Auto-FuzzyJoin: Auto-Tune Fuzzy Joins Without Labeled Examples

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

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

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 join is an important database operator that is widely used in practice. So far the research community has focused exclusively on the important problem of efficient fuzzy join algorithms. Today, practitioners also struggle with a daunting space of parameters to optimize join quality (e.g., distance-function, distance-thresholds, tokenization-options, etc.), where a manual trial-and-error approach has become a major pain-point. This important parameter-tuning problem has thus far received surprisingly little attention from the research community.

Motivated by the observation, we study the problem of "auto-tune" suitable fuzzy-join parameters on given input tables, without requiring explicit human input such as labelled training data.

We develop an Auto-FuzzyJoin framework that, when given two tables L and R, and a user-specified target precision τ (say 0.9), can automatically configure fuzzy-join parameters meeting the precision target τ in expectation, while maximizing fuzzy-join recall (defined as the number of correctly joined records). We study the hardness of this problem, and develop effective algorithms that are extensively tested on both existing benchmarks as well as a new benchmark of 50 fuzzy-join tasks we created from Wikipedia data. Our evaluation suggests that the proposed Auto-FuzzyJoin significantly outperforms existing unsupervised approaches (e.g., Microsoft Excel and ZeroER). It is surprisingly competitive even against supervised approaches (e.g., Magellan and DeepMatcher) when 50% of ground-truth joins are used as training data. We will release our code and benchmark to facilitate future research.

ACM Reference Format:
Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. 2021. Auto-FuzzyJoin: Auto-Tune Fuzzy Joins Without Labeled Examples. In *Proceedings of 2021 International Conference on Management of Data (SIGMOD '21).* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Both authors contributed equally to this research.

SIGMOD '21, June 20–25, 2021, Xi'an, Shaanxi, China

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Fuzzy-join, also known as similarity-join, fuzzy-match, and entity resolution, is an important database operator that takes two tables L and R, and produces record pairs that approximately match in 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 fuzzy-join scalability (e.g., [9, 10, 13, 20, 23, 26, 35, 37]). We have witnessed a fruitful line of research and remarkable progress in this area, leading to wide adoption of fuzzy-join technologies as features in popular commercial software that can successfully scale to large tables (e.g., in Alteryx [1], Trifacta [6], and Microsoft Excel [3]).

The need to parameterize fuzzy-joins. While the scalability of fuzzy-join has been greatly improved, we argue that the usability of fuzzy-join has now become the main pain-point. Specifically, given the need to optimize for different types of data sets, today's fuzzy-join implementations offer rich configuration options with a puzzling number of parameters, many of which need to be carefully tuned before fuzzy-joins can function properly.

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. Out of the 19 options, 11 are binary (can be either `true` or `false`), which however already correspond to a total of $2^{11} = 2048$ configurations that are difficult to tune manually. Similarly `py_stringmatching`, a popular open-source fuzzy-join package, boasts 92 options. Note that the numbers above have not yet included parameters from numeric domains – options like "similarity threshold" and "containment bias" can take any value from $[0, 1]$.

Not surprisingly, we have seen recurring user questions on a variety of user forums on how to set parameters appropriately, including how to set similarity-thresholds[1], token-weights[2],

[1] https://www.reddit.com/r/excel/comments/9y6o6a/how_is_similarity_threshold_calculated_when_doing/
[2] https://answers.microsoft.com/en-us/msoffice/forum/all/token-weights-for-fuzzy-lookup-add-in-for-excel/c9c4a0f3-014f-4e2e-8672-b2303cfe3a4d

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 basebal 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 1: A sample of fuzzy-join parameters.

