1 Methodology.

1.1 Set Verification

We outline the standard verification phase in a matching-based fuzzy set similarity join in Algorithm 2. The goal of verification is to score two candidate sets, R and S, based on approximate similarity. First, the algorithm proceeds with a deduplication procedure, which eliminates the duplicate elements between R and S. In practice, each duplicate element yields a similarity measure of 1 to itself, which is incorporated into the overall similarity through the overlap calculation. Following deduplication, if the size of R_d is zero (we guarantee prior to verification that the smallest set of the pairing is labeled as R), then there is no matching to be done, and we output the scoring based on the overlaps. Otherwise, we proceed to build the graph, where we assign a similarity score as edge weight between each element (the vertices) in the R_d and S_d records. We also maintain an upper bound (UB) on maximum weighted matching through the maximum score generated which allow us to skip matching if the UB is smaller than the threshold δ (as shown in [20]). Finally, we compute the matching (if not skipped) on the graph and include the number of overlaps to generate a score of overall similarity between R and S.

1.2 Matching Algorithms

We formally define the matching problem and provide a brief literature review in §2. In this section, we detail the matching algorithms in the context of set similarity join. The state-of-the-art bipartite matching based fuzzy set join methods, such as [3, 20], employ a primal-dual based method [8] commonly referred to as the Hungarian (HG) algorithm. The HG algorithm is an iterative one, where each iteration requires $O(n^2)$ time with at most n iterations giving the total $O(n^3)$ runtime. In this paper, we propose incorporating efficient approximate matching into the verification phase of the Fuzzy set similarity workflow. We choose three representative approximate maximum weight matching methods: Greedy (GD), Locally Dominant (LD) and the semi-streaming algorithm of the Paz and Schwartzman (PS). We briefly describe these matching algorithms alongside optimizations we made within the implementations.

1.2.1 ***Hungarian (HG) Method:*** The Hungarian (Kuhn-Munkres) [13] is a primal-dual algorithm for solving MWBM problem optimally. It starts with a set of feasible dual variables of the primal and dual formulations in Fig. 1. The primal solution x (i.e., the matching M) is initialized with an empty set. We find a trivial feasible dual solution, which sets $y(u)$ for $u \in R$ as the maximum weight of its incident edges, and sets $y(v) = 0$, for $v \in S$. Note that, we start with an infeasible primal but feasible dual solution. The goal is to go towards primal feasibility while maintaining the dual feasible solutions. To facilitate that, we maintain an equality graph $(G_\ell(V_\ell, E_\ell)$, which is a subgraph of G containing the tight edges in terms of dual variables (i.e., the edges e that satisfy $y(u) + y(v) = w(e)$). The HG algorithm runs in phases, where in each phase we have two possibilities: i) The matching (primal) is increased by one edge through finding an augmenting path in G_ℓ, or ii) the equality subgraph is expanded to $G_{\ell'}$, where $E_\ell \subset E_{\ell'}$. The algorithm ends when M is a perfect matching (without loss of generality, we assume that the matching here is maximum-weight perfect matching). Note that, during the algorithm we always maintain $w(M) = \sum_{v \in V} y(v)$. So, from Lemma 2.1, when M is a perfect matching (i.e., a primal feasible solution) the strong duality holds and we conclude that M is also a maximum weight matching. It can be shown that there are at most n phases and each phase can be implemented in $O(n^2)$ time resulting in the $O(n^3)$ overall runtime. We use the HG implementation provided in NetworkX [10] for empirical evaluations.

1.2.2 ***Efficient Verification (EV) Method:*** The authors of TokenJoin [20] present an *Efficient Verification* method based on the primal-dual basis of the Hungarian method. The EV method tailors the Hungarian method to the application, beginning with an empty matching and calculating a tentative verification score based on the current state of the matching. An upper bound for the score is generated by matching every unmatched vertex to its highest weight neighbor, allowing overlaps in the matching. A lower bound is similarly calculated by greedily matching all unmatched vertices. If the upper bound falls short of the threshold, then the candidate set pairing can be discarded. If the lower bound surpasses the threshold, then the candidate set pairing can be committed to the resultant set. This provides an early-termination condition for the traditional Hungarian approach. We specifically use the upper bound and lower bound (UBLB) variation, calculating both bounds and terminating the matching early if one of the two conditions are satisfied. We use the implementation provided by [20].

