/3)×3+1=2 (assuming pebbles are sorted as in the table) and $TW_2(\mathcal{B})$ = 1+1/3=4/3.

Definition 4 (Accumulated Similarity). *Given a string S and a positive integer i, let P denote an arbitrary well-defined segment of S and let $\mathcal{B}[i, n]$ be the pebble sublist of S from the i-th element till the end. The Accumulated Similarity $AS(i, S)$ is calculated by:*

$$AS(i, S) = \sum_{\forall P} \max_{\forall f} W(\mathcal{B}_{P,f}[i, n]) \tag{9}$$

Note that the accumulated similarity can be greater than 1, because it sums up the individual similarities of all segments of S.

Signature selection. Algorithm 2 formally describes the procedure for selecting pebble signatures. Intuitively, given a string S, we remove as many pebbles as possible from the sorted pebble list $\mathcal{B}$ until the accumulated similarity of removed pebbles can reach certain thresholds (calculated through Function GetMinPartitionSize), by assuming that all removed pebbles exist in similar strings. We now go through Algorithm 2. Line 1 obtains all pebbles of S and sorts them by a global order (e.g., by the ascending order of frequencies of pebbles). Line 2 calculates the minimal number of partitions of S, which will be discussed in details in the next paragraph. Line 4 is the key step, which iteratively removes the last pebble from the list until the accumulative similarity of removed ones becomes big enough. Finally, Line 5 returns the first i (i.e. remaining) pebbles in $\mathcal{B}$ as the signature of S.

In Function GetMinPartitionSize, we calculate the minimal number of partitions in S (denoted $MP(S)$). To understand its motivation, recall the definition of unified similarity in Equation (6) where the denominator is the maximal number of partitions of S and T. Note that the minimal number of partitions of S is clearly a lower bound

Algorithm 2: Generating pebble signatures for U-Filter

```
   Input: a string S and a similarity threshold θ
   Output: the pebble signatures of S
1  B ← all pebbles of S sorted by a global order
2  m ← GetMinPartitionSize(S)
3  i ← |B|
4  while mθ > AS(i, S) do i ← i − 1            // remove the last pebble
5  return the first i pebbles of B
6  Function GetMinPartitionSize(S):
7      U ← all tokens of S
8      n ← the size of the largest segment; A ← ∅
9      while U ≠ ∅ do
10         P ← a well-defined segment of S that maximises |P ∩ U|
11         U ← U \ P;  A ← A ∪ {P}
12     return ⌈|A|/(ln n + 1)⌉            // estimation lower bound [29]
```

Table 3: Illustration to Example 6 and 7. J: Jaccard, S: Synonym, T: Taxonomy.

i	Pebble	P_1: "espresso"			P_2: "cafe"			P_3: "Helsinki"			AS	TW
		J	T	S	J	T	S	J	T	S		
23	es	**1/7**									0.143	1.667
22	ss	**2/7**									0.286	1.667
21	fe	**2/7**			**1/3**						0.619	1.667
20	Wikipedia	**2/7**	1/5		**1/3**						0.675	1.667
19	food	2/7	**2/5**		**1/3**						0.733	1.667
...	...	...	...		...			...			...	...
8	ca	4/7	**1**		**2/3**			**5/8**			2.292	1.476
7	ki	4/7	**1**		**2/3**			**6/8**			2.417	1.476

Algorithm 3: Unified set join with U-Filter

Input: $\mathcal{L}_\mathcal{S}$ and $\mathcal{L}_\mathcal{T}$ as two inverted lists constructed from token sets $\mathcal{S}$ and $\mathcal{T}$, a similarity threshold θ
Output: $\{(S,T) \in \mathcal{S} \times \mathcal{T} : USIM(S,T) \geqslant \theta\}$

1 $\mathcal{B} \leftarrow$ overlapped keys (pebbles) between $\mathcal{L}_\mathcal{S}$ and $\mathcal{L}_\mathcal{T}$
2 **foreach** $B \in \mathcal{B}$ **do** *// filtering*
3 $\quad \ell_\mathcal{S}, \ell_\mathcal{T} \leftarrow$ lists indexed by pebble B in $\mathcal{L}_\mathcal{S}$ and $\mathcal{L}_\mathcal{T}$
4 $\quad$ **foreach** $(S,T) \in \{(\ell_\mathcal{S} \times \ell_\mathcal{T})\}$ **do** *// pairs having one overlap*
5 $\quad\quad C \leftarrow C \cup \{(S,T)\}$
6 **foreach** $(S,T) \in C$ **do** *// verification*
7 $\quad$ **if** $USIM(S,T) \geqslant \theta$ **then** $\mathcal{R} \leftarrow \mathcal{R} \cup \{(S,T)\}$
8 **return** $\mathcal{R}$

of such denominator value. Furthermore, unfortunately, it is $\mathcal{NP}$-hard to find the optimal value of $MP(S)$ because it can be reduced to a *minimum exact cover problem* [24]: given a collection of subsets of a set $\mathcal{X}$, a minimum exact cover is to compute the minimum number of subsets in a subcollection $\mathcal{A}$ such that each element in $\mathcal{X}$ is contained in exactly one subset in $\mathcal{A}$. In GetMinPartitionSize, we tackle this problem by a greedy algorithm [29]. In particular, in the loop of Lines 9-11 in Algorithm 2, we iteratively add the segment that maximises the coverage on S. This greedy algorithm can guarantee the approximate bound $\ln n + 1$ [29], where n is the number of tokens in the longest segment.

Example 6. *This example illustrates Algorithm 2. As in Figure 1, given $\theta = 0.8$ and the string T, which contains 3 segments P_1="espresso", P_2="cafe", and P_3="Helsinki". Line 1 generates 23 pebbles. In Line 2, GetMinPartitionSize iteratively selects three segments P_1, P_2, P_3 and returns $m = \lceil 3/(\ln 1 + 1) \rceil = 3$. Next, the while loop in Line 4 starts removing the 23-th pebble. Table 3 shows the values of accumulated similarity (AS) in each iteration. Finally, the algorithm stops at $i = 7$ when $AS(7,T) = 2.417 > m\theta = 2.4$. Line 5 returns the first 7 remaining pebbles as the signature of T.*

Join algorithm. Algorithm 3 shows the pseudocode of join algorithm based on U-Filter. Given two collections of strings $\mathcal{S}$ and $\mathcal{T}$, for each collection, we build the inverted list using each pebble from string signatures as key and string IDs as the value list. In particular, Line 1 finds the common pebbles between two inverted lists. Lines 2-5 generate candidate string pairs which share at least one pebble. Finally, Lines 6-8 verify the real similarity of candidates to return final results. The correctness of Algorithm 3 is manifested by the following lemma.

Lemma 1 (U-Filter). *Given two strings S, T, and their associated pebble lists $\mathcal{B}_S$, $\mathcal{B}_T$ sorted by a global order. Let i, j be two smallest integers satisfying $\theta \cdot MP(S) > AS(i,S)$ and $\theta \cdot MP(T) > AS(j,T)$, where $MP(S)$ and $MP(T)$ denote the minimal partitions number of S and T, respectively. If $USIM(S,T) \geqslant \theta$, then the first $i-1$ pebbles in $\mathcal{B}_S$ and the first $j-1$ pebbles in $\mathcal{B}_T$ must have at least one overlap.*

3.2 AU-Filter: Adaptive Signature Filtering

In the previous section, we introduced U-Filter which finds candidate pairs that share at least one pebble among their signatures.

Algorithm 4: Pebble signature for AU-Filter (heuristics)

Input: a string S, an overlap constraint τ, a similarity threshold θ
Output: the pebble signatures of S

1 $\mathcal{B} \leftarrow$ all pebbles of S sorted by a global order
2 $m \leftarrow$ GetMinPartitionSize(S)
3 $i \leftarrow |\mathcal{B}|$
4 **repeat**
5 $\quad C \leftarrow$ top $\tau - 1$ heaviest pebbles among the first $i-1$ pebbles of $\mathcal{B}$
6 $\quad TW \leftarrow$ the sum of weights of pebbles in C
7 $\quad i \leftarrow i - 1$
8 **until** $m\theta \leqslant AS(i,S) + TW$ *// test i-th pebble against Ineq. (10)*
9 **return** *the first i pebbles of* $\mathcal{B}$

Such policy, however, may produce many false positives because one overlap is often far from being sufficient for two strings being *really* similar. Therefore, in this section, we aim to reduce the number of false positives by increasing the number of required overlapping pebbles. To this end, we define a positive integer τ, called *overlap constraint*, which specifies the minimal number of common pebbles a string pair must have for being considered as a candidate. In other words, given a string S, we choose its signatures which contain at least τ overlapped pebbles with any other similar string T.

The research challenge here is to select pebble signatures such that the signature cardinality is minimised whilst the given overlap constraint τ is still satisfied. In this section, we present two new signature selection methods to achieve such goal: the first one naturally extends Lemma 1 to support more overlaps between signatures, while the second one employs dynamic programming to find a tighter upper bound on the cardinality of signatures.