distance-functions[3], multi-column settings[4], etc. These parameters are widely used in the literature [9, 10, 13, 23, 26, 35, 37], which can be broadly classified into four categories: Pre-processing, Tokenization, Token-weights, and Distance-functions, as shown in Figure 1 (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 (often still requiring many trial-and-errors); less-technical users (e.g., those in Excel or Tableau) would likely never become fuzzy-join masters, and as a result have to live with the sub-optimal default-settings. We argue that parameter-tuning has become a significant pain point, and a major roadblock to wider adoption of fuzzy-join. Therefore, in this paper, we explore the possibility of automatically selecting fuzzy-join parameters from a rich parameter space, based solely on inferred characteristics of input tables, without requiring any forms of inputs from human users (e.g., labeled training examples for match vs. non-match). Our unsupervised approach relies on the assumption that one of the input table is a curated master table (also known as "reference table") with no duplicates (e.g., in data-warehousing settings [17]). We note that the notion of reference tables is widely used in the literature (e.g., [17, 18, 40]), and adopted by commercial systems (e.g., SQL Server [4], OpenRefine/GoogleRefine [5], Excel [3], etc.). This assumption gives us unique signals to infer whether a particular fuzzy-join is good or not in the absence of labeled data.

Key Insights. We use examples to illustrate our key insights. On the left of Figure 2(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 ideally we should tokenize them 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 be joined, but their token overlap is

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

[4]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 small Edit-distance. (l_4, r_4) are joined based on Jaccard-distance.

Figure 2: Examples of Fuzzy Join Cases.

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$(l, r) < 3$). We clarify that we are not trying to come up with a joining rule for each pair of matches; rather, we will search for configurations that can join as many pairs as possible while retaining high precision.

Union-of-Configurations. Because different types of string variations are often present simultaneously (e.g., typos vs. extraneous tokens), ideally a fuzzy-join package 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, tuning such a union of configurations manually is even more intractable than a single configuration. Our approach can nevertheless find optimized configurations automatically from this rich configuration space.

Negative-rule-learning. We note that similarity functions and scores alone are often not sufficient for high-quality fuzzy joins. For instance, existing solutions would join (l_6, r_6) and (l_7, r_7) in Figure 2(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 L and finding out many pairs like "2008 LSU Tigers baseball team" ≠ "2008 LST Tigers football team" (given that both are in the reference table L and they are unlikely to be duplicates), to further deduce rules of the form "baseball" ≠ "football", thus preventing (l_6, r_6) from being joined (and similarly another negative rule "2007" ≠ "2008", so that (l_7, r_7) are not joined).

Leverage Reference-Tables. We additionally leverage intrinsic properties of reference-tables L, to automatically reason what fuzzy-join boundaries are "safe" (with no false-positive joins). In Figure 2(b) for instance, unlike Figure 2(a), even a seemingly small Edit-distance$(l, r) = 1$ is not "safe" on this data set, and would produce many false-positives like (l_1, r_1), (l_5, r_1), (l_2, r_2), (l_6, r_2), etc., none of which are correct joins. While it may be obvious to humans when looking at the results, 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. We note that elements in L can be similar, as long as they correspond to distinct entities. The fact that r_1 joins with *both* l_1 and l_5 would intuitively suggest $l_1 \approx r_1 \approx l_5$, contradicting with $l_1 \neq l_5$ implied by the reference table assumption (similar analysis can be performed on other pairs like (l_2, r_2)). We generalize this transitivity-like idea more broadly, so that the algorithm can decide that Edit-distance$(l, r) = 1$ is not "safe" for this data set and should not be used.

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. Also note that 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 academic literature (e.g., [17, 18, 40]) and commercial fuzzy-join systems (e.g., Excel [3], SQL Server [4], OpenRefine/GoogleRefine [5], etc.). Our problem is also essentially entity resolution where one table has few or no duplicate. In practice, we find most entity resolution benchmark data sets used in the literature to have at least one table meeting the reference table definition, for which our approach is applicable. For example, in the well-known repository of ER data sets compiled by Prof. AnHai Doan's group [5], we find the assumption that one table meeting the reference table definition holds completely on 19/29 datasets, and holds approximately (satisfied by over 95% rows) on 26/29 datasets. We will report an extensive evaluation using these existing data sets in the experiment section.

The reference table assumption essentially serves as additional constraint to prevent our algorithm from using fuzzy-join configurations that are too "loose" (or join more than what is correct). In Appendix A we construct a concrete example to show why some forms of constraints are necessary for unsupervised fuzzy-joins (or otherwise the problem may be under-specified).