1.2.3 ***Greedy (GD) Method:*** The 1/2-approximate greedy matching algorithm [2] starts with an empty matching and follows a simple process: sort the edges of the graph in descending order by weight and add each non-conflicting edge into the matching in that order. In implementation, we designate a sorted list of edge tuples and simply iterate over the edges, checking if either of the endpoints are already matched. Finally, we return the weight of matching by summing the matching edge weights.

1.2.4 ***Locally Dominant (LD) Method:*** We describe the 1/2-approximate Locally Dominant (Pointer Chasing) method from [18] in Algorithm 3. This method focuses on finding locally dominant edges among neighborhood edges, where the algorithm proceeds in two phases: *pointing* and *matching*. In the pointing phase, each vertex scans through its neighborhood and sets a pointer to its highest weighted, unmatched neighbor (ties are broken consistently using vertex ids). In the matching phase, all mutually-pointing vertices are committed to the matching, and all non-mutual pointers are reset for the following iteration. In implementation, we build our graph from adjacency lists for neighborhood traversal each iteration. We substitute HG with LD in Line 16 of Algorithm 2.

1.2.5 ***Semi-Streaming Paz and Schwartzman (PS) Method:*** Now, we turn to a streaming matching algorithm for bipartite weighted matching. In a (semi-) streaming matching setting, the edges are streamed one by one and the algorithm is required to compute the matching at the end of stream without storing the full graph into memory. Thus, the memory usage of this algorithm is

Algorithm 1 Verification with PS Matching

Input: Candidate Sets: R, S, Threshold θ_R, constant ϵ
Output: Score: $sim_\phi(R, S)$

1: R_d, S_d, ov = `deduplication`(R, S)
2: UB = $|R|$; $M \leftarrow \emptyset$; Stk $\leftarrow \emptyset$; $\forall v \in V : \phi(v) = 0$
3: **for all** $r \in R_d$ **do**
4: $\quad max_s = 0$
5: $\quad$ **for all** $s \in S_d$ **do**
6: $\quad\quad w(e) =$ `sim`(r, s) ▷ $e = \{r, s\}$
7: $\quad\quad max_s =$ `max`$(max_s, w(e))$
8: $\quad\quad$ **if** $w(e) > (1+\epsilon)(\phi(r) + \phi(s))$ ▷ *Streaming Process*
9: $\quad\quad\quad w'(e) = w(e) - (\phi(r) + \phi(s))$
10: $\quad\quad\quad \phi(r) = \phi(r) + w'(e); \phi(s) = \phi(s) + w'(e)$
11: $\quad\quad\quad$ Stk.push(e)
12: $\quad$ UB = UB $- (1 - max_s)$
13: $\quad$ **if** $\theta_R >$ UB
14: $\quad\quad$ **return** $\frac{\text{UB}}{(|R|+|S|-\text{UB})}$
15: **while** Stk $\neq \emptyset$ **do** ▷ *Post Processing*
16: $\quad e(r, s) \leftarrow$ Stk.pop()
17: $\quad$ **if** $(V(M) \cap \{r, s\}) = \emptyset$ ▷ $V(M)$: *vertex set covered by* M.
18: $\quad\quad M \leftarrow M \cup \{e\}$
19: **return** $\frac{w(M)+\text{ov}}{(|R|+|S|-(w(M)+\text{ov}))}$

required to be $O(n \log n)$. This model of computation is primarily motivated to solve extreme-scale graphs, where even storing the graph is infeasible. However, as detailed in [7], streaming algorithms could be an efficient alternative to offline algorithms.

We briefly describe the state-of-the-art $\frac{1}{2+\varepsilon}$-approximate semi-streaming algorithm due to Paz and Schartzman (PS) [17], and refer to [7, 9] for a more detailed explanation. In Alg. 1, we show a detailed pseudo-code of the PS algorithm integrated with the verification phase. The PS method begins with an empty stack, Stk, and initializes a set of variables, $\phi(.)$ (approximate dual values) to zero for each vertex of the graph. For an edge (r, s) that arrives in the stream, we push the edge to Stk if its weight is greater than $(1 + \varepsilon)$ times $(\phi(r) + \phi(s))$, otherwise we discard the edge. If $\{r, s\}$ survives, we update the $\phi(r)$ and $\phi(s)$ according to the line 10 as shown in Alg. 1.

