Auto-FuzzyJoin: Auto-Program Fuzzy Similarity Joins Without Labeled Examples

Peng Li*
Georgia Institute of Technology
Atlanta, USA
pengli@gatech.edu

Xiang Cheng*
Georgia Institute of Technology
Atlanta, USA
cxworks@gatech.edu

Xu Chu
Georgia Institute of Technology
Atlanta, USA
xu.chu@cc.gatech.edu

Yeye He
Microsoft Research
Redmond, USA
yeyehe@microsoft.com

Surajit Chaudhuri
Microsoft Research
Redmond, USA
surajitc@microsoft.com

ABSTRACT

Fuzzy similarity join is an important database operator widely used in practice. So far the research community has focused exclusively on optimizing fuzzy join *scalability*. However, practitioners today also struggle to optimize fuzzy-join *quality*, because they face a daunting space of parameters (e.g., distance-functions, distance-thresholds, tokenization-options, etc.), and often have to resort to a manual trial-and-error approach to program these parameters in order to optimize fuzzy-join quality. This key challenge of automatically generating high-quality fuzzy-join programs has received surprisingly little attention thus far.

In this work, we study the problem of "auto-program" fuzzy-joins. Leveraging a geometric interpretation of distance-functions, we develop an unsupervised Auto-FuzzyJoin framework that can infer suitable fuzzy-join programs on given input tables, without requiring explicit human input such as labelled training data. Using Auto-FuzzyJoin, users only need to provide two input tables L and R, and a desired precision target τ (say 0.9). Auto-FuzzyJoin leverages the fact that one of the input is a reference table to automatically program fuzzy-joins that meet the precision target τ in expectation, while maximizing fuzzy-join recall (defined as the number of correctly joined records).

Experiments on both existing benchmarks and a new benchmark with 50 fuzzy-join tasks created from Wikipedia data suggest that the proposed Auto-FuzzyJoin significantly outperforms existing unsupervised approaches, and is surprisingly competitive even against supervised approaches (e.g., Magellan and DeepMatcher) when 50% of ground-truth labels are used as training data. We have released our code and benchmark on GitHub[1] to facilitate future research.

*Both authors contributed equally.
[1]https://github.com/chu-data-lab/AutomaticFuzzyJoin

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452824

CCS CONCEPTS

• **Information systems** → **Entity resolution**; • **Computing methodologies** → **Unsupervised learning**.

KEYWORDS

fuzzy join; similarity join; entity resolution; unsupervised learning

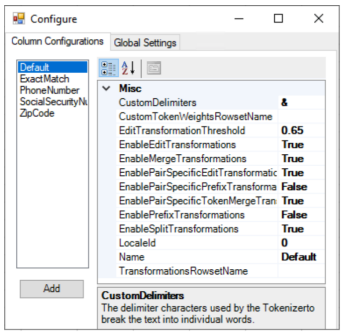
ACM Reference Format:
Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. 2021. Auto-FuzzyJoin: Auto-Program Fuzzy Similarity Joins Without Labeled Examples. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3448016.3452824

1 INTRODUCTION

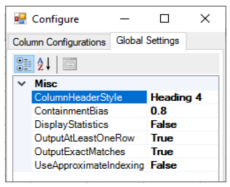
Fuzzy-join, also known as similarity-join, fuzzy-match, and entity resolution, is an important operator that takes two tables L and R as input, and produces record pairs that approximately match from the two tables. Since naive implementations of fuzzy-joins require a quadratic number of comparisons that is prohibitively expensive on large tables, extensive research has been devoted to optimizing the *scalability* of fuzzy-joins (e.g., [11, 12, 14, 20, 23, 26, 36, 39]). We have witnessed a fruitful line of research producing significant progress in this area, leading to wide adoption of fuzzy-join as features in commercial systems that can successfully scale to large tables (e.g., Microsoft Excel [3], Power Query [8], and Alteryx [1]).

The need to parameterize fuzzy-joins. With the scalability of fuzzy-join being greatly improved, we argue that the usability of fuzzy-join has now become a main pain-point. Specifically, given the need to optimize join quality for different input tables, today's fuzzy-join implementations offer rich configurations and a puzzling number of parameters, many of which need to be carefully tuned before high-quality fuzzy-joins can be produced.

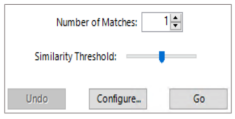
Microsoft Excel, for instance, has a popular fuzzy-join feature available as an add-in [3]. It exposes a rich configuration space, with a total of 19 configurable options across 3 dialogs, as shown in Figure 1. Out of the 19 options, 11 are binary (can be either `true` or `false`), which already correspond to $2^{11} = 2048$ discrete configurations, which are clearly difficult to program manually. Similarly, `py_stringmatching` [6], a popular open-source fuzzy-join package, boasts 92 options. Note that we have not yet included parameters



(a) Params in the "Custom Setting" dialog



(b) Params in the "Global Setting" dialog



(c) Params in the "Threshold" dialog

Figure 1: A total of 19 parameters exposed to users in Fuzzy-join for Microsoft Excel (across 3 dialog windows).

Parameters	Sample Values	Example of Results
Preprocessing (P)	○ Lowercase (L) ○ Remove Punctuation (RP) ○ Stemming (S) ○ ...	**Input:** "2008 LSU baseball team" **Result:** L: "2008 lsu baseball team" S: "2008 LSU baseball team"
Tokenization (T)	○ 2-Gram (2G) ○ 3-Gram (3G) ○ Space (SP) ○ ...	**Input:** "2008 lsu baseball team" **Result:** 3G : {'$$2', '$20', '200', '008', ..., 'm$$'} SP: {'2008', 'lsu', 'baseball', 'team'}
Token Weights (W)	○ Equal Weights (EW) ○ IDF Weights (IDFW) ○ ...	**Input Tokens:** {'2008', 'lsu', 'baseball', 'team'} EW: $w_{2008} = w_{lsu} = w_{baseball} = w_{team} = 1$ IDFW: $w_{2008} = 2$, $w_{lsu} = 8$, $w_{baseball} = 4$, $w_{team} = 1.2$
Distance Function (D)	• Set-Based Distances ○ Jaccard Distance (JD) ○ Cosine Distance (CD) ○ Max-include Distance (MD) ○ Dice Distance (DD) ○ Intersection Distance (ID) ○ ...	**Input:** l ={'2012', 'tigers', 'lsu', 'baseball', 'team'}, r ={'2012', 'lsu', 'baseball', 'team'}. Assume equal weight. **Result:** JD:0.2, CD: 0.11, MD:0, DD:0. 11, ID: 0. 56
	• Character-Based Distances ○ JaroWinkler (JW) ○ Edit Distance (ED) ○ ...	**Input:** l = "2012 tigers lsu baseball team", r = "2012 lsu baseball team" **Result:** JW: 0.14, ED: 7
	• Embedding Distances ○ GloVe (GED) ○ FastText (FED) ○ ...	**Input:** l = "2012 tigers lsu baseball team", r = "2012 lsu baseball team" **Result:** GED: 0.05, FED: 0.06
	• Number-Based ○ Absolute Difference (AD) ○ Percentage Difference (PD) ○ ...	**Input:** $l = 9, r = 1$ **Result:** AD: 8, RD: 160%

Figure 2: A sample of fuzzy-join parameters.

from numeric continuous domains, e.g., "similarity threshold" and "containment bias" that can take any value in ranges like $[0, 1]$.

Not surprisingly, we have seen recurring user questions in places like Excel user forums, asking how fuzzy-joins can be programmed appropriately, including how to set parameters like similarity-thresholds[2], token-weights[3], distance-functions[4], multi-column settings[5], etc. We note that these parameters are widely used in the literature [11, 12, 14, 23, 26, 36, 39], which can be broadly classified into four categories: Pre-processing, Tokenization, Token-weights, and Distance-functions, as shown in Figure 2 (we will describe these options in more detail in Section 2.2).

While seasoned practitioners may inspect input data and use their experience to make educated guess of suitable parameters to use (often still requiring trials-and-errors); less-technical users (e.g., those in Excel or Tableau) struggle as they either have to laboriously try an infeasibly large number of parameter combinations, or live with the sub-optimal default parameters. We argue that this is a

[2]https://www.reddit.com/r/excel/comments/9y6o6a/how_is_similarity_threshold_calculated_when_doing/

[3]https://answers.microsoft.com/en-us/msoffice/forum/all/token-weights-for-fuzzy-lookup-add-in-for-excel/c9c4a0f3-014f-4e2e-8672-b2303cfe3a4d

[4]https://www.excelforum.com/excel-programming-vba-macros/810739-fuzzy-logic-search-for-similar-values.html

[5]https://www.mrexcel.com/forum/excel-questions/659776-fuzzy-lookup-add-multiple-configurations-one-matchup.html

L-id	L-Table (Reference Table)		R-id	R-Table (Input Table)
l_1	2008 LSU Tigers baseball team	✓	r_1	2008 LSU baseball team
l_2	2008 LSU Tigers football team	✓	r_2	2008 LSU football team
l_3	2008 Mississippi State Bulldogs baseball team	✓	r_3	2008 Missisippi State Bulldog baseball team
l_4	2008 Mississippi State Bulldogs football team	✓	r_4	2008 Mississippi State Bulldog football team
l_5	...		r_5	...
l_6	2007 LSU Tigers football team	✗	r_6	2007 LSU Tigers baseball team
l_7	2007 Wisconsin Badgers football team	✗	r_7	2008 Wisconsin Badgers football team
l_8	2011 LSU Tigers football team	✗	r_8	2010 LSU Tigers football team
l_9	2007 LSU Tigers baseball team	✗	r_9	2007 LSU Tigers football team

(a) Example: NCAA-Teams. (l_1, r_1), (l_2, r_2) are joined using Jaccard-distance, (l_3, r_3), (l_4, r_4) are joined using Edit-distance. (l_6, r_6) are not joined because of an inferred *negative-rule* "football" ≠ "baseball", (l_7, r_7) are not joined because "2007" ≠ "2008".

L-id	L-Table (Reference Table)		R-id	R-Table (Input Table)
l_1	Super Bowl XX Championship Game	✗	r_1	Super Bowl XI Championship Game
l_2	Super Bowl XXI Championship Game	✗	r_2	Super Bowl XVI Championship Game
l_3	Super Bowl XXII Championship Game	✗	r_3	Super Bowl XVII Championship Game
l_4	...		r_4	...
l_5	Super Bowl XL Championship Game	✓	r_5	Super Bowl XL Game
l_6	Super Bowl XLI Championship Game	✓	r_6	Super Bowl XLI Game

(b) Example: Super Bowl Games. (l_1, r_1), (l_2, r_2) are not joined despite of having small Edit-distance. Pairs like (l_5, r_5), (l_6, r_6) are joined based on Jaccard-containment.

Figure 3: Examples of Fuzzy Join Cases.

significant pain point, and a major roadblock to wider adoption of fuzzy-join.

In this paper, we explore the possibility of automatically programming fuzzy-joins, using suitable parameters tailored to given input tables. Our approach is designed to be *unsupervised*, requiring no inputs from human users (e.g., labeled training examples for matches vs. non-matches). It exploits a key property of fuzzy-join tasks, which is that one of the input tables is often a "reference table", or a curated master table that contains few or no duplicates. We note that the notion of reference tables is widely used in the literature (e.g., [18, 19, 44]), and adopted by commercial systems (e.g., SQL Server [4], OpenRefine/GoogleRefine [5], Excel [3], etc.). As we will see, leveraging this key property of reference tables allows us to infer high-quality fuzzy-joins programs without using labeled data.