3.2.1 Heuristic signature selection

We now present the first method by extending U-Filter to AU-Filter in Algorithm 4. In a nutshell, this algorithm continuously removes pebbles from the pebble list such that any string similar with S must share at least τ pebbles among S's remaining pebbles. More specifically, given a string S, a join threshold θ, and a parameter τ, the algorithm first generates a sorted list of pebbles $\mathcal{B}$ of S (Line 1) and computes the lower bound of segment number $m = MP(S)$ (Line 2). Then, it iteratively removes pebbles from the list until the sum of (i) the current accumulated similarity of removed pebbles (i.e. $AS(i,S)$) and (ii) the sum of weights of the top $\tau - 1$ heaviest remaining pebbles (Lines 5-6) reaches $m\theta$.

Lemma 2 (AU-Filter (Heuristics)). *Given two strings S, T, and their associated pebble lists $\mathcal{B}_S$, $\mathcal{B}_T$ sorted by a global order. Let i and j be two smallest integers satisfying Inequalities (10) and (11) respectively for S and T. If $USIM(S,T) \geqslant \theta$, then the first $i-1$ pebbles in $\mathcal{B}_S$ and the first $j-1$ pebbles in $\mathcal{B}_T$ must have at least τ overlaps.*

$$\theta \cdot MP(S) > AS(i,S) + TW_{\tau-1}(\mathcal{B}[1,i-1]) \quad (10)$$

$$\theta \cdot MP(T) > AS(j,T) + TW_{\tau-1}(\mathcal{B}[1,i-1]) \quad (11)$$

Example 7. *This example illustrates Algorithm 4. Recall String T in Figure 1 and assume $\theta = 0.8$, $\tau = 4$. Line 1 generates 23 pebbles, and Line 2 gives m=3. Table 3 shows the accumulated similarity AS and TW in the repeat loop in Lines 4-8. As an example, when i=19, the top $\tau - 1 = 3$ heaviest remaining pebbles are "coffee shop" (the lhs of synonym rule of P_2, weight 1) and two grams "ca", "af" (each weight 2/3). Since $AS(19,T) = 0.733$, the 19th pebble cannot be removed because the right-hand-side of Inequality (10) is $0.733 + 1 + 2/3 = 2.4$, no less than $m\theta = 2.4$. Therefore, the algorithm returns the first 19 pebbles as the signature.*

3.2.2 Signature selection by dynamic programming

Table 4: Illustrating Example 8, the DP prefix selection method. $i = 19$. Settings are the same as in Example 7.

p	Segment	DP table $\mathbb{W}_{19}$				Accessory table $\mathbb{V}_{19}$			
		d: 0	1	2	3	c: 0	1	2	3
0	-	0	0	0	0	-	-	-	-
1	P_1: "espresso"	0	0.2	0.4	0.6	0	3/5-2/5	4/5-2/5	5/5-2/5
2	P_2: "cafe"	0	0.667	0.867	1.067	0	1-1/3	1-1/3	1-1/3
3	P_3: "Helsinki"	0	0.667	0.867	1.067	0	1/8-0	2/8-0	3/8-0

The above solution may generate signatures containing pebbles more than necessary. For example, recall Example 7 where $i = 19$, the pebble "food" is included in the signature set returned by Algorithm 4. However, we will show later in Example 8 that our new approach can safely remove this pebble and therefore obtains a smaller signature set. Specifically, our refined solution computes a tighter upper bound of the similarity by utilising *dynamic programming* (DP), which solves a problem by splitting it into a set of subproblems. We describe our subproblems, initialisation procedure, and recurrence function as follows:

Subproblems: We create subproblems as follows. Given a string S and an integer τ, let $p \in [0, t]$ and $d \in [0, \tau - 1]$ be two integers where t is the total number of segments of string S. Given an integer i, let $\mathbb{W}_i[p, d]$ be a tight upper bound of the increment of the similarity because of the insertion of d pebbles of the first p segments from $\mathcal{B}[1, i-1]$. Finally, $\mathbb{W}_i[t, \tau-1]$ computes the maximal similarity increment by adding $\tau-1$ pebbles from $\mathcal{B}[1, i-1]$. In other words, $\mathbb{W}_i[t, \tau - 1]$ returns a tighter bound than $TW_{\tau-1}(\mathcal{B}[1, i - 1])$ in Inequalities (10) and (11).

Initialisation: For all $p \in [0, t]$, we set $\mathbb{W}_i[p, 0] = 0$. For all $d \in [0, \tau - 1]$, we set $\mathbb{W}_i[0, d] = 0$.

Recurrence function: Consider the subproblem of computing a value for the entry $\mathbb{W}_i[p, d]$. We have $d+1$ options. For each option, we can add c ($c \leqslant d$) pebbles from the current p-th segment (while the other $d - c$ pebbles are from previous p-1 segments). To determine which c is the best option, we construct an *accessory table* $\mathbb{V}_i$ which presents the maximal similarity increment by adding c new pebbles of the p-th segment. The following formulas show the recurrence function:

$$\mathbb{W}_i[p, d] = \max_{c \in [0,d]} \left\{ \mathbb{W}_i[p - 1, d - c] + \mathbb{V}_i[p, c] \right\} \tag{12}$$

Let P denote the p-th segment. $\mathbb{V}_i[p, c]$ is calculated by:

$$\mathbb{V}_i[p, c] = R(P, i, c) - R(P, i, 0) \tag{13}$$

$$R(P, i, c) = \max_{\forall f} \left\{ W(\mathcal{B}_{P,f}[i, n]) + TW_c(\mathcal{B}_{P,f}[1, i - 1]) \right\} \tag{14}$$

Specifically, $R(P, i, c)$ selects the single measure that has the maximal similarity on the segment P by adding c more pebbles. It includes two parts: (i) the weight sum of all pebbles in $\mathcal{B}_{P,f}[i, n]$ and (ii) the weight sum of top c heaviest remaining pebbles in $\mathcal{B}_{P,f}[1, i - 1]$.

DP algorithm. Using the analysis above, Algorithm 5 shows the pseudocode for generating pebble signatures by DP. We first process the string and obtain the necessary information about segments, pebbles, and the partition bound (Lines 1-3). Then, during each iteration, we initialise two tables $\mathbb{W}_i$ and $\mathbb{V}_i$ (Line 6-8) and populate them row by row (Lines 9-12) according to the aforementioned formulas. Specifically, in Lines 13-14, we test whether the sum of (i) the current accumulated similarity $AS(i, S)$ and (ii) the increment from d inserted pebbles can reach $\theta \cdot MP(S)$. If *yes*, we terminate the process and return the first i pebbles as the signatures of S.

Example 8. *The purpose of this example is to show that DP method can return a signature set smaller than the previous heuristic method. Recall the setting in Example 7. In particular, consider the moment when $i = 19$. Table 4 illustrates the DP table $\mathbb{W}_{19}$ and the accessory table $\mathbb{V}_{19}$, e.g., $\mathbb{V}_{19}[1, 3]$=$R(P_1, 19, 3)$-$R(P_1, 19, 0)$=5/5-2/5=3/5. Note that by Algorithm 5, the 19th pebble can be safely removed because $AS(19, T)$+$\mathbb{W}_{19}[3, 3]$=0.733+1.067 = 1.8, less than $\theta \cdot MP(T)$=2.4. Compared with the previous heuristic method in Inequality (10) where $TW_3(\mathcal{B}[1, 18])$=1.667, DP finds a tighter similarity bound, i.e., $\mathbb{W}_{19}[3, 3]$=1.067< $TW_3(\mathcal{B}[1, 18])$.*

Algorithm 5: Pebble signature for AU-Filter (DP)

Input: a string S, an overlap constraint τ, a similarity threshold θ
Output: the pebble signatures of S

1 $\mathcal{M} \leftarrow$ all well-defined segments of S; $t \leftarrow |\mathcal{M}|$
2 $\mathcal{B} \leftarrow$ all pebbles of S sorted by a global order
3 $m \leftarrow$ `GetMinPartitionSize`(S)
4 $i \leftarrow |\mathcal{B}|$
5 **while** $i > 1$ **do**
6 $\quad \mathbb{W}_i \leftarrow$ initialise a DP table with $t + 1$ rows and τ columns
7 $\quad \mathbb{V}_i \leftarrow$ initialise a Accessory table with t rows and τ columns
8 $\quad \mathbb{W}_i[0, \_] \leftarrow 0$; $\mathbb{W}_i[\_, 0] \leftarrow 0$; $\mathbb{V}_i[\_, 0] \leftarrow 0$ *// init $\mathbb{W}$ and $\mathbb{V}$*
9 $\quad$ **foreach** $p \in [1, t]$ **do**
10 $\quad\quad$ compute the row $\mathbb{V}_i[p, \_]$ according to Eq. (13)
11 $\quad\quad$ **foreach** $d \in [1, \tau - 1]$ **do**
12 $\quad\quad\quad$ compute the cell $\mathbb{W}_i[p, d]$ according to Eq. (12)
13 $\quad\quad\quad$ **if** $AS(i, S) + \mathbb{W}_i[p, d] \geqslant m\theta$ **then** *// early termination*
14 $\quad\quad\quad\quad$ **return** *the first i pebbles of $\mathcal{B}$*
15 $\quad i \leftarrow i - 1$