In the post-processing phase, we compute a maximal matching in the stack order, by unwinding the stack one edge at a time and inserting into the matching if it does not violate the matching constraints. The PS algorithm requires linear time in the number of edges. The space efficiency comes from the fact that many edges are pruned during the streaming phase and only $O(\frac{n \log n}{\varepsilon})$ edges are survived in the stack. The $\varepsilon > 0$ here provides a trade-off between the memory efficiency and the approximation guarantee.

From the construction of the ϕ values, it can easily be shown that $(1 + \varepsilon)\phi(v), \forall v \in V$ is a feasible dual of dual problem 2. So, using weak-duality of Lemma 2.1, we derive the following observation:

Observation 1. *The* $(1 + \varepsilon) \sum_{v \in V} \phi(v)$ *is an upper bound on the maximum weight bipartite matching of the graph* G.

We experimented with this upper bound as an alternative to the matching score to show greater adaptability of our approximate-based methods.

The streaming nature of the PS method also allows us to maintain a matching as the pair of sets are generated in the set similarity join work flow and we describe this integration with the verification procedure in Algorithm 1. We note that the same graph building process occurs as in Algorithm 2, except the single edge generated is immediately considered for the candidate stack Stk in Line 8 of Algorithm 1. This allows us to discard edges that are not pertinent to our matching as we calculate similarity scores. Following this, we finalize the matching in the post processing scheme of Line 15, building the matching from the edges in Stk and generating a score based on the weight.

Appendix

2 Bipartite Weighted Matching

2.1 LP formulation and Duality

Linear programming (LP) and duality theory play a significant role in designing algorithms for bipartite matching problems. In Eqn. 1 of Fig. 1, we show the primal linear programming relaxation of the MWBM problem. We define $x(e)$ a non-negative real variables for each edge, which encodes the matching in the graph. The objective function is to maximize the sum of weights of the edges in the matching. This linear program relaxes binary x variables in the integer linear program of MWBM, and thus provides an upper bound on the optimum but does not necessarily guarantee an integer (i.e., $x(e) \in \{0, 1\}$) solution. However, for bipartite matching, the primal constraints can be expressed as $Ax \leq b$, where A is an $n \times m$ incidence matrix of G and b is a vector of all 1s. Since A is totally unimodular and b is integral, there exists an integral optimal solution of the primal problem that represents a feasible maximum weighted matching.

$$
\begin{aligned}
&\max \quad \sum_{e \in E} w(e)x(e) && && \min \quad \sum_{v \in V} y(v) && \\
&\text{s.t.} && && \text{s.t.} && \\
&\sum_{e \in \delta(v)} x(e) \leq 1 \quad \forall v \in V && (1) && y(u) + y(v) \geq w(e) \quad \forall \{u, v\} \in E && (2) \\
&x(e) \geq 0 \qquad\quad \forall e \in E && && y(v) \geq 0 \qquad\quad \forall v \in V &&
\end{aligned}
$$

Figure 1: Primal (left) and dual (right) formulations for the LP relaxation of the matching problem.

For each primal problem, we can also devise a dual LP that provides an upper bound on the primal. Here in Eqn. 2, the dual variables y are defined on each vertex and the dual constraints are derived from the standard dual formulation technique. We say a solution pair of both primal and dual problem a feasible pair if both of these solutions (i.e., x and y) satisfy the constraints. We state the following result from standard duality theory of linear programming, which is used in the Hungarian algorithm and the semi-streaming PS algorithm described in section 1.2.

Lemma 2.1. *Let (x, y) be a feasible primal-dual pair and W_* be the weight of a maximum weighted matching, then $\sum_{v \in V} y(v) \geq \sum_{e \in E} w(e) \cdot x(e) \geq W_*$ (Weak Duality). Furthermore, if $\sum_{v \in V} y(v) = \sum_{e \in E} w(e) \cdot x(e)$, and x is integral (which exists since A is totally unimodular), then x encodes the maximum weighted matching of G (Strong Duality).*

2.2 Optimal Matching:

Matching is arguably one of the most studied combinatorial optimization problems [19]. The Hungarian algorithm, as we know from the work of Kuhn [13] for bipartite weighted matching, was in fact known as early as Jacobi [12]. The famous Blossom algorithm of Edmonds [5] that works for general graphs pioneered the notion of polynomial time algorithms as an efficient algorithmic paradigm. The exact algorithms, although polynomial, is both complex and expensive from a real-world dataset perspective. These motivate the development of approximate weighted matching, where instead of computing optimal matching, one seeks for a matching with weights at least $0 < \alpha < 1$ times the optimal.