An intuitive example. We illustrate a few key ideas we leverage to auto-program fuzzy-joins using an intuitive example. On the left of Figure 3(a) is a reference table L with NCAA team names, and on the right is a table R with team names that need to be matched against L. As can be seen, (l_1, r_1) and (l_2, r_2) share a large set of common tokens, so intuitively we should tokenize by word boundaries and join them using set-based metrics like Jaccard distance or Jaccard-Containment distance. On the other hand, (l_3, r_3) should intuitively also join, but their token overlap is not high (Jaccard distance can be computed as 0.5), because of misspelled "Missisippi" and "Bulldog" in r_3. Such pairs are best joined by viewing input strings as sequences of characters, and compared using Edit-distance (e.g., Edit-distance$(l, r) < 3$).

Union-of-Configurations. Because different types of string variations are often present at the same time (e.g., typos vs. extraneous tokens), ideally a fuzzy-join program should contain a *union* of fuzzy-join configurations to optimize recall – in the example above, Edit-distance$(l, r) < 3 \vee$ Jaccard-distance$(l, r) < 0.2$, to correctly join these records. Note that for humans, programming such a union of configurations manually is even more challenging than tuning for a single configuration. Our approach can automatically search disjunctive join programs suitable for two given input tables, which can achieve optimized join quality (Section 2.2).

Learn-safe-join-boundaries. In order to determine suitable parameters for fuzzy-joins on given input tables, we leverage reference-tables L to infer what fuzzy-join boundaries are "safe" (generating

few false-positives). In Figure 3(b) for instance, unlike Figure 3(a), even a seemingly small Edit-distance$(l, r) \leq 1$ is not "safe" on this data set, and would produce many false-positives like (l_1, r_1), (l_2, r_2), etc., none of which are correct joins. While it may be obvious to humans recognizing roman numerals in the data, it is hard for algorithms to know without labeled examples. Here we leverage an implicit property of the reference table L that it has few or no duplicates, to perform automated inference. Assume for a moment that a fuzzy-join with Edit-distance$(l, r) \leq 1$ on this pair of L and R is a "safe" distance to use. Because L and R are similar in nature, it then follows that this fuzzy-join on L and L is also "safe". However, applying this self-fuzzy-join on L leads to many joined pairs like (l_1, l_2), (l_2, l_3), etc., contradicting with the belief that L has few fuzzy-duplicates, and suggesting that Edit-distance$(l, r) \leq 1$ likely joins overly aggressively and is actually not "safe". We generalize this idea using a geometric interpretation of distance functions in a fine-grained (per-record) manner, in order to learn "safe" fuzzy-join programs that can maximize recall while ensuring high precision (Section 3.1). This is a key idea behind Auto-FuzzyJoin.

Negative-rule-learning. We observe that similarity functions and scores alone are sometimes still not sufficient for high-quality fuzzy joins. For instance, existing solutions would join (l_6, r_6) and (l_7, r_7) in Figure 3(a) because of their high similarity scores, which as humans we know are false-positive results. Our approach is able to automatically learn what we call "negative rules", by analyzing the reference table L. Specifically, we will find many pairs of records like ("2008 LSU Tigers baseball team", "2008 LST Tigers football team") present at the same time in the reference table L, and because these are from the reference table and thus unlikely to be duplicates, we can infer a negative rule of the form "baseball" ≠ "football", which would prevent (l_6, r_6) from being joined. Similarly, we can learn a negative rule like "2007" ≠ "2008", so that (l_7, r_7) is not joined. (Section 3.3).

Key features of Auto-FuzzyJoin. Auto-FuzzyJoin has the following features that we would like to highlight:

- Unsupervised. Unlike most existing methods, Auto-FuzzyJoin does not require labeled examples of matches/non-matches.
- High-Quality. Despite not using labeled examples, it outperforms strong supervised baselines (e.g., Magellan and DeepMatcher) even when 50% of ground-truth joins are used as training data.
- Robust. Our approach is robust to tables with varying characteristics, including challenging test cases adversarially-constructed.
- Explainable. Compared to black-box methods (e.g., deep models), our approach produces fuzzy-join programs in a disjunctive form, which is easy for practitioners to understand and verify.
- Extensible. Parameter options listed in Figure 2 are not meant to be exhaustive, and can be easily extended (e.g., new distance functions) in our framework in a manner transparent to users.

2 PRELIMINARIES

2.1 Many-to-one Fuzzy Joins

Definition 2.1. *Let L and R be two input tables, where L is the* reference table. A fuzzy join J between L and R is a *many-to-one join, defined as $J : R \rightarrow L \cup \perp$.*

The fuzzy join J defines a mapping from each tuple $r_j \in R$ to either one tuple $l_i \in L$, or an empty symbol $\perp$, indicating that no matching record exists in L for r_j, as L may be incomplete. Note that because L is a reference table, each r_j can join with at most one tuple in L. However, in the other direction, each tuple in L can join with multiple tuples in R, hence a many-to-one join.

The notion of reference tables is widely used both in the fuzzy-join/entity-matching literature (e.g., [18, 19, 44]) and commercial systems (e.g., Excel [3], SQL Server [4], OpenRefine/GoogleRefine [5], etc.). In practice, we find that most benchmark datasets used for entity-resolution in the literature indeed have a reference table, for which our approach is applicable. For example, in the well-known Magellan data repository of ER[6], we find 19/29 datasets to have a reference table that is completely duplicate-free, and 26/29 datasets to have a reference table that has less than 5% duplicates, confirming the prevalence of reference tables in practice.[7]

The reference table property essentially serves as a structural constraint to prevent our algorithm from using fuzzy-join configurations that are too "loose" (or join more than what is correct). We refer readers to a full version of the paper [9] for an additional analysis showing that some form of constraints are necessary before good fuzzy-joins can be inferred.

2.2 The Space of Join Configurations

A standard way to perform fuzzy-join is to compute a distance score between r and l. There is a rich space of parameters that determine how distance scores are computed. Figure 2 gives a sample such space. There are four broad classes of parameters: pre-processing (P), tokenization (T), token-weights (W), and distance-functions (D). The second column of the figure gives example parameter options commonly used in practice [11, 12, 14, 23, 26, 36, 39]. Combination of these parameters (P, T, W, D) uniquely determines a distance score for two input strings (l, r), which we term as a *join function* $f \in \mathcal{F}$, where $\mathcal{F}$ denotes the space of join functions.

Example 2.1. *Consider join function $f = (L, SP, EW, JD)$, which uses lower-casing (L), space-tokenization (SP), equal-weights (EW), and Jaccard-distance (JD) from Figure 2. Applying this f to l_1, r_1 in Figure 3(a), we can compute $f(l_1, r_1) = 0.2$. Additional examples of score computation are shown in the last column of Figure 2.*

In our experiments we consider a rich space with hundreds of join functions. Our Auto-FuzzyJoin approach treats these parameters as black-boxes, and as such can be easily extended to additional parameters not listed in Figure 2.

Given distance $f(l, r)$ computed using f, the standard approach is to compare it with a threshold θ to decide whether l and r can be joined. Together θ and f define a *join configuration C*.

Definition 2.2. *A join configuration C is a 2-tuple $C = \langle f, \theta \rangle$, where $f \in \mathcal{F}$ is a join function, while θ is a threshold. We use $S = \{\langle f, \theta \rangle | f \in \mathcal{F}, \theta \in \mathbb{R}\}$ to denote the space of join configurations.*

Given two tables L and R, a join configuration $C \in S$ induces a fuzzy join mapping J_C, defined as:

$$J_C(r) = \underset{l \in L, f(l,r) \leq \theta}{\arg\min} \; f(l, r), \forall r \in R \tag{1}$$

[6] https://sites.google.com/site/anhaidgroup/useful-stuff/data

[7] As we will see, for cases where reference tables are absent, our approach will still work but may generate overly conservative fuzzy-join programs, which is still of high precision but may have reduced recall.

The fuzzy join J_C defined in Equation (1) ensures that each r record joins with $l \in L$ with the smallest distance. Note that this can also be empty if $f(l, r) > \theta, \forall l \in L$.

We observe that real data often have different types of variations simultaneously (e.g., typos vs. missing tokens vs. extraneous information), one join configuration alone is often not enough to ensure high recall. For example, in Figure 3(a), Jaccard distance with threshold 0.2 may be suitable for joining pairs like (l_1, r_1) as these pairs differ by one or two tokens. However, for pairs like (l_3, r_3) that have spelling variations, Jaccard-distance (0.5) is high, and Edit-distance is required to join (l_3, r_3).

In order to handle different types of string variations, in this work our algorithm will search for joins that use a set of configurations $U = \{C_1, C_2, \ldots, C_K\}$ (as opposed to a single configuration), where the join result of U is defined as the union of the result from each configuration C_i.

Definition 2.3. *Given L and R, a set of join configurations $U = \{C_1, C_2, \ldots, C_K\}$ induces a fuzzy join mapping J_U, defined as:*

$$J_U(r) = \bigcup_{C_i \in U} J_{C_i}(r), \forall r \in R \tag{2}$$

Intuitively, each C_i produces high-quality joins capturing a specific type of string variations, and two records are joined in U if and only if they are joined by one configuration $C_i \in U$ (we discuss scenarios with conflicts in Section 3).

2.3 Auto-FuzzyJoin: Problem Statement

Given R and L, and a space of join configurations $\mathcal{S}$, the problem is to find a set of join configurations U that produces "good" fuzzy-join results. Let J_U denote the fuzzy join mapping induced by U and let J_G denote the ground truth fuzzy join mapping. The "goodness" of a solution U can be measured using *precision* and *recall*:

$$precision(U) = \frac{|\{r \mid r \in R,\ J_U(r) \neq \emptyset,\ J_U(r) = J_G(r)\}|}{|\{r \mid r \in R,\ J_U(r) \neq \emptyset\}|} \tag{3}$$

$$recall(U) = |\{r \mid r \in R,\ J_U(r) \neq \emptyset,\ J_U(r) = J_G(r)\}| \tag{4}$$

The *precision* of U is the fraction of predicted joins that are correct according to the ground-truth; and the *recall* of U is defined as the number of correct matches (a variant widely-used in the IR literature [38]). We note that this definition of recall in absolute terms simplifies our analysis, which is no different from the relative recall [13], because the total number of correct joins ($|\{r \mid r \in R,\ J_G(r) \neq \emptyset\}|$) is always a constant for a given data set.

Problem Statement. Given L and R, and a target precision τ. Let $\mathcal{S} = \{\langle f, \theta\rangle | f \in \mathcal{F}, \theta \in \mathbb{R}\}$ be the space of fuzzy-join configurations. We would like to find a set of configurations $U = \{C_1, C_2, \ldots, C_K\}$ with $C_i \in S$, that maximizes $recall(U)$, while observing the required precision τ. This recall-maximizing fuzzy-join problem (RM-FJ) formulation can be written as an optimization problem:

$$\text{(RM-FJ)} \qquad \max\ recall(U) \tag{5}$$

$$\text{s.t. } precision(U) \geq \tau \tag{6}$$

$$U \in 2^S \tag{7}$$

Theorem 2.1. *The decision version of the RM-FJ problem is NP-hard.*

The hardness result can be obtained using a reduction from densest-k-subhypergraph [30].

3 SINGLE-COLUMN AUTO-FUZZYJOIN

We now discuss Auto-FuzzyJoin when the join key is a pair of single columns (we will extend it to multi-columns in Section 4).

3.1 Estimate Precision and Recall

In the RM-FJ formulation above, the hardness result assumes that we can compute the precision and recall of any configuration U. In reality, however, we are only given L and R, with no ground truth. To solve RM-FJ, we first need a way to estimate precision/recall of a fuzzy-join without using ground-truth labels, which is a key challenge we need to address in Auto-FuzzyJoin.

In the following, we show how precision/recall can be estimated in our specific context of fuzzy-joins, by leveraging a geometric interpretation of distances, and unique properties of the reference table L. For ease of exposition, we will start our discussion with a single join configuration C, before extending it to a set of configurations $U = \{C_1, C_2, \ldots, C_K\}$.