Algorithm 6: Unified set join with AU-Filter

Input: $\mathcal{L}_\mathcal{S}$ and $\mathcal{L}_\mathcal{T}$ as two inverted lists constructed from string sets $\mathcal{S}$ and $\mathcal{T}$, a similarity threshold θ, a positive integer τ
Output: $\{(S, T) \in \{\mathcal{S} \times \mathcal{T}\} : USIM(S, T) \geqslant \theta\}$

1 $\mathcal{P} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset, \mathcal{R} \leftarrow \emptyset$
2 $\mathcal{B} \leftarrow$ overlapped keys between $\mathcal{L}_\mathcal{S}$ and $\mathcal{L}_\mathcal{T}$
3 **foreach** $B \in \mathcal{B}$ **do**
4 $\quad \ell_\mathcal{S}, \ell_\mathcal{T} \leftarrow$ lists indexed by B in $\mathcal{L}_\mathcal{S}$ and $\mathcal{L}_\mathcal{T}$
5 $\quad$ **foreach** $(S, T) \in \{\ell_\mathcal{S} \times \ell_\mathcal{T}\}$ **do**
6 $\quad\quad \mathcal{P} \leftarrow \mathcal{P} \cup \{(S, T)\}$
7 **foreach** $(S, T) \in \mathcal{P}$ **do** *// find τ overlaps*
8 $\quad$ **if** (S, T) *appears at least τ times in $\mathcal{P}$* **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{(S, T)\}$
9 **foreach** $(S, T) \in \mathcal{C}$ **do** *// verification*
10 $\quad$ **if** $USIM(S, T) \geqslant \theta$ **then** $\mathcal{R} \leftarrow \mathcal{R} \cup \{(S, T)\}$
11 **return** $\mathcal{R}$

Join algorithm with AU-Filter. Based on the signatures generated from the previous algorithm, we present the join procedure in Algorithm 6 which follows the filter-and-verification framework. Unlike U-Filter in Algorithm 3, AU-Filter selects candidate pairs that share at least τ overlapped pebbles to reduce the size of candidates hence improve the overall join efficiency (Lines 7-8).

4. PARAMETER RECOMMENDATION

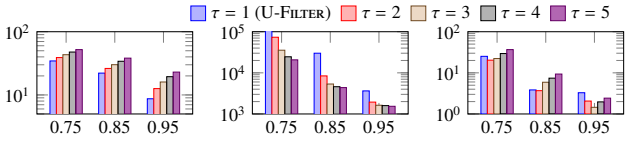
In AU-Filter, the overlap constraint τ is affecting the overall join time. Intuitively, a smaller τ leads to a faster filtering by generating shorter pebble signatures but a slower verification due to increased candidates. In contrast, a larger τ can lead to a slower filtering time (more signatures) but faster verification through the reduced number of candidates.

To demonstrate this trade-off, we performed a set of empirical experiments on two 20K subsets of MED dataset[1]. As shown in Figure 3(a), the size of signatures per string increases as τ becomes larger, yielding more filtering cost. Meanwhile, the number of candidate decreases since larger τ's have filtered out more pairs shown in Figure 3(b). Finally, Figure 3(c) demonstrates that, given

[1]Detailed info for this dataset can be found in Tables 6 and 7.

Table 5: Table of notations used in Section 4.

Notation	Description
T_τ	the number of processed pairs during filtering given full datasets
V_τ	the number of candidate pairs given full datasets
$\hat{T}_\tau, \hat{V}_\tau$	Bernoulli estimator of T_τ or V_τ
$\hat{\mu}_{T_\tau}, \hat{\mu}_{V_\tau}$	sample mean of estimations of $\hat{T}_\tau$ or $\hat{V}_\tau$
$\hat{\sigma}^2_{T_\tau}, \hat{\sigma}^2_{V_\tau}$	sample variance of estimations of $\hat{T}_\tau$ or $\hat{V}_\tau$
$T'^{(n)}_\tau$	the number of processed pairs when using the n-th sample
$\hat{T}^{(n)}_\tau, \hat{\mu}^{(n)}_{T_\tau}, \hat{\sigma}^{2,(n)}_{T_\tau}$	the value of $\hat{T}_\tau$, $\hat{\mu}_{T_\tau}$, or $\hat{\sigma}^2_{T_\tau}$ after the n-th iteration
C_τ	the overall cost of join with parameter τ



(a) Avg. signature length **(b) # of candidates** **(c) Join time (s)**

Figure 3: Different overlap constraints affecting join performance. Note that y-axes are in log scale.

different join thresholds, there exists a specific τ which gives the optimal join time. Therefore, it is fascinating to study how to dynamically select the best τ which minimises the join time. In this section, we will tackle this challenge by proposing an estimation and suggestion algorithm. Table 5 summarises used notations.

4.1 The Estimator

The overview of our strategy is to build a cost model and develop a sampling-based method to estimate the effect of different τ's on join time to select the optimal value.

Cost model. Our first effort is to build a cost model of join time. In particular, given two collections of strings $\mathcal{S}, \mathcal{T}$, and a positive integer τ, the total cost of joining $\mathcal{S}$ and $\mathcal{T}$ using Algorithm 6 consists of two parts, i.e., *filtering* and *verification*: (i) *Filtering*: the algorithm traverses common keys between two inverted lists to find string pairs having τ overlapped pebbles. Assume that the average cost for processing one pair is c_f. Therefore, the overhead is $c_f \cdot \sum_{p \in (\mathcal{L}_\mathcal{S} \cap \mathcal{L}_\mathcal{T})} \left(\left|\ell(\mathcal{L}_\mathcal{S}, p)\right| \cdot \left|\ell(\mathcal{L}_\mathcal{T}, p)\right| \right)$, where p is a common pebble between two inverted lists, $\ell(\mathcal{L}_\mathcal{S}, g)$ ($\ell(\mathcal{L}_\mathcal{T}, g)$) denotes a list of strings indexed by the key g from the inverted lists $\mathcal{L}_\mathcal{S}$ ($\mathcal{L}_\mathcal{T}$). (ii) *Verification*: similarity computation is performed on all candidates. Let c_v denote the average cost for verifying one string pair and V_τ be the number of candidates, the verification overhead is therefore $c_v V_\tau$. Let C_τ denotes the total cost of the algorithm for τ.

$$C_\tau = C_{T_\tau} + C_{V_\tau} = c_f T_\tau + c_v V_\tau \tag{15}$$

$$T_\tau = \sum_{p \in (\mathcal{L}_\mathcal{S} \cap \mathcal{L}_\mathcal{T})} \left(\left|\ell(\mathcal{L}_\mathcal{S}, p)\right| \cdot \left|\ell(\mathcal{L}_\mathcal{T}, p)\right| \right) \tag{16}$$

In the above equations, c_f and c_v can be assumed as two constants which are insensitive to τ. Therefore, the key task is to dynamically estimate the cardinality of T_τ and V_τ given two datasets.

Bernoulli estimator. It is certainly unfeasible to run the join algorithm on full datasets to get the exact values of T_τ and V_τ. Instead, we can estimate these values using the *independent Bernoulli sampling* [53], where each string in the input collection $\mathcal{S}$ ($\mathcal{T}$) has probability p_s (p_t) for being in the sample. Therefore, a pair (S, T) has the probability $p_s \cdot p_t$ for being sampled out. Then, the estimated value $\hat{T}_\tau$ and $\hat{V}_\tau$ can be computed as follows, where T'_τ and V'_τ denote the corresponding values computed in the sampled data:

$$E[T'_\tau] = T_\tau \cdot p_s p_t \Rightarrow \hat{T}_\tau = \frac{T'_\tau}{p_s p_t}. \quad \text{Similarly, } \hat{V}_\tau = \frac{V'_\tau}{p_s p_t} \tag{17}$$

Both estimators $\hat{T}_\tau$ and $\hat{V}_\tau$ are unbiased. After obtaining $\hat{T}_\tau$ and $\hat{V}_\tau$, we plug them into Equation (15) to obtain estimated cost $\hat{C}_\tau$. The

Algorithm 7: Suggesting the best τ for AU-Filter

Input: Samples $\{\mathcal{S}'_1, \cdots, \mathcal{S}'_k\}$ and $\{\mathcal{T}'_1, \cdots, \mathcal{T}'_k\}$ with small sampling probabilities p_s and p_t, a positive integer n^*, a Student's t quantile t_*, and an universe $\mathbb{U}$ of τ's
Output: τ which has the minimal estimated join time

```
1 n ← 0
2 repeat
3     n ← n + 1                                   // refine the estimation
4     foreach τ ∈ 𝕌 do
5         run the filtering stage (Lines 1-8) of Algorithm 6 on samples
            S'_n and T'_n, compute estimations T̂^(n)_τ and V̂^(n)_τ by Eq. (17)
6         compute μ̂^(n)_{T_τ}, σ̂^{2,(n)}_{T_τ}, μ̂^(n)_{V_τ}, and σ̂^{2,(n)}_{V_τ} by Eqs. (20-21)
7         compute the confidence interval CI(Ĉ^(n)_τ) by Eq. (23)
8 until n ⩾ n* and Inequality (24) holds true
9 return arg min_τ (Ĉ^(n)_τ)
```

Bernoulli estimator in Equation (17) is a static estimation strategy where sampling probability p_s and p_t are determined beforehand. Further, it is technically difficult to decide an optimal sampling probability, since a higher probability can improve the accuracy at the cost of increasing estimation time, while a lower probability can save estimation time but decrease its accuracy. Next, we propose a Monte Carlo-based solution to address this challenge.