2.3 Approximate Matching:

Developing approximate matching algorithms has been the forefront of matching research for the last few decades. Among these, the simplest and the most popular one is the 1/2-approximate greedy algorithm [2] (Henceforth, GD), which sorts the edges according to weights in descending order and then scans the edges to compute a maximal matching in this order. The execution time and memory complexity of the greedy algorithm is $O(m \log n)$ and $O(m)$, respectively, for a graph with n vertices and m edges. The first linear time (i.e., $O(m)$) algorithm for 1/2-approximate matching is due to Preis [18], which uses local dominance properties of the greedy matching. Later it was observed that these local properties are also useful in parallel computing environments. The locally dominant algorithm (Henceforth, LD), which matches a vertex to its locally heaviest available neighbor, along with its variant, are the most performant matching algorithm in shared memory [11, 15], distributed memory [21], and GPU architectures [14, 16]. The best linear time approximation algorithm is based on scaling techniques and achieved a $(1 - \epsilon)$-approximation for arbitrarily small constant ϵ [4]. However, this complex algorithm is not shown to perform better than the 1/2-approximate ones for real world graph [1].

2.4 Semi-streaming Matching:

Motivated by solving extremely large matching problems, the most recent developments on matching includes semi-streaming computations, where the edges are streamed one by one and and an algorithm is allowed to use only $O(n \log n)$ space. The semi-streaming model for graph algorithms was proposed in the seminal work of Feigenbaum et al [6], where they developed a 1/6-approximation algorithm for matching. These were improved in subsequent work, culminating to the breakthrough 1/2-approximation algorithm of Paz and Schwartzman (Henceforth, PS algorithm) [17]. The PS algorithm can also be interpreted through primal-dual paradigm [9], which provides a post-posterior instance-wise quality guarantee. Ferdous et al. [7] compared the PS algorithm against state-of-the-art approximate matching algorithms and showed that it is extremely efficient in execution time and memory usage.

In this paper, we integrate three approximate matching algorithms (GD, LD, and PS) into TokenJoin, a state-of-the-art set similarity join workflow, and demonstrate the effectiveness of approximate matching algorithms. To the best of our knowledge, this is the first such work for set similarity join.

References

[1] Ahmed Al-Herz and Alex Pothen. 2022. A 2/3-approximation algorithm for vertex-weighted matching. *Discrete Applied Mathematics* 308 (2022), 46–67.

[2] David Avis. 1983. A survey of heuristics for the weighted matching problem. *Networks* 13, 4 (1983), 475–493.

[3] 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

Algorithm 2 Set Verification

Input: Candidate Sets: R, S, Threshold: θ_R
Output: Score: $sim_\phi(R, S)$

ov $= |R \cap S|$ ▷ *Phase 1: Deduplication*
$R_d, S_d =$ `deduplication`(R, S)
if $R_d = \emptyset$
  return $\frac{\text{ov}}{(|R|+|S|-\text{ov})}$
$G(V, E, w) = \emptyset$, UB $= |R|$ ▷ *Phase 2: Graph Building*
for all $r \in R_d$ **do**
  $max_s = 0$
  for all $s \in S_d$ **do**
    $score =$ `sim`(r, s)
    $max_s =$ `max`$(max_s, score)$
    $V = V \cup r \cup s$
    $E = E \cup \{r, s, score\}$ ▷ *Add edge to graph*
  UB $=$ UB $- (1 - max_s)$
  if $\theta_R >$ UB
    return $\frac{\text{UB}}{(|R|+|S|-\text{UB})}$
$W_M =$ ov $+$ `matching`(G) ▷ *Phase 3: Bipartite Matching*
return $\frac{W_M}{(|R|+|S|-W_M)}$