Estimate for a single-configuration C. Given a configuration C, and two tables L and R, we show how to estimate the precision/recall of C. Recall that a configuration $C = \langle f, \theta\rangle$ consists of a join function f that computes distance between two records, and a threshold θ.

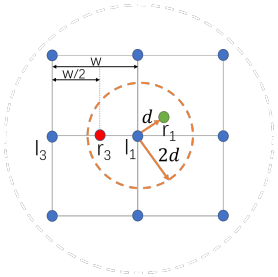
Assuming a "complete" L. We will start by analyzing a simplified scenario where the reference table L is assumed to be *complete* (with no missing records). This is not required in our approach, but used only to simplify our analysis (which will be relaxed later).

Using a geometric interpretation, given some distance function f, records in a table can intuitively be viewed as points embedded in a multi-dimensional space (e.g., with metric embedding [10]), like visualized in Figure 4.

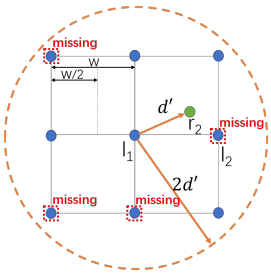
When L is complete (containing all possible l records in the same domain), for each $l \in L$, the *closest neighbors* of each l tend to differ in some standardized/structured manner, making these *closest neighbors* to have similar distances to l. For instance, for most records l in Figure 3(a), their closest neighbors differ from l by only one token (either year or sport-name), which translates to a Jaccard-distance of around 0.2. In Figure 3(b), for most records l, their closest neighbors differ from l by one character (in roman numerals), or an Edit-distance of 1. The same extends to many other domains – e.g., in a reference table with addresses, closest neighbors to each l will likely differ from l by only house-numbers (one token); and in a reference table with people-names, closest neighbors to each l will likely differ by only last-names (one token), etc.

Because the closest neighbors of l tend to have similar distances to l, we can intuitively visualize L points in the local neighbors of each l as points on unit-grids in a multi-dimensional space – Figure 4 visualizes reference records l as blue points on the grid, with similar distances between close neighbors (this is shown in 2D but can generalize into higher dimensions too).

Using an analogy from astronomy, we can intuitively think of reference records in L as "stars" on unit-grids, whereas the query records $r \in R$ (having different string variations from their corresponding l) can be thought of as "planets" orbiting around "stars" (with different distances to their corresponding l). Determining which l should each r join amounts to finding the closest l (star), when using a suitable distance function f.



(a) l_1 is the closest left record to r_1. We say (l_1, r_1) is a "safe" join, because no other L records exist in the ball of distance $2d$.



(b) l_1 is the closest left record to r_2, since l_2 is missing from L. We can infer that (l_1, r_2) is not a "safe" join, because we find many L records in the ball of $2d'$.

Figure 4: Infer whether a join pair (l, r) is "safe", when using a join function f. We compute distance $d = f(l, r)$, and draw a ball of distance $2d$ centered around l. The more L records we find in the $2d$-ball, the more likely this join is not "safe".

In an idealized setting where L is *complete*, conceptually finding the correct left record to join for a given r is straightforward, as one only needs to find the closest l. In Figure 4(a) for example, we would join r_1 with l_1 as l_1 is the closest left record (blue dots) to r_1.

Dealing with an "incomplete" L. In practice, however, L can be incomplete or have many missing records – in our visual model, this would lead to missing blue points on the grid. Given some $r \in R$ whose corresponding l record is missing in L, the simplistic approach of joining this r with its closest $l \in L$ leads to a false-positive and lower precision.

For example, in Figure 4(b), r_2 is a record that should join with the reference record l_2, which however is currently missing in L. This makes correct fuzzy-joins challenging, because given l_2 is absent in L, the closest L record to r_2 becomes l_1, and a naive approach would attempt to fuzzy-join (r_2, l_1) using a distance of $d = f(r_2, l_1)$, which creates a false-positive join. The key question we try to address, is how to infer (without using ground-truth) that this particular d is too lax of a distance to use, and the resulting (r_2, l_1) join is likely a "bad" join (false-positive).

Our key idea here, is to infer the distances between L records, and use that to determine "safe" fuzzy-join distances to use. Specifically, recall that when L is complete and L records are visualized on unit-grids, like shown in Figure 4(a), closest neighbors of an l tend to have similar distances to l, which we refer to as w, or the "width" of the grid (shown in the figure). A hypothetical record r_3 in Figure 4(a) that lies right in between of l_1 and l_3 is then of distance $\frac{w}{2}$ to both the two L records, which intuitively cannot be reliable joined with either l_1 or l_2. (Analogously, a "planet" lying right in between two "stars" cannot be "claimed" by either). Intuitively, we can see that in this case, the "safe" distance to join a pair of (l, r) is when $d = f(l, r) < \frac{w}{2}$, which is when this r would clearly lie on one side and be closest to one L record. (This can be shown formally via triangle-inequality).

In the case when L is incomplete with missing records, like shown in Figure 4(b), estimating this grid-width w may not be as straightforward. As a result, we perform this analysis in the other direction – given a pair (l, r) that we want to join, we compute their distance $d = f(l, r)$. We then draw a ball centered around l with a radius of $2d$, and test how many additional L records would fall within this $2d$-ball. Because if $d < \frac{w}{2}$, which is a "safe" distance to join based on our analysis above, then it follows that $2d < w$, meaning that in this $2d$-ball centered around l we should expect to see no other L records (except l). If we indeed see no L record in the $2d$-ball, we can be confident that this d used to join (l, r) is small enough and the join is "safe". Alternatively, if we observe many L records in the $2d$-ball, this likely indicates that $2d \geq w$, or $d \geq \frac{w}{2}$, which based on our analysis above is too "lax" of a distance to be "safe".

Example 3.1. *In Figure 4(a), to join r_1, we first find its closest L, which is l_1, and compute $d = f(l_1, r_1)$. We then draw this $2d$ ball around l_1, and find no other L records, indicating that this d is small enough and a "safe" distance to use for fuzzy-joins based on l_1's local neighborhood.*

In Figure 4(b), r_2 should join l_2, which however is missing in L. In this case, we would find l_1 to be closest to r_2 in the absence of l_2, with a distance $d' = f(l_1, r_2)$. When we draw a $2d'$ ball around l_1, we find many additional L records, which based on our analysis above indicates that it is likely that this $d' \geq \frac{w}{2}$, which is too "lax" to use in this local neighborhood, and we should not join r_2 with l_1.

Note that in Figure 4(b), we have 4 missing L records (marked by dotted rectangles). This incomplete L, however, still allows us to conclude that joining (l_1, r_2) is not "safe". In fact, in this 2-D example, we can "tolerate" up to 7 missing L records in the neighborhood while still correctly deciding that (l_1, r_2) is likely not "safe" to join. We should note that this tolerance level goes up exponentially when records are embedded in a higher-dimensional space (e.g., in a 3-D unit-cube, we can tolerate up to 25 missing l out of 27 positions).

Estimating join precision. Given $r \in R$, let $l \in L$ be the closest to r with distance $d = f(l, r)$, we can estimate the precision of this join pair (l, r) (the likelihood of it being correct), to be the inverse of the number of L records within the $2d$ ball. We write this as $precision(l, r)$, shown in Equation (8). We use the multiplicative-inverse to estimate precision, because all l within the 2d-ball are reasonably close to r, and are thus plausible counterparts to join with this l.

$$precision(l, r) = \frac{1}{|\{l' | l' \in L, f(l, l') \leq 2f(l, r)\}|} \tag{8}$$

Example 3.2. *The precision of (l_1, r_1) in Figure 4(a) can be estimated as 1 per Equation (8), because l_1 is closest to r_1, and the $2d$ ball around l_1 has only one L record (itself).*

The precision of (l_1, r_2) in Figure 4(b) can be estimated as $\frac{1}{5}$, since the $2d'$ ball has 5 L records (note that 4 L records are missing).

For an example from tables, we revisit Figure 3(b). Here for r_1, the closest in L by Edit-distance is l_1 with $d = 1$. While the pair is as close as it gets for Edit-distance, the 2d-ball around l_1 (with a radius of Edit-distance=2) has many L records (e.g., l_2, l_5, etc.), indicating that the join (r_1, l_1) is of low precision.

We would like to note that this estimate $precision(l, r)$ is not intended to be exact when L is incomplete. Because in our application users typically want high-precision fuzzy-joins (e.g., target precision of 0.9 or 0.8), our precision estimate only needs to be informative to qualitatively differentiate between high-confidence joins (clean balls), and low-confidence joins (balls with more than one L record). As soon as the balls contain more than one L record, the estimated precision drops quickly to below 0.5, at which point our algorithm would try to avoid given a high precision target (i.e., it does not really matter if the estimate should really be $\frac{1}{5}$ or $\frac{1}{8}$).

Using the precision estimate for a single (l, r) pair in Equation (8), we can now estimate precision for a given configuration $C = \langle f, \theta \rangle$. Recall that given C, each $r \in R$ is joined with $J_C(r)$ (defined in Equation (1)), which can be an $l \in L$ or empty (no suitable l to join with). The estimated precision of a r joined using C if $J_C(r) \neq \emptyset$ is:

$$precision(r, C) = \frac{1}{|\{l' | l' \in L, l = J_C(r), f(l, l') \leq 2\theta\}|} \tag{9}$$

The expected number of true-positives $TP(C)$ is the sum of expected precision of each r that C can join:

$$TP(C) = \sum_{r \in R, J_C(r) \neq \emptyset} precision(r, C) \tag{10}$$

And the expected number of false-positives $FP(C)$ is:

$$FP(C) = \sum_{r \in R, J_C(r) \neq \emptyset} \big(1 - precision(r, C)\big) \tag{11}$$

Thus, the estimated precision and recall of a given C is:

$$precision(C) = \frac{TP(C)}{TP(C) + FP(C)}, recall(C) = TP(C) \tag{12}$$

Estimate for a set of configurations U. We now discuss how to estimate the quality for a set of configurations U.

In the simple (and most common) scenario, the join assignment of each $r \in R$ has no conflicts within U. This can be equivalently written as $\forall r \in R, |J_U(r)| \leq 1$ (recall $J_U(r)$ is the result induced by U defined in Equation (2)). In such scenarios, estimating for U is straightforward. $TP(U)$ can be simply estimated as $TP(U) = \sum_{C \in U} TP(C)$, and $FP(U)$ as $FP(U) = \sum_{C \in U} FP(C)$.

It is more complex when some r has conflicting join assigning in U, with say $J_{C_i}(r) = l$ and $J_{C_j}(r) = l'$, where $l \neq l'$. Because we know each r should only join with at most one $l \in L$ (as L is the reference table), we use our precision estimate in Equation (9) to compare $precision(r, C_i)$ and $precision(r, C_j)$, and pick the more confident join as our final assignment. Other derived estimates like $TP(U)$ and $FP(U)$ can be updated accordingly.