4.2 Refinement and Parameter Suggestion

To achieve both high estimation accuracy and short estimation time, roughly speaking, our method is to generate a collection of very small samples (e.g., around 100 records in each) by using the above independent Bernoulli method. Then, we perform multiple iterations of estimation based on these samples to find the best τ with high confidence. Key tasks in this framework consist of (i) *how to aggregate the estimation mean and variance based on the results of multiple different samples* and (ii) *how to decide a termination criterion ensuring both efficient estimation and high-quality suggestion*.

Mean and variance computation. To solve the first challenge, note that multiple iterations of running different samples give a series of estimations. Since all of them are independent and identically distributed (i.i.d.), the *central limit theorem* (CLT) holds such that means of estimations of T_τ and V_τ, i.e., $\hat{\mu}_{T_\tau}$ and $\hat{\mu}_{V_\tau}$, converges to normal distributions. That is,

$$\hat{\mu}_{T_\tau} \sim \mathcal{N}\Big(E[\hat{T}_\tau], Var[\hat{T}_\tau]/n\Big) \text{ when } n \to \infty \tag{18}$$

$$\hat{\mu}_{V_\tau} \sim \mathcal{N}\Big(E[\hat{V}_\tau], Var[\hat{V}_\tau]/n\Big) \text{ when } n \to \infty \tag{19}$$

CLT also alludes that the mean and variance of underlying distribution, $E[\hat{T}_\tau]$, $Var[\hat{T}_\tau]/n$, $E[\hat{V}_\tau]$, and $Var[\hat{V}_\tau]/n$, can be estimated by the sample mean and variance, $\hat{\mu}_{T_\tau}$, $\hat{\sigma}^2_{T_\tau}$, $\hat{\mu}_{V_\tau}$, and $\hat{\sigma}^2_{V_\tau}$. All four estimators are unbiased [60] and can be calculated by using, e.g., an efficient on-line recursive formula [14, 23] below:

$$\hat{\mu}^{(n)}_{T_\tau} = \hat{\mu}^{(n-1)}_{T_\tau} + \frac{\hat{T}^{(n)}_\tau - \hat{\mu}^{(n-1)}_{T_\tau}}{n} \tag{20}$$

$$\hat{\sigma}^{2,(n)}_{T_\tau} = \frac{n-2}{n-1}\hat{\sigma}^{2,(n-1)}_{T_\tau} + n \cdot (\hat{\mu}^{(n)}_{T_\tau} - \hat{\mu}^{(n-1)}_{T_\tau})^2 \tag{21}$$

In the above equations, $\hat{T}^{(n)}_\tau$ stands for the Bernoulli estimator value from the n-th sample, i.e., by using Equation (17) where $T'^{(n)}_\tau$ is obtained directly from the n-th sample. Similarly, one can replace T by V to get formulas for $\hat{\mu}^{(n)}_{V_\tau}$ and $\hat{\sigma}^{2,(n)}_{V_\tau}$. Recall Equation (15). The mean and variance of the total cost C_τ can now be estimated as follows:

$$\hat{\mu}^{(n)}_{C_\tau} = c_f \hat{\mu}^{(n)}_{T_\tau} + c_v \hat{\mu}^{(n)}_{V_\tau} \text{ and } \hat{\sigma}^{2,(n)}_{C_\tau} = c_f^2 \hat{\sigma}^{2,(n)}_{T_\tau} + c_v^2 \hat{\sigma}^{2,(n)}_{V_\tau} \tag{22}$$

Stopping criterion. To solve the second challenge, intuitively, we can safely terminate the estimation procedure once *the penalty due to a wrong suggestion is less than the overhead of running one more estimation iteration*. More precisely, to measure the penalty in the worst-case, we compute the confidence interval (CI) for the estimation $\hat{C}_\tau^{(n)}$ of C_τ at n-th iteration as follows, where $\hat{C}_\tau^{(n)} = \hat{\mu}_{C_\tau}^{(n)}$, t_* is an appropriate *Student's t quantile* as confidence level:

$$CI(\hat{C}_\tau^{(n)}) = \left[L_{\hat{C}_\tau}^{(n)}, U_{\hat{C}_\tau}^{(n)}\right] = \left[\hat{\mu}_{C_\tau}^{(n)} - t_* \frac{\hat{\sigma}_{C_\tau}^{(n)}}{\sqrt{n}}, \hat{\mu}_{C_\tau}^{(n)} + t_* \frac{\hat{\sigma}_{C_\tau}^{(n)}}{\sqrt{n}}\right] \quad (23)$$

Let $\mathbb{U}$ be the universe of τ's, and $\tau_{\min} \in \mathbb{U}$ the optimal τ in the n-th iteration, i.e., $\tau_{\min} = \arg\min_\tau \left(\hat{C}_\tau^{(n)}\right)$. Then, we can terminate the sampling process when the following inequality holds true:

$$U_{\hat{C}_{\tau_{\min}}}^{(n)} - \min_{\forall \tau \in \mathbb{U} \setminus \tau_{\min}} L_{\hat{C}_\tau}^{(n)} < c_f \cdot \sum_{\tau \in \mathbb{U}} T_\tau'^{(n+1)} \quad (24)$$

where the left-hand-side represents the maximal penalty if the suggested $\tau_{\min}$ cannot achieve the lowest cost, while the right-hand-side is the cost of running the model for one more iteration.

Finally, we are ready to present the overall suggestion process in Algorithm 7. Given k different small independent samples form $\mathcal{S}$ and $\mathcal{T}$, it returns the best τ to minimise the join time. In particular, for each iteration, the algorithm runs the filtering stage of AU-Filter for every possible τ, obtains T'_τ and V'_τ, and estimates the mean and variance of $\hat{C}_\tau$ (Lines 5-7). The procedure terminates when the best τ is found to satisfy a predefined confidence level specified by t_*. Note that the procedure runs at least n^* iterations (Line 8) to avoid the effect of instability in the early stage, known as the *burn-in* period in Monte Carlo simulations [36].

5. EXPERIMENTAL ANALYSIS

In this section, we report an extensive experimental evaluation of proposed algorithms in this paper. Section 5.1 describes the experimental setting. Section 5.1~5.4 respectively evaluate our similarity measure, join algorithms and parameter suggestion algorithm. Section 5.5 comparing our solution with state-of-the-art alternatives.

5.1 Datasets and Experimental Settings

We employed multiple datasets to evaluate algorithms in different real-world environments. The datasets differ from each other in terms of data size, taxonomy complexity, numbers of strings, tokens, and rules. A description of each dataset and its accompanying rules can be found in Tables 6 and 7. Specifically, WIKI dataset includes Wikipedia category strings[2], while MED dataset[3] contains research papers' keywords where all keywords can be mapped to MeSH taxonomy[4]. As for knowledge sources, we employed (i) two taxonomies: *MeSH tree* and *Wikipedia categories*, containing hierarchical IS-A relations such as "Nature → Energy → Energy conversion → Hydro-power"; (ii) two synonym sources: *MeSH alternative names* and *Wikipedia Synonyms*[5] holding equivalent terms like "myocardial infarction" vs "heart attack". All algorithms were executed by JVM 8 on a Ubuntu computer with a Xeon 2.53GHz CPU and 32GB RAM. The implementation can be downloaded at `https://github.com/HY-UDBMS/AU-Join`.

5.2 Evaluation of Similarity Measures

The purpose of the first set of experiments is to evaluate the performance of our unified similarity measure and the corresponding approximation algorithm.

[2] `https://wiki.dbpedia.org/`

[3] `https://trec.nist.gov/data/t9_filtering.html`

[4] `https://www.nlm.nih.gov/mesh/`

[5] `https://en.wikipedia.org/wiki/Wikipedia:LCM`

Table 6: Characteristics of used taxonomies and synonyms.

