1 Preliminaries.

1.1 Fuzzy Set Similarity Join.

Our inputs for Fuzzy set similarity join problem are two collections of sets ($\mathcal{R}$ and $\mathcal{S}$), a set similarity function $sim_\phi(R, S)$ (where $R \in \mathcal{R}$ and $S \in \mathcal{S}$), and a user-defined threshold δ. Our goal is to compute all pairs of similar sets, which is defined as: $\mathcal{R} \bowtie_\delta \mathcal{S} = \{(R, S) \in \mathcal{R} \times \mathcal{S} \mid sim_\phi(R, S) \geq \delta\}$ [4, 13, 17]. A set $R \in \mathcal{R}$ contains elements $r \in R$, where each element can be composed of a set of tokens $t \in r$ that are designated as q-grams in string tokenization. In this work, we focus on the *fuzzy set similarity problem* to approximately match set elements to each other, rather than the traditional definition that focuses on exact set overlap to compute $sim_\phi(R, S)$. We describe this process in detail as follows, but also provide a visual example in Fig. 1.

To approximately relate two sets R and S based on a threshold δ, we can formulate a bipartite graph $G(V, E, w)$, $V = R \cup S$, between the elements of R and S with edges weighted by a chosen similarity measure $\phi(r, s) \in [0, 1]$. There exist many measures for set similarity, but we focus on Jaccard similarity (JAC) and normalized edit similarity (NEDS), which are defined as follows relative to two elements r, s:

$$JAC(r, s) = \frac{|r \cap s|}{|r \cup s|}, \; NEDS(r, s) = 1 - \frac{LD(r, s)}{\max(|r|, |s|)}.$$

For NEDS, LD is the Levenshtein distance [9], which is the number of *edit* operations (delete, insert, substitute) to transform one string into the other.

After constructing the bipartite graph G, we find a maximum weight bipartite matching M on G to compute a similarity score between the pair (R, S) (defined in §1.2). The total weight of the matching, $|R \widetilde{\cap}_\phi S|$, is incorporated into set similarity between R and S as follows:

$$sim_\phi(R, S) = \frac{|R \widetilde{\cap}_\phi S|}{|R| + |S| - |R \widetilde{\cap}_\phi S|}.$$

Using this notion of fuzzy set similarity, we can define the problem of fuzzy set similarity join as follows:

Problem 1. *(Threshold-Based Fuzzy Set Similarity Join) Given two collections of sets $\mathcal{R}$ and $\mathcal{S}$, a similarity function $sim_\phi(R, S)$, and a similarity threshold $\delta \in [0, 1]$, find all pairs of sets: $\mathcal{R} \bowtie_\delta \mathcal{S} = \{(R, S) \in \mathcal{R} \times \mathcal{S} \mid sim_\phi(R, S) \geq \delta\}$.*

Notation	Description
$\mathcal{R}, \mathcal{S}$	Collections of sets
$R \in \mathcal{R}, S \in \mathcal{S}$	Sets within a collection
$r \in R, s \in S$	Elements within a set

Table 1: Key notations used in the paper.

1.2 Bipartite Weighted Matching.

A bipartite graph $G(V, E, w)$ is defined on a vertex set $V = R \cup S$, where R and S are the two disjoint parts. The edge set E consists of sets $\{u, v\}$, where $u \in R$ and $v \in S$. $w : E \rightarrow R_{\geq 0}$ is a non-negative weight function defined on the edges. Throughout the paper, we denote $n \coloneqq |V|$, and $m \coloneqq |E|$. For a vertex v, let $\delta(v)$ be the set of edges incident on v.

A *matching* M in G is a subset of E, where for each edge in M is vertex disjoint, i.e, $e_i \cap e_j = \emptyset$, where $i \neq j$ and $e_i, e_j \in M$. A *perfect matching* is a matching that covers all the vertices. Formally, if M is perfect then $\bigcup\{e \in M\} = V$. Let $w(M)$ be the sum of weights of the edges in the matching, i.e., $w(M) = \sum_{e \in M} w(e)$. The maximum weight bipartite matching (MWBM) is to find a matching M_*, whose $w(M_*)$ is the maximum among all possible matchings of G. Note that the graphs generated from similarity joins are complete bipartite graphs, and the resultant maximum weighted matching is always perfect. Such problems are also referred to as the assignment problem in the literature. For a constant $0 < \alpha < 1$, an *approximate weighted matching* M_a is a matching whose weight is at least α fraction of maximum weight, i.e., $w(M_a) \geq \alpha \cdot w(M_*)$.