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 1 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 [9, 10, 13, 23, 26, 35, 37]. 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 1. Applying this f to l_1, r_1 in Figure 2(a), we can compute $f(l_1, r_1) = 0.2$. Additional examples of score computation are shown in the last column of Figure 1.*

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

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 1.

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:

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

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 2(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), Jaccard-distance is high (0.5) because of spelling variations in a few tokens, for which Edit-distance is more appropriate.

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 [36]). We note that this definition of recall in absolute terms simplifies our analysis, which is no different from the relative recall [12], 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:

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

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

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

Theorem 2.1. *The RM-FJ problem is NP-hard.*

Proof. We prove the hardness of RM-FJ using a reduction from the densest k-subhypergraph (DkH) problem [30], which is known to be NP-hard. Recall that in DkH, we are given a hypergraph $H(V, E)$, and the task is to pick a set of k vertices such that the sub-hypergraph induced has the maximum number of hyper-edges. This optimization problem can be solved with a polynomial number ($|E|$) of its decision problem: given $H(V, E)$ and k, decide whether there exists a subset of vertices $E' \subset E$ that induces a sub-hypergraph $H'(V', E')$, with $|E'| \leq k$ and $|V'| \geq p, \forall p \in \{1, \ldots, |E|\}$.

We show that the decision version of DkH can be reduced to a decision version of the RM-FJ problems as follows. Give an instance of the DkH problem with $H(V, E)$, a size budget k, and a profit target p, we construct a decision version of the RM-FJ. Map each hyper-edge $e \in E$ in DkH to a configuration C_e in RM-FJ, and each vertex $v \in V$ incident on e to a false-positive fuzzy-join pair associated with $C(e)$. Finally let each $C(e)$ has unit profit of 1 (true-positive fuzzy-join pairs). It can be seen that each instance of DkH translates directly to a corresponding RM-FJ problem, by setting $S = \{C(e) | e \in E\}$, and $\tau = \frac{p}{p+k}$. If we were able to solve RM-FJ optimally in polynomial time, we would have in turn solved DkH, contradicting the hardness result of DkH [30]. Therefore, the hardness of RM-FJ is NP-hard. □

Discussions. Our problem formation of maximizing recall for a given target precision is widely-used in prior ER research (e.g. [11, 14]). It is also a scenario explicitly used by popular software packages like Magellan (Section 3.1 of Magellan [32]). However, the alternative formulation of maximizing precision given a recall target is equally interesting, and deserves future research.

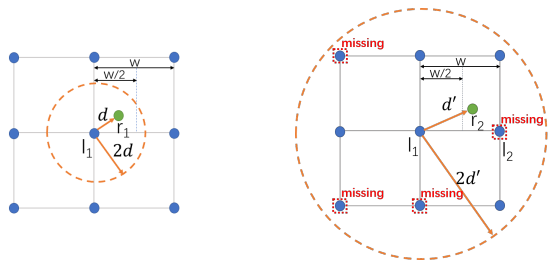
3 SINGLE-COLUMN AUTO-FUZZYJOIN

We now discuss Auto-FuzzyJoin when the join key is a pair of single columns (we will discuss multi-columns settings later).

3.1 Estimate Precision and Recall

In the RM-FJ formulation above, the hardness results assume 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 or labeled examples to directly compute these measures. To solve RM-FJ, we first need a way to estimate the precision and recall of given configurations using L and R only, so that we can instantiate Equation (5) and (6) in this optimization problem. The challenge is to do so without labels. While this may seem infeasible in general, we show that it is possible in our specific context of fuzzy-joins, by leveraging the reference table L. For simplicity, 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 precision/recall of



(a) l_1 is the closest left record to r_1. We say (l_1, r_1) is a "good" join, because no other l exists in the ball of distance $2d$.

(b) l_1 is the closest left record to r_2, given that l_2 is missing from L. We estimate (l_1, r_2) as an "unreliable" join because we find many left records in the ball of $2d'$.