Taxonomy (Height in min/avg/max)				Synonym	
Source	# nodes	Height	Avg. fanout	Source	# rules
MeSH tree	57,840	1 / 5.1 / 12	157	Aliases	180,259
Wiki categories	1,212,943	1 / 6.2 / 26	32,300	Synonyms	680,625

Table 7: Characteristics of used string datasets.

Source	# of strings	per string (min/avg/max)			
		Characters	Tokens	Taxonomies	Synonyms
MED	293,294	2 / 110.5 / 452	1 / 8.4 / 26	0 / 3.2 / 18	0 / 4.3 / 15
WIKI	3,512,954	2 / 161.5 / 8,588	1 / 8.2 / 277	0 / 6.2 / 185	0 / 2.0 / 98

Table 8: Effectiveness w.r.t. similarity measures. T: taxonomy; J: Jaccard; S: synonym. P: precision; R: recall; F: F-measure.

Measure	MED, θ: 0.7			MED, 0.75			WIKI, 0.7			WIKI, 0.75		
	P	R	F	P	R	F	P	R	F	P	R	F
J	0.88	0.27	0.42	0.85	0.19	0.32	0.81	0.26	0.40	0.79	0.15	0.25
T	0.89	0.12	0.20	0.86	0.09	0.17	0.83	0.08	0.15	0.83	0.05	0.10
S	0.88	0.60	0.71	0.88	0.56	0.68	0.86	0.02	0.03	0.86	0.02	0.03
TJ	0.90	0.43	0.58	0.87	0.29	0.44	0.83	0.92	0.87	0.82	0.54	0.65
TS	0.91	0.63	0.74	0.89	0.50	0.64	0.82	0.36	0.50	0.80	0.20	0.31
JS	0.89	0.77	0.83	0.88	0.60	0.71	0.84	0.11	0.20	0.84	0.07	0.14
TJS	0.86	0.96	**0.91**	0.88	0.75	**0.81**	0.83	0.98	**0.90**	0.82	0.58	**0.68**

Effectiveness. To evaluate the effectiveness of similarity measures, we applied a crowd-sourcing platform [18–20] to obtain 268 (MED dataset) and 961 (WIKI dataset) string pairs as ground truths. In MED, the inconsistencies are mainly due to synonyms (e.g., "mitral valve insufficiency" vs "bicuspid valves incompetence"), while in WIKI, we find typos and taxonomic-similar strings (e.g., "computer network, massively parallel computers" vs "computer networks, supercomputers"). We presented the experimental results in Table 8. Seven similarity measures are compared: Jaccard (J), Taxonomy (T), Synonym (S), and all their combinations. The quality-of-measures are precision (P), recall (R), and F-measure = $\frac{2PR}{P+R}$.

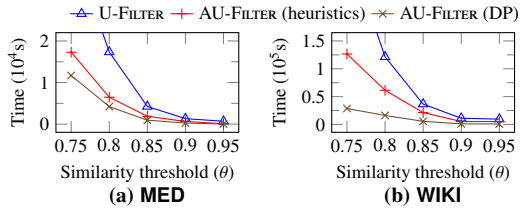
We have made several observations: (i) Single similarity measures are not capable of finding many similar strings which mix different kinds of similarities. For example, the average recall of T, J, and S is only 30% on MED and 9.6% on WIKI, resulting in low F-measure scores. (ii) The F-measure can be improved by utilising two similarity measures, since it can capture multiple similarity relations in a string pair. Moreover, different datasets favour different similarity measures. For instance, the measure that combines Jaccard and Synonym similarities (`JS`) works well for MED dataset, while WIKI favours taxonomy and Jaccard combination (`TJ`). (iii) By combining all three kinds of similarity measures (i.e. `TJS`), we achieved the best F-measure on all tested datasets, because the unified similarity measures can capture most similar pairs to achieve the best matching results.

Approximation Accuracy. This experiment is designed to evaluate the approximate ratio of Algorithm 1 for computing the unified similarity. For each dataset, we selected 3K string pairs such that the exponential-time exact algorithm completes in a reasonable short time. We calculate the approximate ratio $r = \frac{A^*}{A}$, where A^* and A denote similarity values returned by the exact and the approximate algorithm respectively. The results are presented in Table 9, where we varied the maximal partition size k for each string. We use five percentile ratios: 2nd, 25th, 50th (median), 75th, and 98th. As an example, for k=10 on MED data, the value 0.95 means that 50% approximations are having at least 0.95 accuracy.

We have the following observations: (i) The approximation ratio of Algorithm 1 is high in practice. For example, when k = 9 or 10, the accuracy of more than half string pairs is at least 0.87 and 0.9 on MED and WIKI datasets respectively. (ii) With the growth of k, the approximate ratio increases for both datasets. This is a little surprising result because the theoretical analysis in Theorem 2 shows that the worst-case bound becomes larger with the growth

Table 9: Approximation accuracy w.r.t. longest rule size k.

k	MED					WIKI				
	2%	25%	50%	75%	98%	2%	25%	50%	75%	98%
3	0.37	0.50	0.50	0.50	0.62	0.18	0.50	0.53	0.70	1.00
4	0.48	0.55	0.60	0.80	0.96	0.33	0.62	0.75	0.91	1.00
5	0.48	0.54	0.58	0.82	1.00	0.41	0.59	0.77	0.92	1.00
6	0.48	0.55	0.70	0.83	1.00	0.53	0.72	0.81	0.93	1.00
7	0.50	0.55	0.82	0.92	1.00	0.54	0.85	0.91	1.00	1.00
8	0.74	0.83	0.95	0.97	1.00	0.71	0.93	1.00	1.00	1.00
9	0.55	0.75	0.87	1.00	1.00	0.72	0.76	0.80	0.90	0.99
10	0.69	0.80	0.95	1.00	1.00	1.00	1.00	1.00	1.00	1.00



(a) MED **(b) WIKI**

Figure 4: Join time of proposed algorithms.

of k, while the empirical results manifest that approximate ratios can even improve by having a larger k. To explain this, we carefully examined many string instances and observed that the theoretical analysis emphasises only the worst-case when Algorithm 1 selects a long rule that cannot contribute to the optimal similarity. However, in practice, a longer rule is very likely to contribute to the optimal result because it has more substantial similarity contributions. In such scenarios, Algorithm 1 is more likely to return the optimal value for a larger k.

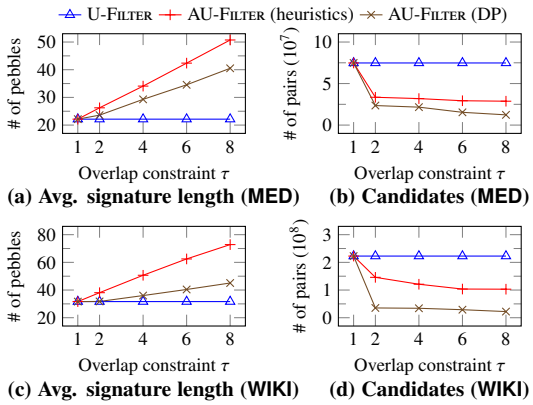
5.3 Evaluation of Similarity Join Algorithms

The second set of experiments is to evaluate the efficiency and scalability of various join algorithms. Here, we implemented all three proposed methods: U-Filter, AU-Filter with `heuristics`, and AU-Filter with `DP`. We take two measures: (i) the total running time and (ii) the size of candidates (i.e., survivors of the filtering).

Join time. Figures 4 depicts the time cost of proposed algorithms w.r.t. join thresholds. We made the following observations: (i) two AU-Filter-family algorithms, i.e. AU-Filter (`heuristics`) and AU-Filter (`DP`), outperform the baseline algorithm U-Filter. This can be explained by the fact that the AU-Filter algorithms adaptively determine the key τ parameter to enhance the filtering power thus produces fewer candidates. (ii) AU-Filter (`DP`) is the clear winner among all three methods by about 5× faster than which with `heuristics` for small join thresholds. This improvement owes to the dynamic programming that ensures short prefixes such that many false positives are being removed.

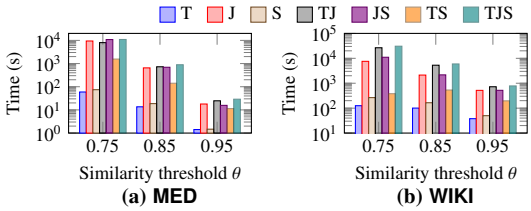
To further investigate the filtering power of each algorithm with various overlap constraints, we depicted the average length of pebble signatures and the number of candidates for different algorithms in Figure 5. It shows that, AU-Filter (`heuristics`) can filter out 50% to 60% candidate pairs, while AU-Filter (`DP`) can prune away an impressive 70% to 90% pairs on both datasets. The dramatic reduction brought by AU-Filter can undoubtedly accelerate the verification process.