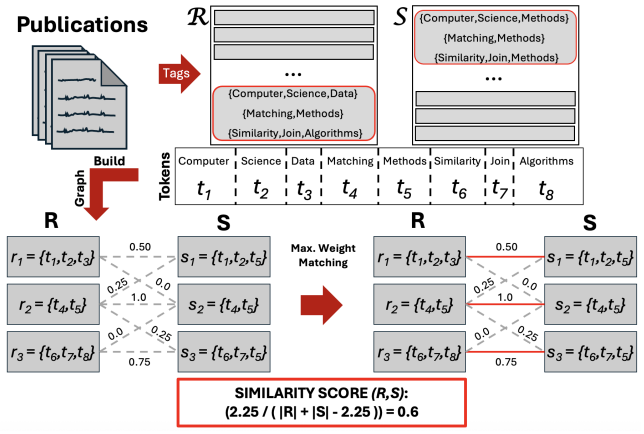

Figure 1: Illustration depicting the usage of matching within the fuzzy set similarity join workflow. Text data, such as publication tags, are split into tokens and further into a bipartite graph with similarity-based weights. The maximum weight matching is incorporated into the resultant fuzzy similarity score between the sets R and S.

2 Related Work.

2.1 Traditional Set Similarity:

Traditional and exact set similarity works have been studied as a fundamental problem in data processing [5–7, 10, 11, 15], often utilizing a method based on candidate pair generation and verification through a filter-verify framework. These methods utilize similarity functions (e.g., Jaccard, cosine, or overlap coefficient[8]) to determine set overlap. Typical advancements rely on improving filtering

techniques that consider tradeoffs between efficiency and effectiveness [3, 12] and targeting performance improvements in large-scale applications. Filtering methods commonly studied include *prefix filtering* [2], *size filtering* [1] and *positional filtering* [16], which have all seen respective modern improvements. Traditional set similarity methods are not robust to perturbations within data, often leading to imprecise similarity values when data is slightly ambiguous. These cases are better considered in the *fuzzy set similarity* to approximately match set elements.

2.2 Fuzzy Set Similarity:

The fuzzy set similarity join problems are based on bipartite-matching that approximates the set relatedness. Matching algorithms are computationally expensive, which increases the verification time, thus encouraging advanced filtering methods to outweigh the slower runtimes in the filter-verify framework. We discuss some of the previous methods that extend this motivation, while focusing on optimizations within verification.

The early work includes the FastJoin method [14], which proposes a fuzzy-token similarity function to address the fuzzy set similarity problem. This method works by assigning *signatures* to token sets generated as q-grams, identifying sets containing those signature tokens and performing the bipartite matching to compute fuzzy overlap. They also introduce the concept of a threshold for the matching weight relative to set similarity and apply the filter-verify framework in the context of fuzzy set similarity.

SilkMoth [4] checks for approximate equivalence of sets through set similarity and further checks for subset properties as set containment. Like FastJoin, SilkMoth follows a signature-based scheme but eliminates false-negatives through filtering to better curate their candidate set. They further improve verification by considering the triangle inequality of mapping the similarity function to a distance function $\psi(r, s) = 1 - \phi(r, s)$. Thus, if the triangle inequality is satisfied by ψ, then the identical elements of R and S must be in the maximum matching. This reduces the number of vertices in the bipartite graph, as they can be immediately inserted into the matching beforehand. SilkMoth showed 13× performance relative to FastJoin with a 30-50% improvement in verification times using the optimization.

The TokenJoin method [17] improves SilkMoth and the state-of-the-art for fuzzy similarity joins. TokenJoin relies on a token-based filtering method (rather than the element-based filtering of SilkMoth), which assigns utility values to tokens and optimizes filtering relative to these utilities. For this, they include a *positional filter* and a *joint utility filter*, utilizing token positions in prefixing and utilities between the set pairing (R, S) for improved refinement, respectively. The TokenJoin method also introduces an *Efficient Verification* technique that can terminate the Hungarian algorithm early based on current upper and lower bound of the matching and the given threshold.Experiments show up to a 4.5× performance improvement of Efficient Verification on large dataset instances with element counts greater than 100. Results demonstrate that TokenJoin is an order of magnitude faster than SilkMoth.

Appendix

3 Details on Threshold Join.

The threshold based Fuzzy Join workflow can be divided into several components which we describe in this section. Our discussion is primarily based on the TokenJoin method [17]. Given a constant $\delta \in [0, 1]$ that represent the admissibility of a pair into the join, we can convert an equivalent constant using the matching score $|R \widetilde{\cap}_{\phi} S|$ for a query set R and a candidate set S.

3.1 Candidate Generation:

For candidate generation, an inverted list I is built using $\mathcal{D}$ to map each token to the sets that contain that token, sorted in increasing order by size. Candidates are pulled from I based on a size filter $|R|/\delta$. For each candidate S, if S is already marked as a candidate, its utility score is increased; otherwise, S is added to the set of candidates with its utility score initialized. The sum of possible utility scores is kept as σ and is decremented for each candidate pairing's utility score added. Candidate generation stops when σ falls below the threshold $\theta = \frac{2 \cdot \delta}{1+\delta}$.

3.2 Candidate Refinement:

Following generation, refinement seeks to apply the matching threshold $\theta_{RS} = \frac{\delta}{1+\delta}(|R| + |S|)$ to each candidate S. Based on the utility score set in candidate generation, S can be directly pruned if its utility score and σ fall below θ_{RS}. This upper bound is then repeatedly refined per token, updating σ and checking if a token $t \in S$. If so, its utility score is increased; otherwise, the upper bound is check and S is pruned if its upper bound is below θ_{RS}.