Given $TP(U)$ and $FP(U)$, the estimated precision/recall of U is:

$$precision(U) = \frac{TP(U)}{TP(U) + FP(U)}, recall(U) = TP(U) \tag{13}$$

3.2 AutoFJ Algorithm

Given the hardness result, we propose an intuitive and efficient greedy approach AutoFJ to solve the RM-FJ problem. Recall that our goal is to maximize recall while keeping precision above a certain threshold τ, where precision and recall can be estimated according to Equation (13). A greedy strategy is then to prefer configurations that can produce the most number of true-positives (TP), i.e., maximal recall, at the "cost" of introducing as few false-positives as possible (FP), i.e., minimal precision loss. We call this ratio of TP to FP "profit" to quantify how desirable a solution is:

$$profit(U) = \frac{\text{TP(U)}}{\text{FP(U)}} \tag{14}$$

Algorithm 1 AutoFJ for single column

Require: Tables L and R, precision target τ, search space S
1: $LL, LR \leftarrow$ apply blocking with $L - L$ and $L - R$
2: $LR \leftarrow$ Learn negative-rules from LL and apply rules on LR (Alg. 2)
3: Compute distance with different join functions $f \in S$
4: Pre-compute precision estimation for each configuration $C \in S$
5: $U \leftarrow \emptyset$
6: **while** $S \setminus U \neq \emptyset$ **do**
7: $\quad max\_profit \leftarrow 0$
8: $\quad$ **for all** $C \in S \setminus U$ **do**
9: $\quad\quad$ **if** $profit(U \cup \{C\}) > max\_profit$ **then**
10: $\quad\quad\quad C^* \leftarrow C, max\_profit \leftarrow profit(U \cup \{C\})$
11: $\quad$ **if** $precision(U \cup \{C^*\}) > \tau$ **then**
12: $\quad\quad U \leftarrow U \cup \{C^*\}$
13: $\quad$ **else**
14: $\quad\quad$ **break**
15: **return** U

Given a space of possible configurations S, our greedy algorithm in Algorithm 1 starts with an empty solution U (Line 5). It iteratively finds the configuration from the remaining candidates in $S \setminus U$, whose addition into the current U leads to the highest profit (Line 9).[8] The entire greedy algorithm terminates when the estimated precision of U falls below the threshold τ (Line 14) or there is no remaining candidate configurations (Line 6).

Efficiency Optimizations. We perform two main optimizations to improve efficiency of the greedy algorithm. First, we pre-compute $precision(r, C)$ $\forall r \in R, C \in S$ based on given L and R, as opposed to computing these measures repeatedly in each iteration.

Second, we apply blocking [15, 20, 41] to avoid comparing all record pairs. However, unlike standard blocking, we could not expect users to tune parameters in the blocking component (e.g., tokenization schemes, what fraction of tokens to keep, etc.) based on input data, precisely because our goal is to have end-to-end hands-off Auto-FuzzyJoin. Instead of performing automated parameter-tuning for blocking, we use a default blocking that is empirically effective: we use 3-gram tokenization to tokenize each record and we use TF-IDF weighting schema to weight each token; we measure the similarity between each l and r by summing the weights of their common tokens; for each r, we keep the top $|\sqrt{L}|$ number of candidate matches from L with the largest similarity scores and block others. As we will show in experiments, our default blocking strategy achieves a significant reduction in running time with close to zero loss in recall.

Complexity of Algorithm 1. Since the number of L-L and L-R tuple pairs after blocking is $|L|\sqrt{|L|} + |R|\sqrt{|L|}$, it takes $O(|S|(|L|\sqrt{|L|} + |R|\sqrt{|L|}))$ to compute the distance (Line 3). To compute $precision(r, C)$, we need to first find $l \in L$ closest to r, then we need to find $l' \in L$ that have distance smaller than 2θ with l. Since after blocking, for each $r \in R$ or $l \in L$, we have $|\sqrt{L}|$ records in the candidate set. Hence the time complexity for computing the $precision(r, C)$ is $O(\sqrt{L})$ and the complexity for the pre-computing step (Line 4) is $O(|S||R|\sqrt{|L|})$. At each iteration, with our pre-computation, it takes $O(1)$ time to compute the profit for each configuration (Line 9). Therefore, the time complexity of greedy steps (Line 6 to Line 14) is $O(|S||R|)$ (we have at most $|R|$ iterations since each iteration needs to join a new right record to increase

[8] If there are multiple configurations with the same profit at an iteration, which rarely happens on large datasets, we break ties randomly.

Algorithm 2 Learning and Applying Negative Rules

Require: Tables L and R, LL and LR
1: Apply lowercasing, stemming, and removing punctuation for all L and R
2: $NR \leftarrow \emptyset$
3: **for** $l_1, l_2 \in LL$ **do**
4: $W_1 \leftarrow$ set of words of l_1, $W_2 \leftarrow$ set of words of l_2
5: $\Delta_1 \leftarrow |W_1 \setminus W_2|, \Delta_2 \leftarrow |W_2 \setminus W_1|$
6: **if** $|\Delta_1| = 1$ and $|\Delta_2| = 1$ **then**
7: $NR \leftarrow NR \cup (\Delta_1, \Delta_2)$
8: **for** $l, r \in LR$ **do**
9: $W_1 \leftarrow$ set of words of l, $W_2 \leftarrow$ set of words of r
10: $\Delta_1 \leftarrow |W_1 \setminus W_2|, \Delta_2 \leftarrow |W_2 \setminus W_1|$
11: **if** $|\Delta_1| = 1$ and $|\Delta_2| = 1$ and $(\Delta_1, \Delta_2) \in NR$ **then**
12: Remove (l, r) from LR
13: **return** LR

profit). Hence, the total time complexity is $O(|S||L|\sqrt{|L|}+|S||R|\sqrt{|L|})$. The space complexity is dominated by computing distance between tuple pairs, which is in $O(|S||L|\sqrt{|L|} + |S||R|\sqrt{|L|})$.

3.3 Learning of Negative-Rules

While tuning fuzzy-join parameters is clearly important and useful, we observe that there is an additional opportunity to improve join quality not currently explored in the literature.

Specifically, in many real datasets there are record pairs that are syntactically similar but should not join. For example, in Figure 3(a), (l_6, r_6) with "2007 LSU Tigers football team" and "2007 LSU Tigers baseball team" should not join despite their high similarity, because as human we know that "football" ≠ "baseball". Similarly (l_7, r_7) with "2007 Wisconsin Badgers football team" and "2008 Wisconsin Badgers football team" should not join, since "2007" ≠ "2008".

Such negative rules are often dataset-specific with no good "global" rules to cover diverse data. Our observation is that we can again leverage reference table L to "learn" such negative rules — if a pair of records in the L table only differ by one pair of words, then we learn a *negative rule* from that pair. The learned negative rules can then be used to prevent false positives in joining L and R.

Definition 3.1. *Let $l_1, l_2 \in L$ be two reference records, $W(l_1)$ and $W(l_2)$ be the set of words in the two records, respectively. Denote by $\Delta_{12} = W(l_1) \setminus W(l_2)$, and $\Delta_{21} = W(l_2) \setminus W(l_1)$. We learn a* negative rule *$NR(\Delta_{12}, \Delta_{21})$, if $|\Delta_{12}|$=1 and $|\Delta_{21}|$=1.*

Note that since L is a reference table with little or no duplicates, the negative rules we learned intuitively capture different "identifiers" for different entities of the same entity type.

We summarize the algorithm for learning and applying negative rules in Algorithm 2. The inputs are the L-L and L-R tuple pairs that survive in the blocking step. The tuples will be first preprocessed by lowercasing, stemming and removing punctuations (Line 1). The algorithm will then learn negative rules from L-L tuple pairs (Line 2 to Line 6). Then it applies the learned negative rules on L-R tuple pairs (Line 7 to Line 11), where the tuple pairs that meet the negative rules will be discarded and will not be joined.

While negative-rule learning can be applied broadly regardless of whether fuzzy-joins are auto-tuned or not, in the context of Auto-FuzzyJoin our experiments show that it provides an automated way to improve join quality on top of automated parameter tuning.

4 MULTI-COLUMN AUTO-FUZZYJOIN

We now consider the more general case, where the join key is given as multiple columns, or when the join key is not explicitly given, in which case our algorithm has to consider all columns.

L-id	L-name	L-director	L-description
l_1	Carrie	Brian De Palma	Carrie White is shy and outcast ...
l_2	Vibes	Ken Kwapis	Psychics hired to find lost temple...
l_3	...	...	...

R-id	R-name	R-director	R-description
r_1	Carrie	Brian DePalma	This classic horror movie based ...
r_2	Vibes	Ken Kwapis	Two hapless psychics unwittingly...
r_3	...	...	...

Figure 5: Example multi-column fuzzy join: Movies.

Figure 5 shows an example of two movie tables with attribute like names, directors, etc. Intuitively, we can see that names and directors are important for fuzzy-join, but not descriptions. Users may either select name and director as key columns for Auto-FuzzyJoin, or may provide no input to the algorithm. In either case, the algorithm has to figure out what columns to use and their relative "importance" in making overall fuzzy-join decisions.

4.1 Multi-Column Join Configuration

Given that multiple columns may have different relative "importance", we extend single-column configuration $C = \langle f, \theta \rangle$ as follows. We define a *join function vector* as $\mathbf{F} = (f^1, f^2, ...f^m)$, where $f^j \in \mathcal{F}$ is the join function used for the j^{th} column pair. In addition, we define a *column-weight vector* as $\mathbf{w} = (w^1, w^2, ..., w^m)$, where $w^i \in [0, 1]$ is the weight associated with j^{th} column pair.

Let $l[j]$ and $r[j]$ be the value in j^{th} column of record l and r, respectively. Given $\mathbf{F}$ and $\mathbf{w}$, the distance between l and r is computed as the sum of weighted distances from all columns:

$$\mathbf{F}_{\mathbf{w}}(l, r) = \sum_{j=1}^{m} w^j f^j(l[j], r[j])$$

Definition 4.1. *A multi-column join configuration is a 3-tuple $\langle \mathbf{F}, \mathbf{w}, \theta \rangle$, where $\mathbf{F} \in \mathcal{F}^m$ is a join function vector, $\mathbf{w} \in \mathbb{R}^m$ is a column-weight vector, and $\theta \in \mathbb{R}$ is a threshold.*

Let $S = \{\langle \mathbf{F}, \mathbf{w}, \theta \rangle | \ \mathbf{F} \in \mathcal{F}^m, \mathbf{w} \in \mathbb{R}^m, \theta \in \mathbb{R}\}$ be the space of possible multi-column join configurations. A multi-column join configuration $C \in S$ induces a fuzzy join mapping $J_C(r)$ for each $r \in R$, defined as:

$$J_C(r) = \underset{l \in L, \mathbf{F}_{\mathbf{w}}(l,r) \le \theta}{\arg\min} \mathbf{F}_{\mathbf{w}}(l, r), \forall r \in R \quad (15)$$

4.2 Multi-Column AutoFJ

Given the space of multi-column configurations S, the Auto-FuzzyJoin problem is essentially the same as RM-FJ in the single-column setting: we want to find a set of configuration $U \in 2^S$ that maximizes the $recall(U)$, while having $precision(U) \ge \tau$.

A naive approach is to invoke the single-column fuzzy join solution in Algorithm 1 with the multi-column join configuration space S. However, such a simple adaptation is not practical, because the new multi-column search space is exponential in the number columns m (each column has its own space of fuzzy-join configurations, which can combine freely with configurations from other columns). Exploring this space naively would be too slow.

Our key observations here is that (1) given a wide table, there are often only a few columns that contribute positively to the overall fuzzy-join decisions; (2) the relative "importance" of these useful columns is often a static property, which depends only on the data and task at hand, and is independent of the search algorithm used. For example, in Figure 5, the fact that column "names" is the most important, "directors" is less important, and "description" is not useful, would hold true irrespective of the distance-functions used and/or the set of input columns considered.