To evaluate the effect of similarity measures on join time, we ran AU-Filter (`DP`) with seven different similarity measures and presented their running times in Figure 6. We found that although `TJS` is the most comprehensive function, which combines three similarity measures into one measure, its performance is still comparable to a single measure. This owes to the powerful filters in AU-filter (`DP`), which significantly reduces the number of candidates for final verification. Together with the join effectiveness verified in Section 5.2, we emphasise that our unified similarity measure can return the



(a) Avg. signature length (MED) **(b) Candidates (MED)**
(c) Avg. signature length (WIKI) **(d) Candidates (WIKI)**

Figure 5: Filtering power of various filters, $\theta = 0.85$.



(a) MED **(b) WIKI**

Figure 6: Join time of AU-Filter (DP) by similarity measures. T: taxonomy; J: Jaccard; S: synonym.

best (i.e. the most comprehensive) join results while sacrificing only a small amount of joining efficiency.

Scalability. We then assessed the effect of data sizes on join time. As seen from Figure 7, the time overhead of AU-Filter with `heuristics` and `DP` scale better than U-Filter. This is attributed to the adaptive signature selection in AU-Filter, which enables join algorithms to choose the best and fewest pebble signatures to reduce running time. By using `DP` instead of `heuristics`, the scalability of AU-Filter can be even further improved with the increase of join data size. Furthermore, we broke the join time into three pieces (the time to suggest the best parameter τ, filtering time, and verification time) in Table 10. We observed that: (i) Both filtering and verification times grows linearly instead of quadratically (considering the nature of join operations). The ability of AU-Filter (`DP`) to scale up can be credited to the effective signature selection, which removes unfeasible pairs to avoid the quadratic growth of join time. (ii) The cost to suggest the best parameters takes only less than 1% overhead in most cases, which will be discussed in details in the next section.

5.4 Evaluation of Parameter Recommendation

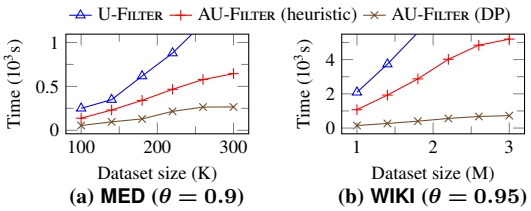
The third set of experiments is to thoroughly evaluate the parameter recommendation algorithm proposed in Section 4, and understand how it accelerates the join algorithms.

Effect on join time. To study the effect of the parameter τ on the overall join time, we implemented experiments to compare three different settings: (i) our suggested parameter, (ii) a random parameter, and (iii) the worst parameter. Table 11 compares the running time with various parameter settings for two datasets. As shown in the figure, our suggested parameter can achieve the best running time, which strongly motives for its use in practice.

Accuracy and efficiency. We then systematically evaluated the recommendation accuracy with various thresholds. We ran our suggestion algorithm for 100 times with various sampled data and recommended parameters. We also exhaustively tested all possible parameter values to obtain the optimal one. The accuracy ratios of recommended parameters with various join thresholds are shown

Table 10: Join time of AU-Filter (DP) breaks into suggestion, filtering, and verification time (seconds).

	Size (K)	100	140	180	220	260	300
MED ($\theta = 0.9$)	Suggestion	14.76	15.00	14.72	15.32	14.92	14.95
	Filtering	23.11	48.43	62.13	100.64	119.72	123.26
	Verification	31.69	47.97	67.67	115.06	144.68	142.54
	Size (M)	1.0	1.4	1.8	2.2	2.6	3.0
WIKI ($\theta = 0.95$)	Suggestion	21.53	21.21	21.28	21.79	21.46	22.05
	Filtering	63.13	135.32	192.54	262.15	306.52	337.07
	Verification	86.60	134.05	209.71	299.73	370.42	389.82



(a) MED ($\theta = 0.9$) **(b) WIKI ($\theta = 0.95$)**

Figure 7: Scalability of proposed join algorithms.

in Table 12. In most empirical cases (95% for MED and 91% for WIKI data), our algorithm can give accurate suggestions with small samples. Note that, in this experiment setting, the size of each sample is fixed to 100 strings for both datasets, which accounts for a very small fraction of the original dataset, i.e., 0.034% for MED and 0.0028% for WIKI, meaning that our algorithm can give accurate suggestions even with a very tiny amount of samples in each iteration. As shown in Table 12, the recommendation procedure occupies less than 1% join time in most cases.

To further understand the relation between the sampling probability of each iteration and the total number of iterations, we plotted Figure 8, which depicts sampling times and the number of iterations with various sampling probabilities. An interesting finding is that, running times of the algorithm do not monotonically increase or decrease with changes of the sample probability. This can be explained by the fact that a small sampling probability needs more iterations to satisfy the predefined confidence level. Therefore, there exists an optimal sampling probability for each dataset which minimises the total suggestion time. Finding such an optimum, which strikes an appropriate balance between iteration times and sampling probabilities, is an exciting direction for future research.

5.5 Comparison with State-of-the-art Alternatives

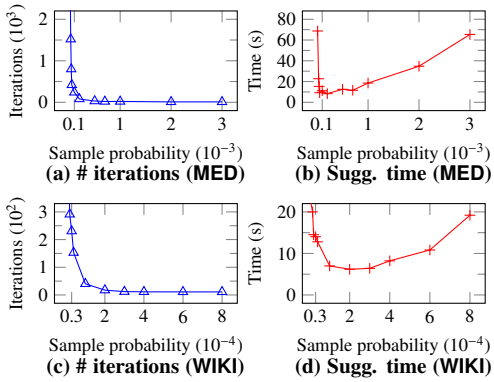
The final set of experiments is to compare our algorithm with state-of-the-art alternatives. To the best of our knowledge, no previous work can handle multiple similarity measures as ours. Therefore, we select several existing algorithms and then combine their answers to provide comprehensive results. In particular, three representative alternatives are PKduck [51] for synonym similarity, K-Join [47] for taxonomy similarity, and AdaptJoin [54] for gram-based measurement. Executable codes of AdaptJoin and PKduck are available online[6] and that of K-Join was kindly provided by the authors of [47].

Effectiveness. Table 13 compares the join effectiveness of different algorithms with varied datasets and join thresholds. Previous methods suffer from low recalls because each of them can capture similar strings with only one similarity measure. To improve the recall, we combine their outputs together as Combination. However, even with this enhancement, Table 13 shows that our method is still better than Combination, thanks to our unified framework which

[6] AdaptJoin: https://www.cs.sfu.ca/~jnwang/projects/adapt/
PKduck: https://github.com/tracyhenry/xClean

Table 11: Running time of AU-Filter (heuristics) w.r.t. different parameter selection method.

Similarity threshold (θ)		0.75	0.80	0.85	0.90	0.95
MED (10^3s)	Using suggested τ	17.30	6.47	1.88	0.64	0.09
	Mean of random τ	24.81	9.47	2.70	0.92	0.27
	Using worst τ	45.04	17.94	4.39	1.36	0.72
WIKI (10^4s)	Using suggested τ	12.66	6.13	2.12	0.51	0.51
	Mean of random τ	22.55	8.30	2.85	0.70	0.69
	Using worst τ	34.41	13.01	3.91	1.18	0.89



(a) # iterations (MED) **(b) Sugg. time (MED)**
(c) # iterations (WIKI) **(d) Sugg. time (WIKI)**

Figure 8: Parameter suggestion while $\theta = 0.8$. $n^* = 10$, $t_* = 1.036$ (70% confidence level on two sides).

can apply a mixture of different similarity measures simultaneously. Consider one example from WIKI dataset: "Anemone Flora United States" and "Anemona Flora California" which involve two kinds of similarities: "Anemona" is a wrongly spelt "Anemone" and "California" is a state of "United States". The string similarity is less than the threshold ($\theta = 0.8$) by using any individual similarity measure, e.g. Jaccard and taxonomy. In contrast, our method can seamlessly integrate two measures to recognise both types of similarities.

Join time. Finally, we compared the join efficiency of various algorithms. To make a fair comparison, we divide comparisons into four groups, such that our algorithm (i.e., AU-Filter) is compared to one or multiple existing algorithms using the same feature. For example, to compare K-Join and AU-Filter, both algorithms use the taxonomy similarity to find similar strings. Table 14 shows their average running times of ten runs. It can be seen that, our methods achieve a better performance than alternatives in most cases. This can be explained by the fact that our AU-Filter can adaptively select the key overlapping parameter τ to maximise the filtering power and thus reduce the time cost of verification (recall Section 4). Note that in the WIKI dataset, AU-Filter takes more time than other alternatives under high thresholds. Such discrepancy is due to different numbers of candidates and results: for example, when $\theta = 0.95$, Combination finds only 17 similar pairs whilst AU-Filter returns 111 (correct) results – six times more than Combination.