Algorithm 3 Locally-Dominant (LD) Matching

Input: Graph: $G(V, E, w)$
Output: A locally dominant matching in *mate* array

$M \leftarrow \emptyset$
while G is not empty **do**
  for all $v \in V$ **do** ▷ *Phase 1: Pointing*
    $mate(v) = \arg\max_{u \in \mathcal{N}(v)} w(\{u, v\})$
  for all $e(u, v) \in E$ **do** ▷ *Phase 2: Matching*
    if $mate(v) = u$ and $mate(u) = v$ ▷ *LD edge*
      $M = M \cup e$
      $G = G \setminus \{e \cup \mathcal{N}(e)\}$

[4] Ran Duan and Seth Pettie. 2014. Linear-Time Approximation for Maximum Weight Matching. *J. ACM* 61, 1 (Jan. 2014), 1–23. https://doi.org/10.1145/2529989 ZSCC: 0000137.

[5] Jack Edmonds. 1965. Paths, trees, and flowers. *Canadian Journal of mathematics* 17 (1965), 449–467.

[6] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. 2005. On graph problems in a semi-streaming model. *Theoretical Computer Science* 348, 2-3 (2005), 207–216.

[7] S M Ferdous, Alex Pothen, and Mahantesh Halappanavar. 2024. Streaming Matching and Edge Cover in Practice. In *22nd International Symposium on Experimental Algorithms (SEA 2024) (Leibniz International Proceedings in Informatics (LIPIcs))*, Leo Liberti (Ed.), Vol. 301. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 12:1–12:22. https://doi.org/10.4230/LIPIcs.SEA.2024.12

[8] Zvi Galil. 2002. Efficient algorithms for finding maximum matching in graphs. *Comput. Surveys* 18, 1 (jul 2002), 23–38. https://doi.org/10.1145/6462.6502

[9] Mohsen Ghaffari and David Wajc. 2019. Simplified and Space-Optimal Semi-Streaming $(2 + \epsilon)$-Approximate Matching. In *Proceedings of the 2nd Symposium on Simplicity in Algorithms (SOSA) (OASIcs)*, Vol. 69. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 13:1–13:8. https://doi.org/10.4230/OASIcs.SOSA.2019.13

[10] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.

[11] Mahantesh Halappanavar, John Feo, Oreste Villa, Antonino Tumeo, and Alex Pothen. 2012. Approximate weighted matching on emerging manycore and multithreaded architectures. *The International Journal of High Performance Computing Applications* 26, 4 (2012), 413–430. https://doi.org/10.1177/1094342012452893

[12] CGJ Jocobi and Carl Wilhelm Borchardt. 1865. De investigando ordine systematis aequationum differentialium vulgarium cujuscunque.

[13] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval research logistics quarterly* 2, 1-2 (1955), 83–97.

[14] Michael Mandulak, Sayan Ghosh, SM Ferdous, Mahantesh Halappanvar, and George Slota. 2024. Efficient Weighted Graph Matching on GPUs. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–16.

[15] Fredrik Manne and Mahantesh Halappanavar. 2014. New effective multithreaded matching algorithms. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 519–528.

[16] Md. Naim, Fredrik Manne, Mahantesh Halappanavar, Antonino Tumeo, and Johannes Langguth. 2015. Optimizing Approximate Weighted Matching on Nvidia Kepler K40. In *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*. 105–114. https://doi.org/10.1109/HiPC.2015.15

[17] Ami Paz and Gregory Schwartzman. 2017. A $(2+\epsilon)$-Approximation for Maximum Weight Matching in the Semi-Streaming Model. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2153–2161. https://doi.org/10.1137/1.9781611974782.140

[18] Robert Preis. 1999. Linear Time 1/2-Approximation Algorithm for Maximum Weighted Matching in General Graphs. In *STACS 99*, Christoph Meinel and Sophie Tison (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 259–269.

[19] A. Schrijver. 2003. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer.

[20] 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

[21] Ümit V. Çatalyürek, Florin Dobrian, Assefaw Gebremedhin, Mahantesh Halappanavar, and Alex Pothen. 2011. Distributed-Memory Parallel Algorithms for Matching and Coloring. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. 1971–1980. https://doi.org/10.1109/IPDPS.2011.360