We therefore propose a multi-column AutoFJ algorithm shown in Algorithm 3, which is inspired by the forward selection approach

Algorithm 3 AutoFJ for multiple columns

Require: Tables L and R, precision target τ, and search space S
1: $U \leftarrow \emptyset, U^* \leftarrow \emptyset$ $\mathbf{w} \leftarrow (0, 0, ...0)$,
2: $E \leftarrow \{\mathbf{e_1}, \mathbf{e_2}, ...\mathbf{e_m}\}$, where $\mathbf{e_j}$ is a m-dimensional vector with the j^{th} position set to 1 and the rest to 0.
3: **while** $E \neq \emptyset$ **do**
4: **for all** $\mathbf{e_j} \in E$ **do**
5: **for all** $\alpha \in \{\frac{1}{g}, \frac{2}{g}, \ldots, \frac{g-1}{g}\}$ **do**
6: $\mathbf{w'} \leftarrow (1-\alpha)\mathbf{w} + \alpha\mathbf{e_i}$
7: $U' \leftarrow$ invoke Algorithm 1 with weight vector $\mathbf{w'}$.
8: **if** $recall(U') > recall(U^*)$ **then**
9: $U^* \leftarrow U', \mathbf{w}^* \leftarrow \mathbf{w'}, \mathbf{e}^* \leftarrow \mathbf{e_j}$
10: **if** $recall(U^*) > recall(U)$ **then**
11: $U \leftarrow U^*, \mathbf{w} \leftarrow \mathbf{w}^*, E = E \setminus \mathbf{e}^*$
12: **else**
13: **break**
14: **return** U

to feature selection in machine learning [17]. At a high-level, our algorithm starts from an empty set of join column (Line 1), and iteratively expands this set by adding the most important column from the remaining columns (Line 4 to Line 9). The importance of a candidate column is determined by the resulting join quality after adding it, which can be estimated using techniques from Section 3.1 (Line 7 to Line 9). The algorithm terminates when the join quality cannot be improved by adding an extra column (Line 13) or there is no remaining columns (Line 3). This adding one-column-at-a-time approach is reminiscent of forward selection [17].

In addition, as the set of candidate columns expands, instead of searching for the column-weight vector $\mathbf{w}$ blindly (which would again be exponential in m), we leverage the fact that column importance is a static property of the data set (Observation (2) above), and thus in each iteration we "inherit" column-weights from previous iterations, and further scale them linearly relative to the observed importance of the new column added in this iteration (Line 6).

Complexity of Algorithm 3. The search algorithm in Algorithm 3 invokes single-column AutoFJ $O(m^2 g)$ times (where m is the number of input columns, and g the discretization steps for weights), which is substantially better than the naive $O(g^m)$ we started with. Hence, the time complexity is $O(m^2 g(|S||L|\sqrt{|L|} + |S||R|\sqrt{|L|}))$. Its space complexity is $O(m(|S||L|\sqrt{|L|} + |S||R|\sqrt{|L|}))$ since we need to precompute distances for all m columns. In practice, we observe that it terminates after a few iterations (only selecting a few columns from a wide table). This, together with other optimizations we propose, makes multi-column AutoFJ very efficient.

5 EXPERIMENTS

We evaluate the effectiveness, efficiency, and robustness of fuzzy-join algorithms. All experiments are performed on a machine with two Intel Xeon E5-2673 v4 CPUs at 2.30GHz and 256GB RAM.

5.1 Single-Column Auto-FuzzyJoin

5.1.1 Datasets. We constructed 50 diverse fuzzy-join datasets using DBPedia [34]. Specifically, we obtained multiple snapshots of DBPedia[9] (from year 2013, 2014, 2015, 2016, etc.), which are harvested from snapshots of Wikipedia over time. Each entity in a DBPedia snapshot has a unique "`entity-id`", an "`entity-name`" (from Wikipedia article titles), and an "`entity-type`" (e.g., `Political Parties`, `Soccer Leagues`, `NCAA teams`, `Politicians`, etc., which are extracted from Wikipedia info-boxes). Because these entity-names are edited by volunteers, their names can have minor changes over time (e.g., "`2012 Wisconsin Badgers football team`" and "`2012 Wisconsin Badgers football season`" are the titles/entity-names used in two different snapshots, referring to the same Wikipedia article/entity because they share the same unique "`entity-id`" across time).

For each DBPedia snapshot from a specific year, and for each entity-type, we build a table with names of all entities in that type (e.g., `NCAA-Teams` from the snapshot in year 2013). Two tables of the same type from different years can then be used as a fuzzy-join task (e.g., `NCAA-Teams` in year 2013 vs. 2016). Because the `entity-id` of these entities do not change over time, it allows us to automatically generate fuzzy-join ground-truth using `entity-id`.

We randomly select 50 entity-types for benchmarking. We use the 2013 snapshot as L, and use the union of all other snapshots as R, which would create difficult cases where multiple right records join with the same left record, as well as cases where a right record has no corresponding left record. We further remove equi-joins from all datasets that are trivial for fuzzy joins. These 50 data sets and their sizes are shown in the leftmost two columns of Table 2. We released this benchmark together with our Auto-FuzzyJoin code on GitHub[10] to facilitate future research.

5.1.2 Evaluation Metrics. We report quality of Fuzzy-Join algorithms, using the standard *precision* (P) and *recall* (R) metrics, defined in Equation (3) and (4) of Section 2.

Recall that AutoFJ automatically produces a solution that maximizes recall while meeting a certain precision target. In comparison, existing fuzz-join approaches usually output (their own version of) similarity/probability scores for each tuple pair, and ask users to pick the right threshold. In order to compare, for each existing method, we search for the similarity (probability) threshold that would produce a precision score that is "closest to but not greater than" AutoFJ, and report the corresponding recall score (which favors baselines). We call this recall score *adjusted recall* (AR).

For example, suppose AutoFJ produces results with precision 0.91, recall 0.72. Suppose an existing baseline produces the following (P, R) values at different threshold-levels: $\{(0.8, 0.8), (0.9, 0.7), (0.92, 0.6), (0.95, 0.5)\}$. The adjusted recall (AR) for this baseline will be reported as 0.7, for its corresponding precision (0.9) is "closest to but not greater than" the 0.91 precision produced by AutoFJ. We can see that this reported AR clearly favors the baseline, but allows us to compare recall at a fixed precision target.

In addition to the AR, we also measure the quality of fuzzy-joins using Precision-Recall AUC score (PR-AUC), defined as the entire area under the Precision-Recall curves. This is a standard metric that does not require the thresholding procedure above.

5.1.3 Single-Column Fuzzy Join Algorithms.

• AutoFJ. This is our method, and we use target precision $\tau = 0.9$, the step size for discretizing numeric parameters $s = 50$. Table 1 lists the parameter values we used in experiments (c.f. Figure 2). In total, we consider 4 options for preprocessing, 2 for tokenization and 2 for token weights. For distance function, we consider 2 character-based distance, 8 set-based distance and 1 embedding distance [11].

[9] http://downloads.dbpedia.org/

[10] https://github.com/chu-data-lab/AutomaticFuzzyJoin

[11] https://github.com/explosion/spacy-models/releases//tag/en_core_web_lg-2.3.0

Parameters		Values
Preprocessing		L, L+S, L+RP, L+S+RP
Tokenization		3G, SP
Token Weights		EW, IDFW
Distance Function	Character-based	JW, ED
	Set-based	JD, CD, MD, DD, ID
		*Contain-Jaccard
		*Contain-Cosine
		*Contain-Dice Distance
	Embedding	GED

* We design three hybrid distance functions named Contain-Jaccard, Contain-Cosine and Contain-Dice. If two records have containment relationship (i.e. $r \subseteq l$), they are equivalent to the standard distance functions; Otherwise, output 1.

Table 1: Parameter Options Considered in the Experiments

Among the 8 set-based functions, the first 5 of them are standard functions; while the last 3 are hybrid ones we added. In total we have $4 \times 2 + 4 \times 2 \times 2 \times 8 + 4 \times 1 = 140$ join functions (note that the tokenization and token-weight parameters are only applicable to set-based distance).

• Best Static Join Function (BSJ). In this method, we evaluate the join quality of each individual join function from the space of 140 discussed above. We compute the Adjusted-Recall (AR) score of each join function on each data set, and report the join function that has the best average AR over 50 datasets. This can be seen as the best static join function, whereas AutoFJ produces dynamic join functions (different datasets can use different join functions).

• Excel. This is the fuzzy-join feature in Excel [12]. The default parameter setting is carefully engineered and uses a weighted combination of multiple distance functions.

• FuzzyWuzzy (FW). This is a popular open-source fuzzy join package with 5K+ stars [13]. It produces a score for every tuple pair based on an adapted and fine-tuned version of the edit distance.

• ZeroER [48]. This is a recent unsupervised entity resolution (ER) approach that requires zero labeled examples. It uses a generative model that is a variant of a Gaussian Mixture Model to predict the probability of a tuple pair being a match. The features used in ZeroER are generated by the Magellan [32] package.

• ECM [24]: This is an unsupervised approach with the Fellegi and Sunter framework [29]. We use the implementation from [25] that uses binary features and Expectation-Conditional Maximization (ECM) algorithm. The features are generated by the Magellan [32] package and binarized using the mean value as the threshold.

• PPJoin (PP) [49]: This is a set similarity join algorithm that employs several filtering techniques to optimize efficiency. We use an existing implementation [14] and use Jaccard similarity.

• Magellan [32]. This is a supervised approach that uses conventional ML models based on similarity values as features. We use the open-source implementation with random forest as the model. For each dataset, we randomly split the data into a training and a test set by 50%-50%. Note that 50% training data is generous given that the amount of available labeled data is usually much smaller in practice. The reported AR are the average results over 5 runs.

•DeepMatcher (DM) [40]. This is a supervised approach that uses a deep learning model with learned record embedding as features. We use the same setup as Magellan in terms of train/test split. We use the open-source implementation with its default model.

[12]https://www.microsoft.com/en-us/download/details.aspx?id=15011

[13]https://github.com/seatgeek/fuzzywuzzy

[14]https://github.com/usc-isi-i2/ppjoin

• Active Learning (AL). This is an active learning based supervised approach. The algorithm interactively queries users to label new tuple pairs until 50% joined pairs in the data are labeled. We use the implementation from modAL [21] with default query strategy, and we use the same model and features as Magellan.

• Upper Bound of Recall (UBR). There are many ground-truth pairs in L and R that are difficult for fuzzy-joins (e.g., ("Lita (wrestler)", "Amy Dumas"), ("GLYX-13", "Rapastinel"), etc.). These pairs have semantic relationships that are out of the scope of fuzzy-joins. To test the true upper-bound of fuzzy-joins, for each r we find its closest $l \in L$ using *all* possible configurations $C \in S$, which collectively is the set of fuzzy-join pairs that can be produced. We call a ground-truth pair (l, r) feasible if it is in the set, and report the recall using all feasible ground-truth pairs. This gives us a true upper-bound of fuzzy-join on these data sets.

5.1.4 *Single-Column Fuzzy Join Evaluation Results.*

Overall Quality Comparison. Table 2 shows the overall quality comparison between AutoFJ and other approaches on 50 datasets. The average precision of AutoFJ is 0.886, which is very close to the target precision $\tau = 0.9$. We compute the Pearson correlation coefficient between the actual precision and the estimated precision (PEPCC) over AutoFJ iterations for each dataset. As we can see in Table 2, the average PEPCC over all datasets is 0.894, which shows that the actual/estimated precision match well across iterations.

The average recall of AutoFJ is 0.624. Given that the average recall upper bound (UBR) is 0.834, AutoFJ produces about 75% of correct joins that can possibly be generated by *any* fuzzy-join program. As we can see, AutoFJ outperforms all other approaches on 21 out of 50 datasets. On average, the recall of AutoFJ is 0.062 better than Excel, the best among all unsupervised approaches, and 0.129 better than AL, the best among all supervised approaches that use 50% of joins as training data. To test the statistical significance of this comparison, We perform an upper-tailed T-Test over the 50 datasets, where the null hypothesis (H_0) states that the mean of AutoFJ's recall is no better than that of a baseline's AR. As shown in the second last row of Table 2, the p-values of all baselines are smaller than 0.003, showing that the differences are significant.