6. RELATED WORK

Similarity measures. Historically, there exists various similarity measures to capture syntactic differences between strings, to name a few: Levenshtein [54] and Hamming [30] distances, cosine similarity [46] and Jaccard coefficient [15, 31]. Cohen et al. [16] experimentally compared different string similarities to perform name-matching task. In recent years, there are some efforts to use semantic information to enhance the effectiveness of similarity joins, such as synonym [2, 3, 8, 32] and taxonomy [47, 57]. In particular, Arasu et al. [2] and Tao et al. [51] leveraged synonym and abbreviation to extend similarity measures. Shang et al. [47] proposed K-Join

Table 12: Suggestion accuracy and the fraction on the overall join time. Results are the average of 100 runs.

Similarity threshold (θ)		0.75	0.80	0.85	0.90	0.95
MED	Accuracy	95%	95%	99%	95%	95%
	Time fraction	0.09%	0.22%	0.71%	1.94%	1.38%
WIKI	Accuracy	93%	100%	100%	92%	91%
	Time fraction	0.02%	0.03%	0.09%	0.35%	0.34%

Table 13: Effectiveness of our measure vs existing algorithms.

Measure	MED, θ: 0.7			MED, 0.75			WIKI, 0.7			WIKI, 0.75		
	P	R	F	P	R	F	P	R	F	P	R	F
K-Join	0.89	0.12	0.20	0.86	0.09	0.17	0.83	0.08	0.15	0.83	0.05	0.10
AdaptJoin	0.81	0.19	0.30	0.79	0.15	0.25	0.71	0.28	0.40	0.64	0.15	0.24
PKduck	0.78	0.19	0.31	0.80	0.17	0.28	0.64	0.10	0.18	0.67	0.06	0.10
Combination	0.82	0.48	0.61	0.80	0.41	0.54	0.75	0.37	0.50	0.75	0.22	0.34
Ours	0.86	0.96	**0.91**	0.88	0.75	**0.81**	0.83	0.98	**0.90**	0.82	0.58	**0.68**

to introduce taxonomy knowledge in string matching. Further, K-Join$^+$ [47] adds an ad-hoc operation to match multiple taxonomy nodes through approximate match preprocessing. This paper pushes the frontier forward by proposing a new unified similarity framework to seamlessly integrates different similarity measures together.

Similarity join algorithms. There are a plethora of studies on efficient string similarity joins (e.g. [9, 15, 42, 44]). Jiang et al. [28] performed an experimental survey to compare different similarity join algorithms. The widely-adopted technique is prefix filtering. A few papers recently aim to reduce the number of candidates in the prefix filtering. For example, Xiao et al. [56] employed additional filters to enhance the filtering power, and Rong et al. [44] proposed multiple global orderings to perform filtering and join their results to reduce false positives. Wang et al. [54] provided a cost model to select an appropriate prefix for each string judiciously. However, the existing methods cannot be straightforwardly adopted in our problem because our measure requires to involve multiple different similarity measures simultaneously to render a global similarity.

Machine learning for string similarity. Machine learning-based methods aim to support string similarity matching by learning a group of similar string as examples. In particular, Islam et al. [27] determines the similarity of two pieces of text from semantic and syntactic information through supervised and unsupervised learning. Papers [35, 45] parameterise different measures using machine learning techniques to choose an appropriate weight for each measure. Further, there are approaches aiming to learn a similarity model through logistic regression [52], SVM [12], or vector embedding (e.g. word2vec [37] and GloVe [41]). This paper differentiates from above previous works by addressing the important issue of efficiency for similarity joins on large size of records.

7. CONCLUSION

We have studied a problem of integrating multiple similarity measures for improving the quality of string similarity joins. We proposed a unified similarity measure as well as its efficient computation algorithm. To perform joins efficiently, we have developed several pebble-based filtering strategy which judiciously selecting signature size and type. Experiments based on real datasets exhibit the superiority of proposed algorithms for both effectiveness and efficiency.

8. APPENDIX

Proof (Sketch) of Theorem 1. We now show a reduction from *maximum independent set* (MIS) [40] to our problem USIM that operates in polynomial time. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of MIS, we can build an instance of USIM as follows. For each vertex $u_i \in \mathcal{V}$, we create two tokens m_i, n_i and for each edge $e_i \in \mathcal{E}$, we create another two tokens p_i, q_i. Then let $S = (\bigcup_{i=1}^{|\mathcal{V}|} m_i) \cup (\bigcup_{j=1}^{|\mathcal{E}|} p_j)$ and $T = (\bigcup_{i=1}^{|\mathcal{V}|} n_i) \cup (\bigcup_{j=1}^{|\mathcal{E}|} q_j)$. To construct synonym rules, for each vertex $u_i \in \mathcal{V}$, we generate one rule $R_i : \{m_i\} \cup \{p_j | \exists e_j \text{ connects } u_i\} \to n_i$ with a weight $C(R_i) = 1$. For example, consider a simple graph with two nodes v_1 and v_2 and one edge between them. Then we can construct $S = \{m_1, m_2, p_1\}$ and $T = \{n_1, n_2, q_1\}$ and two synonym rules: $R_1 : \{m_1, p_1\} \to n_1$ and $R_2 : \{m_2, p_1\} \to n_2$. Our goal is to prove that finding a maximum independent set for MIS is equivalent to compute a unified similarity on USIM .

First, we observe that given arbitrary partitions $\mathcal{P}_S$ and $\mathcal{P}_T$ in USIM, $\max\{|\mathcal{P}_S|, |\mathcal{P}_T|\} = |\mathcal{V}| + |\mathcal{E}|$, because (i) $|\mathcal{P}_T| = |\mathcal{V}| + |\mathcal{E}|$ regardless how many rules are selected, and (ii) $|\mathcal{P}_S| \leqslant |\mathcal{V}| + |\mathcal{E}|$. Therefore, the denominator of Equation (6) is fixed. Hence, computing the maximal numerator of Equation (6) is equivalent to finding the maximum independent set in $\mathcal{G}$, because each rule corresponds one vertex and we cannot select any two connected vertices (rules) simultaneously.

Proof (Sketch) of Theorem 2. Suppose that I^* is the optimal solution. Let $n(I)$ and $d(I)$ denote the nominator and denominator of Equation (6) for a solution I. By the definition of approximate ratio,

$$\frac{USIM(I^*)}{USIM(I)} = \frac{n(I^*)/d(I^*)}{n(I)/d(I)} = \frac{n(I^*)}{n(I)} \cdot \frac{d(I)}{d(I^*)} \tag{25}$$

Since SquareImp [11] can provide the guarantee on the approximation quality of w-MIS problem, we have

$$\frac{n(I^*)}{n(I)} \leqslant \frac{t}{t-1} \cdot \frac{k+1}{2} \tag{26}$$

Furthermore, since any synonym rule or taxonomy pair contains at most k-1 tokens in one-side, we have

$$\frac{d(I)}{d(I^*)} \leqslant k - 1 \tag{27}$$

By applying Inequalities (26) and (27) with Equation (25), we can establish the approximation ratio, as desired. Further, we construct an instance to show that the bound of $d(I)$, i.e. Equation (27), is indeed tight. Let m, n, p, and q denote the set of tokens of cardinalities $k-1$, 1, $(k-1)^2$, and $k-1$. Define a string S containing k tokens: $\{m_1, \cdots, m_{k-1}, q_1\}$, and T containing $k(k$-1) tokens: $\{n_1, p_1, \cdots, p_{(k-1)^2}, q_2, \cdots, q_{k-1}\}$. We then construct $k+1$ synonym rules. The first $k-1$ rules are: $R_i = \{m_i\} \to \{p_{(k-1)i-(k-2)}, \cdots, p_{(k-1)i}\}$ ($i \in [1, k-1]$), the k-th rule: $R_k = \{q_1\} \to \{n_1, q_2, \cdots, q_{k-1}\}$, and the final rule: $R_{k+1} = \{m_1, \cdots, m_{k-1}\} \to \{n_1\}$. We can assign the weights for all rules such that $C(R_{k+1}) < \sum_{i=1}^{k} C(R_i)$, $C^2(R_{k+1}) > \sum_{i=1}^{k} C^2(R_i)$, and $\frac{1}{t} < \frac{\sum_{i=1}^{k} C(R_i)}{k} - \frac{C(R_{k+1})}{k(k-1)}$ given an arbitrary value t. Note that Algorithm 1 uses SquareImp to select R_{k+1} in Line 1, and the heuristics in Lines 3-4 cannot improve this result. Thus, this ends up with the solution $d(I) = k(k-1)$. In contrast, the optimal solution I^* can select $R_1, \cdots, R_k$ to achieve $d(I^*) = k$. This yields $d(I)/d(I^*) = k - 1$.

Finally, to analyse the time complexity, note that SquareImp runs in polynomial of $t \cdot n$ [11]. Further, Lines 3-4 of Algorithm 1 run for at most $\lfloor t \rfloor$ iterations, and each iteration checks claws whose number is polynomial in n. Therefore, the overall time complexity is still polynomial in $t \cdot n$.

Table 14: Join time (s) of our algorithm vs existing methods.