3.3 Additional Filtering:

The authors of [17] extend the refinement method with both a *Positional Filter* and a *Joint Utility Filter*, extending the TokenJoin method to TJPJ. First, the positional filter relies on the idea that the token's position can be used in the inverted index I to tighten the upper bound on a candidate. Thus, I is adapted to store the position where the token was found as well as the token itself. The Joint Utility filter generalizes candidate pair utility values, that are normally set to a candidate S, to a pairing (R, S), restricting the resultant matching by $l = \min(|R|, |S|)$ edges in the bipartite graph. Thus, this gives a better bound on the utility values, as the top l values can be picked. These additions are included in the candidate refinement for better consideration of both the elements in R and in S.

References

[1] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. 2006. Efficient Exact Set-Similarity Joins. *VLDB*, 918–929.

[2] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. 2007. Scaling up all pairs similarity search. In *Proceedings of the 16th International Conference on World Wide Web* (Banff, Alberta, Canada) *(WWW '07)*. Association for Computing Machinery, New York, NY, USA, 131–140. https://doi.org/10.1145/1242572.1242591

[3] Edans F. de O. Sandes, George Teodoro, and Alba C. M. A. Melo. 2017. Bitmap Filter: Speeding up Exact Set Similarity Joins with Bitwise Operations. *CoRR* abs/1711.07295 (2017). arXiv:1711.07295 http://arxiv.org/abs/1711.07295

[4] Dong Deng, Albert Kim, Samuel Madden, and Michael Stonebraker. 2017. SilkMoth: an efficient method for finding related sets with maximum matching constraints. *Proc. VLDB Endow.* 10, 10 (June 2017), 1082–1093. https://doi.org/10.14778/3115404.3115413

[5] Fabian Fier, Nikolaus Augsten, Panagiotis Bouros, Ulf Leser, and Johann-Christoph Freytag. 2018. Set similarity joins on mapreduce: an experimental survey. *Proceedings of the VLDB Endowment* 11 (06 2018), 1110–1122. https://doi.org/10.14778/3231751.3231760

[6] Fabian Fier, Nikolaus Augsten, Panagiotis Bouros, Ulf Leser, and Johann-Christoph Freytag. 2018. Set similarity joins on mapreduce: An experimental survey. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1110–1122.

[7] Yu Jiang, Guoliang Li, Jianhua Feng, and Wen-Syan Li. 2014. String similarity joins: An experimental evaluation. *Proceedings of the VLDB Endowment* 7, 8 (2014), 625–636.

[8] Vijaymeena M K and Kavitha K. 2016. A Survey on Similarity Measures in Text Mining. *Machine Learning and Applications: An International Journal* 3 (03 2016), 19–28. https://doi.org/10.5121/mlaij.2016.3103

[9] Gonzalo Navarro. 2001. A guided tour to approximate string matching. *ACM Comput. Surv.* 33, 1 (March 2001), 31–88. https://doi.org/10.1145/375360.375365

[10] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2019. A Survey of Blocking and Filtering Techniques for Entity Resolution. *CoRR* abs/1905.06167 (2019). arXiv:1905.06167 http://arxiv.org/abs/1905.06167

[11] Leonardo Ribeiro, Felipe Ferreira Borges, and Diego Junior do Carmo Oliveira. 2021. Efficient Set Similarity Join on Multi-Attribute Data Using Lightweight Filters. *J. Inf. Data Manag.* 12 (2021). https://api.semanticscholar.org/CorpusID:238997860

[12] Leonardo Andrade Ribeiro and Theo Härder. 2011. Generalizing prefix filtering to improve set similarity joins. *Inf. Syst.* 36, 1 (March 2011), 62–78. https://doi.org/10.1016/j.is.2010.07.003

[13] Jiannan Wang, Guoliang Li, and Jianhua Feng. 2014. Extending String Similarity Join to Tolerant Fuzzy Token Matching. *ACM Transactions on Database Systems (TODS)* 39 (01 2014). https://doi.org/10.1145/2535628

[14] Jiannan Wang, Guoliang Li, and Jianhua Feng. 2014. Extending String Similarity Join to Tolerant Fuzzy Token Matching. *ACM Transactions on Database Systems (TODS)* 39 (01 2014). https://doi.org/10.1145/2535628

[15] Xubo Wang, Lu Qin, Xuemin Lin, Ying Zhang, and Lijun Chang. 2017. Leveraging set relations in exact set similarity join. *Proc. VLDB Endow.* 10, 9 (May 2017), 925–936. https://doi.org/10.14778/3099622.3099624

[16] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. 2011. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.* 36, 3, Article 15 (Aug. 2011), 41 pages. https://doi.org/10.1145/2000824.2000825

[17] Alexandros Zeakis, Dimitrios Skoutas, Dimitris Sacharidis, Odysseas Papapetrou, and Manolis Koubarakis. 2022. TokenJoin: Efficient Filtering for Set Similarity Join with Maximum Weighted Bipartite Matching. *Proc. VLDB Endow.* 16, 4 (Dec. 2022), 790–802. https://doi.org/10.14778/3574245.3574263