The last row of Table 2 shows the average PR-AUC scores of AutoFJ and other methods over 50 datasets. As we can see, the PR-AUC of AutoFJ is on average 0.057 better than Excel, the strongest unsupervised method, and 0.056 better than Magellan, the method with the highest PR-AUC score among all supervised methods. This indicates that AutoFJ can outperform other baselines across different precision levels. The details of PR-AUC scores on each dataset can be found at the full version of this paper [9], where we show that AutoFJ outperforms all other methods in terms of PR-AUC on 28 out of 50 datasets.

Among all unsupervised baselines, Excel, as a commercial-grade tool that features carefully engineered weighted combination of multiple distance functions, performs the best. In fact, Excel is even better than BestStaticJF, the best statistic configuration tuned on the 50 datasets. We also observe that FW and ZeroER has generally worse performance than Excel and BestStaticJF, because FW and ZeroER use predetermined sets of similarity functions while Excel and BestStaticJF have various degrees of feature engineering. ECM and PPJoin under-perform other unsupervised methods,

Dataset	Size (L-R)	UBR	AutoFJ				Unsupervised						Supervised			Ablation Study	
							BSJ	Excel	FW	ZeroER	ECM	PP	Magellan	DM	AL	AutoFJ-UC	AutoFJ-NR
			PEPCC	RERCC	P	R	AR	AR	AR	AR	AR	AR	AR	AR	AR	AR	AR
Amphibian	3663 - 1161	0.605	0.942	0.954	0.797	0.537	0.388	0.514	0.513	0.504	0.372	0.485	0.786	0.588	**0.861**	0.511	0.533
ArtificialSatellite	1801 - 72	0.75	0.91	0.986	0.761	**0.486**	0.264	0.375	0.403	0.042	0.194	0.125	0.199	0.011	0.142	0.278	0.486
Artwork	3112 - 245	0.967	0.753	0.993	0.907	0.837	0.755	**0.89**	0.731	0.592	0.371	0.518	0.691	0.354	0.715	0.841	0.873
Award	3380 - 384	0.753	0.986	0.993	0.917	**0.43**	0.331	0.393	0.365	0.115	0.237	0.201	0.209	0.092	0.165	0.367	0.372
BasketballTeam	928 - 166	0.867	0.942	0.993	0.873	0.62	0.554	**0.711**	0.018	0.042	0.398	0.331	0.247	0.089	0.379	0.53	0.681
Case	2474 - 380	0.997	0.936	0.966	0.987	0.976	0.584	0.853	0.763	0.584	0.529	0.166	0.803	0.809	**0.983**	0.958	0.976
ChristianBishop	5363 - 494	0.933	0.952	0.993	0.931	**0.789**	0.662	0.652	0.603	0.407	0.283	0.5	0.649	0.313	0.756	0.713	0.802
CAR	2547 - 190	0.947	0.829	0.992	0.925	0.842	0.711	**0.895**	0.421	0.095	0.221	0.389	0.449	0.135	0.408	0.805	0.842
Country	2791 - 291	0.821	0.969	0.996	0.898	**0.608**	0.471	0.546	0.464	0.241	0.244	0.254	0.29	0.068	0.403	0.423	0.577
Device	6933 - 658	0.878	0.969	0.999	0.93	**0.664**	0.553	0.657	0.477	0.222	0.198	0.295	0.106	0.14	0.298	0.584	0.658
Drug	5356 - 157	0.535	0.96	0.993	0.731	0.363	0.134	0.401	0.376	0.408	0.045	0.07	**0.595**	0.008	0.541	0.293	0.427
Election	6565 - 727	0.872	0.976	0.993	0.926	**0.651**	0.501	0.318	0.162	0.073	0.177	0.11	**0.651**	0.418	0.342	0.55	0.362
Enzyme	3917 - 48	0.813	0.625	0.970	0.775	**0.646**	0.5	0.604	0.583	0.5	0.208	0.5	0.321	0.033	0.318	0.646	0.667
EthnicGroup	4317 - 946	0.938	0.933	0.932	0.958	0.803	0.551	0.765	0.513	0.463	0.225	0.015	0.726	0.464	**0.876**	0.729	0.776
FootballLeagueSeason	4457 - 280	0.871	0.945	0.794	0.878	0.614	0.532	0.65	0.575	0.468	0.132	0.282	**0.882**	0.201	0.437	0.571	0.582
FootballMatch	1999 - 53	0.906	0.958	0.987	1	**0.755**	0.472	0.321	0.34	0.415	0.208	0.623	0.715	0.052	0.466	0.472	0.66
Galaxy	555 - 17	0.529	0.912	1.000	0.714	0.294	0.353	**0.412**	0.118	0.059	0.235	0.235	0.319	0.044	0.217	0.412	0.294
GivenName	3021 - 154	0.994	0.39	0.174	0.973	**0.922**	0.831	0.857	0.078	0.013	0.442	0.286	0.565	0.06	0.886	0.909	0.922
GovernmentAgency	3977 - 571	0.839	0.965	0.998	0.902	**0.627**	0.531	0.623	0.469	0.336	0.261	0.343	0.386	0.41	0.467	0.543	0.611
HistoricBuilding	5064 - 512	0.924	0.958	0.985	0.939	**0.785**	0.654	0.768	0.664	0.416	0.236	0.066	0.537	0.284	0.603	0.656	0.795
Hospital	2424 - 257	0.79	0.961	0.999	0.854	**0.568**	0.475	0.451	0.444	0.136	0.292	0.23	0.191	0.141	0.145	0.49	0.626
Legislature	1314 - 216	0.917	0.908	0.986	0.925	0.801	0.736	**0.819**	0.708	0.509	0.208	0.023	0.66	0.328	0.748	0.75	0.796
Magazine	4005 - 274	0.942	0.849	0.976	0.942	**0.825**	0.741	0.788	0.42	0.179	0.281	0.318	0.123	0.286	0.423	0.755	0.847
MemberOfParliament	5774 - 503	0.972	0.975	0.995	0.949	0.704	0.571	0.147	0.308	0.018	0.205	0.008	0.63	0.251	**0.742**	0.569	0.688
Monarch	2033 - 242	0.917	0.972	0.998	0.902	**0.649**	0.355	0.645	0.306	0.236	0.351	0.095	0.328	0.101	0.454	0.322	0.612
MotorsportSeason	1465 - 388	0.93	0.973	-0.158	0.971	0.874	0.902	0.912	0.827	0.912	0.196	0.912	0.98	0.959	**0.994**	0.905	0.933
Museum	3982 - 305	0.8	0.956	0.997	0.889	**0.58**	0.521	0.58	0.374	0.246	0.193	0.193	0.14	0.11	0.227	0.528	0.633
NCAATeamSeason	5619 - 34	1	NA*	NA*	1	0.412	0.382	0.059	0.588	0.412	0.118	0.294	**0.928**	0.059	0.503	0.824	0.382
NFLS	3003 - 10	1	NA*	NA*	1	0.5	**1**	0.5	0.5	0.5	0.2	**1**	0.933	0	0.633	0.4	0.5
NaturalEvent	970 - 51	0.882	0.815	0.968	0.811	**0.588**	**0.588**	0.333	0.49	0.275	0.412	0.118	0.1	0.241	0.054	0.549	0.588
Noble	3609 - 364	0.915	0.979	0.997	0.936	0.445	0.393	**0.646**	0.426	0.234	0.363	0.033	0.125	0.065	0.443	0.365	0.308
PoliticalParty	5254 - 495	0.76	0.986	0.997	0.819	0.402	0.327	**0.467**	0.408	0.228	0.204	0.069	0.264	0.071	0.309	0.331	0.362
Race	2382 - 175	0.571	0.985	0.990	0.766	0.337	0.269	**0.349**	0.206	0.143	0.194	0.16	0.217	0.034	0.103	0.291	0.309
RailwayLine	2189 - 298	0.836	0.967	0.998	0.877	0.55	0.393	**0.597**	0.117	0.091	0.285	0.289	0.234	0.061	0.325	0.487	0.52
Reptile	797 - 819	0.979	0.849	0.918	0.757	0.966	0.84	0.941	0.932	0.925	0.527	0.893	0.969	0.94	**0.985**	0.938	0.964
RugbyLeague	418 - 58	0.828	0.91	0.994	0.933	**0.483**	0.414	0.224	0.259	0.052	0.276	0.259	0.139	0.221	0.145	0.293	0.483
ShoppingMall	223 - 227	1	0.872	0.994	0.771	0.824	**0.95**	0.887	0.642	0.063	0.509	0.547	0.653	0.446	0.699	0.931	0.862
SoccerClubSeason	1197 - 51	0.98	-0.075	0.921	0.97	0.627	**0.98**	0.922	0.549	0.941	0.275	0.961	0.825	0.331	0.668	0.98	0.627
SoccerLeague	1315 - 238	0.622	0.932	0.992	0.757	**0.433**	0.294	0.387	0.282	0.235	0.168	0.197	0.199	0.076	0.103	0.357	0.471
SoccerTournament	2714 - 290	0.945	0.978	0.885	0.961	0.762	0.666	0.517	0.597	0.4	0.176	0.11	0.764	0.339	**0.797**	0.728	0.672
Song	5726 - 440	0.984	0.862	0.993	0.971	0.916	0.759	0.87	0.445	0.227	0.28	0.327	0.848	0.543	**0.954**	0.875	0.911
SportFacility	6392-672	0.607	0.99	0.999	0.867	**0.418**	0.323	0.378	0.357	0.216	0.201	0.146	0.103	0.043	0.21	0.327	0.396
SportsLeague	3106 - 481	0.638	0.955	0.993	0.738	0.38	0.337	**0.418**	0.289	0.191	0.179	0.214	0.13	0.104	0.139	0.351	0.339
Stadium	5105 - 619	0.591	0.992	0.999	0.854	**0.396**	0.307	0.367	0.339	0.21	0.2	0.139	0.186	0.096	0.281	0.318	0.354
TelevisionStation	6752 - 1152	0.711	0.991	1.000	0.874	0.495	0.486	0.174	0.146	0.048	0.154	0.044	0.385	0.064	**0.638**	0.47	0.451
TennisTournament	324 - 27	0.889	0.619	0.956	0.944	0.63	0.593	0.444	0.556	0.37	0.556	0.519	**0.674**	0.257	0.433	0.593	0.556
Tournament	4858 - 459	0.832	0.983	0.996	0.894	0.606	0.556	0.366	0.468	0.275	0.207	0.431	**0.657**	0.183	0.606	0.503	0.527
UnitOfWork	2483 - 380	0.995	0.952	0.958	0.984	**0.974**	0.811	0.887	0.763	0.618	0.55	0.434	0.825	0.9	**0.974**	0.966	0.974
Venue	4079 - 384	0.737	0.973	0.997	0.885	0.56	0.466	**0.568**	0.49	0.391	0.214	0.133	0.497	0.086	0.423	0.526	0.56
Wrestler	3150 - 464	0.412	0.986	0.996	0.774	0.265	0.203	0.248	0.222	0.006	0.164	0.08	**0.409**	0.091	0.317	0.244	0.265
Average		0.834	0.894	0.938	0.886	**0.624**	0.539	0.562	0.442	0.306	0.267	0.299	0.485	0.24	0.495	0.575	0.608
Significant Test P-value							3e-5	3e-3	3e-9	2e-13	6e-21	3e-12	9e-5	2e-20	5e-6		
Average PR-AUC				**0.715**			0.647	0.658	0.481	0.419	0.156	0.459	0.659	0.411	0.521		

*The correlation coefficients are NA because the algorithm terminates with one iteration.

Table 2: Performance evaluation on 50 single-column fuzzy join datasets.

because ECM binarizes features and lose information, while PPJoin uses vanilla Jaccard similarity.