Method	MED (100K)					WIKI (100K)				
	θ: 0.75	0.8	0.85	0.9	0.95	0.75	0.8	0.85	0.9	0.95
K-Join	2.8	2.7	2.2	2.1	1.8	5.4	5.0	3.1	3.0	2.7
Ours (T)	2.6	2.6	2.2	2.1	1.8	4.5	4.3	2.9	2.9	2.7
AdaptJoin	1045.8	675.4	270.6	48.3	10.5	1360.2	1044.6	480.6	120.0	16.9
Ours (J)	597.9	217.0	85.5	20.9	10.3	1301.3	644.5	274.2	62.6	11.3
PKduck	51.6	30.6	15.1	8.3	7.4	39.5	17.3	8.6	3.5	1.4
Ours (S)	20.8	18.0	14.7	7.0	6.8	15.8	11.1	8.0	4.3	3.2
Combination	1100.2	708.7	287.9	58.7	19.1	1405.1	1066.9	492.3	126.5	21.0
Ours (TJS)	842.1	413.8	253.7	54.7	18.9	1418.1	694.7	308.9	113.5	22.4

9. REFERENCES

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VDLB*, pages 586–597, 2002.

[2] A. Arasu, S. Chaudhuri, and R. Kaushik. Transformation-based framework for record matching. In *ICDE*, pages 40–49, 2008.

[3] A. Arasu, S. Chaudhuri, and R. Kaushik. Learning string transformations from examples. In *PVLDB 2(1)*, pages 514–525, 2009.

[4] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, pages 918–929, 2006.

[5] E. M. Arkin and R. Hassin. On local search for weighted k-set packing. *Math. Oper. Res.*, 23(3):640–648, 1998.

[6] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In *ISWC*, pages 722–735, 2007.

[7] V. Bafna, B. O. Narayanan, and R. Ravi. Nonoverlapping local alignments (weighted independent sets of axis-parallel rectangles). *Discrete Applied Mathematics*, 71(1-3):41–53, 1996.

[8] J. Barbay, A. Golynski, J. I. Munro, and S. S. Rao. Adaptive searching in succinctly encoded binary relations and tree-structured documents. In *CPM*, pages 24–35, 2006.

[9] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, pages 131–140. ACM, 2007.

[10] A. Behm, S. Ji, C. Li, and J. Lu. Space-constrained gram-based indexing for efficient approximate string search. In *ICDE*, pages 604–615, 2009.

[11] P. Berman. A $d/2$ approximation for maximum weight independent set in d-claw free graphs. In *SWAT*, volume 1851 of *Lecture Notes in Computer Science*, pages 214–219. Springer, 2000.

[12] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 39–48, 2003.

[13] B. Chandra and M. M. Halldórsson. Greedy local improvement and weighted set packing approximation. *J. Algorithms*, 39(2):223–240, 2001.

[14] C. Chatfied and A. J. Collins. *Introduction to multivariate analysis*. Springer, 2013.

[15] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, page 5, 2006.

[16] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78, 2003.

[17] C. Dela Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[18] Z. Dong, J. Fan, Lu, X. Du, and T. W. Ling. Using crowdsourcing for fine-grained entity type completion in knowledge bases. In *APWeb/WAIM (2)*, volume 10988 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2018.

[19] Z. Dong, J. Lu, and T. W. Ling. PANDA: A platform for academic knowledge discovery and acquisition. In *BigComp*, pages 10–17, 2016.

[20] Z. Dong, J. Lu, T. W. Ling, J. Fan, and Y. Chen. Using hybrid algorithmic-crowdsourcing methods for academic knowledge acquisition. *Cluster Computing*, 20(4):3629–3641, 2017.

[21] M. Färber, F. Bartscherer, C. Menne, and A. Rettinger. Linked data quality of dbpedia, freebase, opencyc, wikidata, and YAGO. *Semantic Web*, 9(1):77–129, 2018.

[22] E. Ferrara, P. D. Meo, G. Fiumara, and R. Baumgartner. Web data extraction, applications and techniques: A survey. *Knowl.-Based Syst.*, 70:301–323, 2014.

[23] T. Finch. Incremental calculation of weighted mean and variance. *University of Cambridge*, 4:11–5, 2009.

[24] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete problems. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 47–63. ACM, 1974.

[25] M. M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *Algorithmica*, 18(1):145–163, 1997.

[26] C. A. J. Hurkens and A. Schrijver. On the size of systems of sets every t of which have an sdr, with an application to the worst-case ratio of heuristics for packing problems. *SIAM J. Discrete Math.*, 2(1):68–72, 1989.

[27] A. Islam and D. Z. Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *TKDD*, 2(2):10:1–10:25, 2008.

[28] Y. Jiang, G. Li, J. Feng, and W. Li. String similarity joins: An experimental evaluation. *PVLDB*, 7(8):625–636, 2014.

[29] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

[30] G. Kondrak. N-gram similarity and distance. In *SPIRE*, pages 115–126, 2005.

[31] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, 2008.

[32] J. Lu, C. Lin, W. Wang, C. Li, and H. Wang. String similarity measures and joins with synonyms. In *SIGMOD*, pages 373–384, 2013.

[33] J. Lu, C. Lin, W. Wang, C. Li, and X. Xiao. Boosting the quality of approximate string matching by synonyms. *ACM Trans. Database Syst.*, 40(3):15:1–15:42, 2015.

[34] Y. Lu, J. Lu, G. Cong, W. Wu, and C. Shahabi. Efficient algorithms and cost models for reverse spatial-keyword k-nearest neighbor search. *ACM Trans. Database Syst.*, 39(2):13:1–13:46, May 2014.

[35] P. Malakasiotis. Paraphrase recognition using machine learning to combine similarity measures. In *ACL/IJCNLP (Student Research Workshop)*, pages 27–35. The Association for Computer Linguistics, 2009.

[36] S. P. Meyn and R. L. Tweedie. *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

[37] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *ICLR (Workshop)*, 2013.

[38] C. Moore and S. Mertens. *The nature of computation*. OUP Oxford, 2011.

[39] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[40] T. Oosterwijk. On local search and LP and SDP relaxations for k-set packing. *CoRR*, abs/1507.07459, 2015.

[41] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543. ACL, 2014.

[42] J. Qin and C. Xiao. Pigeonring: A principle for faster thresholded similarity search. *PVLDB*, 12(1):28–42, 2018.

[43] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[44] C. Rong, W. Lu, X. Wang, X. Du, Y. Chen, and A. K. H. Tung. Efficient and scalable processing of string similarity join. *IEEE Trans. Knowl. Data Eng.*, 25(10):2217–2230, 2013.

[45] T. Saikh, S. K. Naskar, C. Giri, and S. Bandyopadhyay. Textual entailment using different similarity metrics. In *CICLing (1)*, volume 9041 of *Lecture Notes in Computer Science*, pages 491–501. Springer, 2015.

[46] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, 1988.

[47] Z. Shang, Y. Liu, G. Li, and J. Feng. K-join: Knowledge-aware similarity join. *IEEE Trans. Knowl. Data Eng.*, 28(12):3293–3308, 2016.

[48] K. Slabbekoorn, L. Hollink, and G. Houben. Domain-aware ontology matching. In *The Semantic Web ISWC*, pages 542–558, 2012.

[49] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[50] T. P. Tanon, D. Vrandecic, S. Schaffert, T. Steiner, and L. Pintscher. From freebase to wikidata: The great migration. In *WWW*, pages 1419–1428, 2016.

[51] W. Tao, D. Deng, and M. Stonebraker. Approximate string joins with abbreviations. *PVLDB*, 11(1):53–65, 2017.

[52] Y. Tsuruoka, J. McNaught, J. Tsujii, and S. Ananiadou. Learning string similarity measures for gene/protein name dictionary look-up using logistic regression. *Bioinformatics*, 23(20):2768–2774, 2007.

[53] D. Vengerov, A. C. Menck, M. Zaït, and S. Chakkappen. Join size estimation subject to filter conditions. *PVLDB*, 8(12):1530–1541, 2015.

[54] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *SIGMOD Conference*, pages 85–96, 2012.

[55] Z. Wu and M. S. Palmer. Verb semantics and lexical selection. In *ACL*, pages 133–138. Morgan Kaufmann Publishers / ACL, 1994.

[56] C. Xiao, W. Wang, X. Lin, and J. X. Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140. ACM, 2008.

[57] P. Xu and Lu. Efficient taxonomic similarity joins with adaptive overlap constraint. In *ACM CIKM*, pages 1563–1566, 2018.

[58] P. Xu and J. Lu. Top-k string auto-completion with synonyms. In *DASFAA*, pages 202–218, 2017.

[59] X. Zhang, J. Qin, W. Wang, Y. Sun, and J. Lu. Hmsearch: an efficient hamming distance query processing algorithm. In *SSDBM*, pages 19:1–19:12, 2013.

[60] D. Zwillinger. *CRC standard mathematical tables and formulae*. CRC press, 2002.