Among all supervised baselines that use 50% all joins as training data, AL achieves the best result based on AR as it carefully selects which examples to include in the training set. The deep model DM performs poorly, which is not entirely surprisingly as deep learning approaches typically require a large number of labeled examples to perform well.

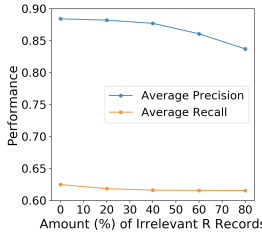
It is also worth highlighting that AutoFJ (and Excel) outperforms the best supervised baseline even when 50% of all ground-truth labels are used as training data.

Ablation Study (1): Contribution of Union of Configurations. To study the benefit of using a set of configurations, we compare AutoFJ with AutoFJ-UC that only uses one single best configuration. Note that the single configuration selected by AutoFJ-UC can be different for each dataset. The column AutoFJ-UC in Table 2 shows the quality of the best single configuration on each dataset. The average adjusted recall is 0.575, which is 0.049 lower than AutoFJ, but still higher than all other methods. This suggests that (1) dynamically using a single configuration is better than using any static configuration; and (2) dynamically selecting a union of configurations can further boost the performance.

Ablation Study (2): Contribution of Negative Rules. The column AutoFJ-NR in Table 2 shows the AR results of AutoFJ without negative rules. As we can see, without negative-rules, the average AR decreases to 0.608, which shows the benefit of negative-rules.

Robustness Test (1): Adding Irrelevant Records to the Right Table. We construct an adversarial test to evaluate the robustness of AutoFJ as follows. For each dataset, we insert irrelevant records to the R by randomly picking records from other 49 datasets. Figure 6(a) shows the average precision and recall over 50 datasets with different amounts of irrelevant records added. As we can see, even when 80% of records in the R are irrelevant, AutoFJ can still achieve an average precision of around 84% with recall almost unaffected.

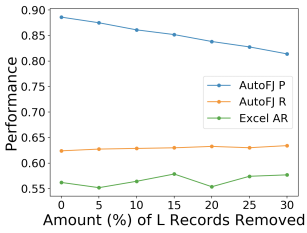
Robustness Test (2): Zero Fuzzy Joins. We construct a second adversarial test, where the L and R are taken from different entity-type that are completely unrelated (e.g., L from "`Satellites`" joins with R from "`Hospitals`"), such that any joins produced are false positives. We construct 10 such cases. Figure 6(b) shows the false positive rate (defined as the number of false positives divided by



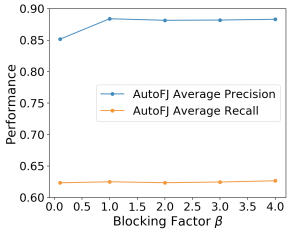
(a) Irrelevant R Records

	False Positive Rate	
Synthetic Datasets (L-R)	**AutoFJ**	**Excel**
Song-Reptile	0.73%	45.18%
Satellite-Hospital	0.78%	15.56%
EthnicGroup-Wrestler	1.29%	43.10%
TelevisionStation-Galaxy	0.00%	58.82%
Award-Drug	0.00%	8.28%
CAR-Artwork	4.08%	31.43%
SportsLeague-Stadium	1.13%	40.71%
RugbyLeague-Tournament	0.65%	33.55%
BasketballTeam-Monarch	0.00%	2.48%
TennisTournament-Legislature	0.00%	2.31%

(b) Zero Fuzzy Joins



(c) L Incompleteness



(d) Sensitivity to Blocking

Figure 6: (a, b, c): Single-Column robustness tests. (d): sensitivity to blocking.

the number of records in R) of AutoFJ and Excel, the best baseline. In all cases, the false positive rate of AutoFJ is below 5% and much smaller than Excel.

Robustness Test (3): L Incompleteness. In this work, we do not assume the reference table L to be complete, and we take this into account when estimating precision (c.f. Equation (9)). However, an extremely sparse L can affect our estimation. To test its robustness, we make the already incomplete L even more sparse by randomly removing records in L. Figure 6(c) shows the average performance of AutoFJ and Excel across 50 datasets with different amounts of records removed from L tables. As expected, the average precision decreases as L table becomes more and more sparse. However, even with 30% L records removed, AutoFJ can still achieve precision of 0.81. In all cases, the recall of AutoFJ is still at least 0.051 higher than Excel.

Sensitivity to Blocking. Figure 6(d) shows the average performance on 50 datasets varying the blocking factor β, where $\beta \times \sqrt{|L|}$ is the number of left records kept for each right record. A smaller β gives faster algorithms, but potentially at the cost of join quality. As we can see, after β exceeds 1.0 (e.g., we keep top $1.0 \times \sqrt{100} = 10$ records for each right record if $|L| = 100$), the performance of AutoFJ remains almost unchanged even if we increase β further.

Varying Target Precision. Figure 7(a) shows the average precision and recall on 50 datasets, as we vary the precision target τ. As τ decreases, the average precision of AutoFJ decreases accordingly. Note that the two align very well (the correlation-coefficient of the two is 0.9939), suggesting that our precision estimation works as intended. Compared to other baseline methods, our method remain the best as we vary the target, and our algorithm consistently outperforms Excel, the strongest baseline, by at least 0.062.

Efficiency Analysis. Overall, AutoFJ finishes 15/50 data sets in 30 seconds, 33/50 in 1 minute, and 49/50 in 130 seconds. To compare the running time of AutoFJ with other methods, we bucketize 50 datasets into 5 groups based on the size of $|L| \times |R|$. Figure 7(b) shows the average running time of AutoFJ and other methods over datasets in each group. As we can see, the running time of AutoFJ is comparable to other methods. PPJoin is the fastest method since it employs an efficient version of Jaccard similarity. DM is on average 10 times slower than other methods because it needs to train deep neural networks. AutoFJ is on average 2-3 times slower than ECM and Excel, but faster than ZeroER, Magellan and FW. AutoFJ is 2-3 times faster than AL.

Varying Configuration Spaces. We run AutoFJ using a varying number of configurations from the space listed in Table 1. The reduced configuration space is achieved by removing some options for the 4 parameters. For example, if we only use L and L+S+RP for pre-processing instead of all four options, the space reduces to 70 from 140. Figure 7(c) shows the average performance of AutoFJ over 50 datasets with different size of the configuration space. As we can see, the average precision is almost unchanged as we vary the space size, showing the accuracy of our precision estimation. The average recall decreases slightly with a smaller number of configurations, because the expressiveness of fuzzy-matching is reduced accordingly. We compute the AR of Excel and Magellan using the precision of AutoFJ with different configuration space. As we can see, even with 24 configurations, the recall of AutoFJ is still 0.036 higher than the AR of Excel and 0.105 higher than Magellan.

Figure 7(d) shows the running time of each component of AutoFJ as we vary the configuration space. As we can see, the running time is greatly reduced as the configuration space shrinks. With 24 configurations, the algorithm becomes 2 times faster than using 140 configurations. Also, as we can see in Figure 7(d), the pre-computation for precision takes less than 10% of the overall time. In contrast, if we compute this repeatedly at every iteration (e.g., with 140 configurations, there are about 45 iterations on average for each dataset), our overall running time can be 6x slower (with this component taking 85% time).

5.2 Multi-Column Auto-FuzzyJoin

5.2.1 Multi-Column Datasets. For multi-column fuzzy joins, we use 8 benchmark datasets in the entity resolution literature [32, 33, 40], as shown in Table 3.

Dataset	Domain	#Attr.	Size (L-R)	#Matches
Fodors-Zagats (FZ) [7]	Restaurant	6	533 - 331	112
DBLP-ACM (DA) [2]	Citation	4	2,616 - 2,294	2,224
Abt-Buy (AB) [2]	Product	3	1,081 - 1,092	1,097
Rotten Tomatoes-IMDB (RI) [22]	Movie	10	7,390 - 556	190
BeerAdvo-RateBeer (BR) [22]	Beer	4	4,345 - 270	68
Amazon-Barnes & Noble (ABN) [22]	Book	11	3,506 - 354	232
iTunes-Amazon Music (IA) [22]	Music	8	6,907 - 484	132
Babies'R'Us-BuyBuyBaby (BB) [22]	Baby Product	16	10,718 - 289	109

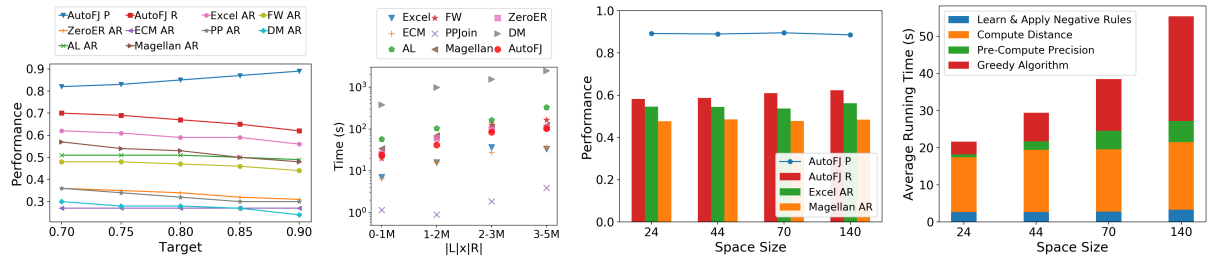
Table 3: Multi-column fuzzy join datasets.

5.2.2 Multi-Column Fuzzy Join Algorithms.

- AutoFJ. This is our proposed Algorithm 3, using precision target $\tau = 0.9$, discretization steps $s = 50$, and the column-weight search steps $g = 10$. Given 140 join functions, and a table with m columns, we can in theory have as many as 140^m configurations. In our experiments, we add an additional constraint that distance functions considered in the same configuration should be the same across all columns. This is for efficiency considerations, but nevertheless produces fuzzy-joins with state-of-the-art quality. To handle missing values in the datasets, we treat missing values as empty strings, and assign maximum distances when comparing two missing values.
- Excel, FW, ZeroER, ECM and PP, Magellan, DM, AL. These are the same methods as we described in Section 5.1.3. Since Excel, FW and PP handle all columns in the same way, we invoke these methods with all columns concatenated.

5.2.3 Multi-Column Fuzzy Join Evaluation Results.

Overall Quality Comparison. Table 4(a) shows the overall quality comparison between AutoFJ and other methods on multi-column datasets. As we can see, AutoFJ remains the best method



(a) Vary Target Precision (b) Running Time Comparison (c) Vary Space Size (d) Vary Space Size

Figure 7: (a): Varying Target Precision, (b): Efficiency Comparison, (c, d): Varying Configuration Space Sizes.

Dataset	Column Selected	Weight Selected	AutoFJ		Unsupervised					Supervised		
			P	R	Excel	FW	ZeroER	ECM	PP	Magellan	DM	AL
RI	name, director	0.9, 0.1	0.955	0.995	0.805	0.947	**1.000**	0.895	0.332	0.990	0.594	**1.000**
AB	name	1	0.957	**0.451**	0.035	0.015	0.045	0.213	0.018	0.035	0.111	0.255
BB	title, company struct	0.6, 0.4	0.688	**0.713**	0.426	0.370	0.019	0.537	0.130	0.418	0.227	0.541
BR	beer name, factory name	0.9, 0.1	0.909	0.882	0.824	0.721	0.515	0.824	0.765	0.574	0.572	**0.967**
ABN	title, pages	0.8, 0.2	0.8	0.983	0.966	0.901	0.957	0.987	0.948	0.796	0.812	**1.000**
DA	title, year	0.8, 0.2	0.967	0.987	0.978	0.692	0.942	0.108	0.980	0.985	0.966	**1.000**
FZ	phone, class	0.1, 0.9	0.8	**1**	**1.000**	0.857	0.929	0.179	0.929	**1.000**	0.896	**1.000**
IA	song name, genre	0.7, 0.3	0.967	0.853	0.794	0.265	0.824	0.824	0.618	0.944	0.323	**0.988**
Average			0.880	**0.858**	0.728	0.596	0.654	0.571	0.590	0.718	0.563	0.844
P-value					0.024	0.003	0.029	0.028	0.011	0.034	0.001	0.369
Average PR-AUC			0.847		0.785	0.583	0.676	0.487	0.744	**0.879**	0.729	0.864

(a) Overall Multi-Column Join Quality Comparison

Dataset	AutoFJ	Excel	AL
	ΔR	ΔAR	ΔAR
FZ	0	-0.018	0
DA	0	-0.018	-0.001
AB	0	-0.01	-0.066
RI	0	-0.079	0
BR	0	-0.015	-0.093
ABN	0	0.004	0
IA	0	-0.176	-0.024
BB	0	-0.343	-0.041
Average	0	-0.082	-0.028

(b) Multi-Column Robustness

Table 4: Multi-Column Fuzzy Join Evaluations.

on average in the multi-column joins. The recall of AutoFJ on average is 0.13 better than Excel, the strongest unsupervised baseline, and 0.014 better than AL, the strongest supervised method. AutoFJ outperforms all other methods on 3 out of 8 datasets and achieves comparable results to the best baseline on the remaining datasets. We also perform upper-tailed T-Test to verify the statistical significance of our results. As shown in the second to the last row of Table 4, with the exception of AL, the p-values for all other baselines are smaller than 0.034.

The last row of Table 4(a) shows the average PR-AUC of AutoFJ and other methods. As we can see, AutoFJ significantly outperforms all other unsupervised methods and achieve comparable performance compared to supervised methods such as Magellan and AL that uses 50% joins as training data. The PR-AUC on each dataset can be found in the full version [9] of this paper.

Effectiveness of Column Selection. Table 4(a) reports the columns selected by AutoFJ and their corresponding weights. Observe that the selected columns are indeed informative attributes, such as Name and Director in Rotten Tomatoes-IMDB (RI) dataset (with Name being more important). Also note that AutoFJ is able to achieve these results typically using only one or two columns.

Robustness Test: Adding Random Columns. We test the robustness of AutoFJ on multi-column joins by adding adversarial columns with randomly-generated strings in both L and R tables. The length of each random string is between 10-50. Table 4(b) shows the change of performance of AutoFJ, Excel and AL, after adding random columns. Since random columns do not provide any useful information, they are not selected by AutoFJ, and hence have no effect on our results. In contrast, as Excel and AL use all input columns, adding random columns does affect their results.

6 RELATED WORK

Fuzzy join, also known as entity resolution and similarity join, is a long-standing problem in data integration [27, 28], with a long line of research on improving the *scalability* of fuzzy-join algorithms (e.g., [11, 12, 14, 20, 23, 26, 36, 39, 42, 43, 45, 46, 50]).

Existing state-of-the-art in optimizing join *quality* are predominantly supervised methods (e.g., Magellan [32] and DeepMatcher [40]), which require labeled data of matches/non-matches to be provided before classifiers can be trained. In contrast, our proposed Auto-FuzzyJoin is unsupervised and mainly leverages structural properties of reference tables, Surprisingly, this unsupervised approach outperforms supervised methods even when 50% of ground-truth labels are used as training data.

Among unsupervised methods, our evaluation suggests that the carefully-tuned Excel (with default settings) is a strong baseline. It employs a variant of the generalized fuzzy similarity [18], which is a weighted combination of multiple distance functions. The weight functions, as well as pre-processing parameters, were carefully-tuned on English data.

Other entity-matching approaches include AutoEM [51] and Ditto [37], which uses pre-trained entity-type-models and language-models for entity-matching, respectively. ZeroER [48] is a recent unsupervised method that uses a predetermined set of features and Gaussian Mixture Model to determine matches.

Additional methods to facilitate complex table joins include methods that leverage search engines [16, 31, 35], and program-synthesis [47, 52].

7 CONCLUSIONS

In this paper, we propose an unsupervised Auto-FuzzyJoin to auto-program fuzzy joins without using labeled examples. We formalized this as an optimization problem that maximizes recall under a given precision constraint. Our results suggest that this unsupervised method is competitive even against state-of-the-art supervised methods. We believe unsupervised fuzzy entity matching is an interesting area that is still under-studied, and clearly worth attention from the research community.

REFERENCES

[1] [n.d.]. Alteryx: Fuzzy Match Documentation. https://help.alteryx.com/2018.2/FuzzyMatch.htm.

[2] [n.d.]. Benchmark datasets for entity resolution. https://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution.

[3] [n.d.]. Excel: Fuzzy Lookup Add-In. https://www.microsoft.com/en-us/download/details.aspx?id=15011. ([n. d.]).

[4] [n.d.]. Fuzzy Lookup in SQL Server. https://docs.microsoft.com/en-us/sql/integration-services/data-flow/transformations/fuzzy-lookup-transformation.

[5] [n.d.]. OpenRefine Fuzzy Reconciliation. https://github.com/OpenRefine/OpenRefine/wiki/Reconciliation.

[6] [n.d.]. Python string match library: py_stringmatching. http://anhaidgroup.github.io/py_stringmatching/v0.4.1/Tutorial.html.

[7] 2019.7.12. Duplicate Detection, Record Linkage, and Identity Uncertainty: Datasets. http://www.cs.utexas.edu/users/ml/riddle/data.html.

[8] 2019.7.12. Fuzzy Join in Power Query. https://support.microsoft.com/en-us/office/fuzzy-match-support-for-get-transform-power-query-ffdd5082-c0c8-4c8e-a794-bd3962b90649.

[9] 2020.07.06. Supplemental materials for AutoFJ, with a anonymous URL. https://www.dropbox.com/sh/myiees5wv716n2f/AAAPx5AVDxt4kXJoKXnskSxa?dl=0.

[10] Ittai Abraham, Yair Bartal, and Ofer Neimany. 2006. Advances in metric embedding theory. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*. 271–286.

[11] Foto N. Afrati, Anish Das Sarma, David Menestrina, Aditya G. Parameswaran, and Jeffrey D. Ullman. 2012. Fuzzy Joins Using MapReduce. In *Proceedings of ICDE*.

[12] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. 2006. Efficient Exact Set-Similarity Joins. In *Proceedings of VLDB*.

[13] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*. Vol. 463. ACM press New York.

[14] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. 2007. Scaling up all pairs similarity search. In *Proceedings of WWW*.

[15] Mikhail Bilenko, Beena Kamath, and Raymond J Mooney. 2006. Adaptive blocking: Learning to scale up record linkage. In *Sixth International Conference on Data Mining (ICDM'06)*. IEEE, 87–96.

[16] Christian Bizer. 2014. Search Joins with the Web.. In *ICDT*. 3.

[17] Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. 2018. Feature selection in machine learning: A new perspective. *Neurocomputing* 300 (2018), 70–79.

[18] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. 2003. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 313–324.

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

[20] Xu Chu, Ihab F Ilyas, and Paraschos Koutris. 2016. Distributed data deduplication. *Proceedings of the VLDB Endowment* 9, 11 (2016), 864–875.

[21] Tivadar Danka and Peter Horvath. 2018. modAL: A modular active learning framework for Python. *arXiv preprint arXiv:1805.00979* (2018).

[22] Sanjib Das, AnHai Doan, Paul Suganthan G. C., Chaitanya Gokhale, Pradap Konda, Yash Govind, and Derek Paulsen. 2019.07.12. The Magellan Data Repository. https://sites.google.com/site/anhaidgroup/projects/data.

[23] Akash Das Sarma, Yeye He, and Surajit Chaudhuri. 2014. Clusterjoin: A similarity joins framework using map-reduce. *Proceedings of the VLDB Endowment* 7, 12 (2014), 1059–1070.

[24] Jonathan De Bruin. 2015. Probabilistic record linkage with the Fellegi and Sunter framework: Using probabilistic record linkage to link privacy preserved police and hospital road accident records. (2015).

[25] J De Bruin. 2019. *Python Record Linkage Toolkit: A toolkit for record linkage and duplicate detection in Python*. https://doi.org/10.5281/zenodo.3559043

[26] Dong Deng, Guoliang Li, Shuang Hao, Jiannan Wang, and Jianhua Feng. 2013. MassJoin: A MapReduce-based Algorithm for String Similarity Joins. In *Proceedings of ICDE*.

[27] AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of data integration*. Elsevier.

[28] Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. 2007. Duplicate Record Detection: A Survey. *IEEETKDE* 19, 1 (2007), 1–16.

[29] Ivan P Fellegi and Alan B Sunter. 1969. A theory for record linkage. *J. Amer. Statist. Assoc.* 64, 328 (1969), 1183–1210.

[30] MT Hajiaghayi, K Jain, K Konwar, LC Lau, II Mandoiu, A Russell, A Shvartsman, and VV Vazirani. 2006. The minimum k-colored subgraph problem in haplotyping and DNA primer selection. In *Proceedings of the International Workshop on Bioinformatics Research and Applications (IWBRA)*. Citeseer, 1–12.

[31] Yeye He, Kris Ganjam, and Xu Chu. 2015. Sema-join: joining semantically-related tables using big table corpora. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1358–1369.

[32] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. 2016. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1197–1208.

[33] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 484–493.

[34] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. 2015. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195.

[35] Oliver Lehmberg, Dominique Ritze, Petar Ristoski, Robert Meusel, Heiko Paulheim, and Christian Bizer. 2015. The mannheim search join engine. *Journal of Web Semantics* 35 (2015), 159–166.

[36] Guoliang Li, Dong Deng, Jiannan Wang, and Jianhua Feng. 2011. PASS-JOIN: A Partition-based Method for Similarity Joins. In *Proceedings of VLDB*.

[37] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2021. Deep entity matching with pre-trained language models. *VLDB 2021* (2021).

[38] Charles T Meadow, Donald H Kraft, and Bert R Boyce. 1999. *Text information retrieval systems*. Academic Press, Inc.

[39] Ahmed Metwally and Christos Faloutsos. 2012. V-SMART-Join: A Scalable MapReduce Framework for All-Pair Similarity Joins of Multisets and Vectors. In *Proceedings of VLDB*.

[40] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 19–34.

[41] George Papadakis, Jonathan Svirsky, Avigdor Gal, and Themis Palpanas. 2016. Comparative analysis of approximate blocking techniques for entity resolution. *Proceedings of the VLDB Endowment* 9, 9 (2016), 684–695.

[42] Yasin N Silva, Walid G Aref, and Mohamed H Ali. 2010. The similarity join database operator. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. IEEE, 892–903.

[43] Yasin N Silva and Jason M Reed. 2012. Exploiting mapreduce-based similarity joins. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 693–696.

[44] Mohammad Karim Sohrabi and Hosseion Azgomi. 2017. Parallel set similarity join on big data based on locality-sensitive hashing. *Science of computer programming* 145 (2017), 1–12.

[45] Rares Vernica, Michael J Carey, and Chen Li. 2010. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 495–506.

[46] Jiannan Wang, Guoliang Li, and Jianhua Feng. 2012. Can we beat the prefix filtering? An adaptive framework for similarity join and search. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 85–96.

[47] Robert H Warren and Frank Wm Tompa. 2006. Multi-column substring matching for database schema translation. In *Proceedings of the 32nd international conference on Very large data bases*. Citeseer, 331–342.

[48] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. ZeroER: Entity Resolution using Zero Labeled Examples. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1149–1164.

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

[50] Minghe Yu, Guoliang Li, Dong Deng, and Jianhua Feng. 2016. String similarity search and join: a survey. *Frontiers of Computer Science* 10, 3 (2016), 399–417.

[51] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end Fuzzy Entity-Matching using Pre-trained Deep Models and Transfer Learning. In *The World Wide Web Conference*. 2413–2424.

[52] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1034–1045.