Figure 3: Estimate the goodness of a join pair (l, r) using join function f. We first compute distance $f(l, r) = d$, then draw a ball of distance $2d$ centered around l. The more records in the $2d$-ball, the more unreliable the join is.

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 every $r \in R$ has a matched tuple $l \in L$. We call that "L" is *complete* relative to R in this case. Note that this is not a requirement of our approach – it is used only to assist conceptual reasoning.

Intuitively, records in a table can be viewed as embedded in a multi-dimensional space under the transformation of distance function f. Assuming that L is complete, in the "small neighborhood" of each $l \in L$, we will likely find neighboring records of l with similar distances to l, which is a property we call "*local homogeneity*". For instance, most l in Figure 2(a) have close "neighbors" that differ from l by only one or two tokens (which translates to a Jaccard-distance of around 0.2). Similarly in Figure 2(b), most l have neighbors that differ by one character (or Edit-distance around 1).

We can visualize the local neighbors of each l as points on unit-grids in a multi-dimensional space, when projected using distances given by f, as shown in Figure 3 in a 2-D visualization. In Figure 3, reference records l are visualized as blue points on the grid, with similar distance to close neighbors. Intuitively, we can think of reference records as "stars" in an astronomy analogy, whereas the query records $r \in R$ (with string variations) can be thought of as "planets" orbiting around "stars" (at different distances). Determining which l should this r join amounts to finding the closest l (star), under the transformation of some suitable distance function f.

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 3(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, there are often $r \in R$ that should not be joined with any $l \in L$. In our visualized model, this would lead to missing blue points on the grid.

For example, in Figure 3(b), l_2 is a reference record that should join with r_2, but can be missing in L. This creates difficulties in the context of Auto-FuzzyJoin, because given l_2 is absent in L, the closest left record to r_2 becomes l_1, and a naive approach would attempt to fuzzy-join (r_2, l_1) using a distance of $d = f(r_2, l_1)$. The

key question we try to address, is how to infer (without ground-truth) that this particular d is too lax of a distance to use, and (r_2, l_1) is a "bad" join pair given that L is potentially incomplete.

Recall that when L is complete, the close neighbors of an l tend to have similar distances to l. Intuitively we can compute this "distance-between-neighbors" in the neighborhood of l, referred to as $w(l)$ or the width of the grid, as illustrated in Figure 3. By triangular-inequality, we show that a join pair (l, r) with distance $d = f(l, r)$ is "safe", if d is smaller than $\frac{w(l)}{2}$, half of the grid-width.

This can be performed in the other direction – namely given a pair (l, r), 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 left records would fall within this ball. We can be confident that the join is "reliable" if no other left records fall within the ball. The confidence goes down if we find other records in the $2d$-ball.

Example 3.1. *In Figure 3(a), to join r_1, we draw this $2d$ ball around l_1 which is the closest to r_1, and find no other l's, indicating that this d is a small and "safe" distance to use for fuzzy-joins based on the structure of l's local neighborhood.*

In Figure 3(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 suggests that this join-distance d' is too "lax" in this local neighborhood, and we should thus not join r_2 with l_1 even if l_1 is the closest.

Note that, as is often the case in practice, the unit-grid in Figure 3(b) is incomplete. In this case we have 4 missing l records (marked by dotted rectangles). This incomplete L, however, does not prevent us from concluding that joining (l_1, r_2) is risky. 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 to be an incorrect 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 multiplicative-inverse because all l within the 2d-ball are reasonably close to r based on the structure of the local neighborhood, and we cannot be sure which l is the best to join, so we assume each l has equal chance of being correct.

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

Example 3.2. *The precision of (l_1, r_1) in Figure 3(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 3(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 2(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 $\frac{1}{8}$).

We now rewrite Equation (8) 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 $FP(U) = \sum_{C \in U} FP(C)$.

The more complex scenario is when the join assignment of some r has conflicts, 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 one that is a more confident prediction and discard the other conflicting join. $TP(U)$, $FP(U)$ can be updated accordingly.

Given $TP(U)$ and $FP(U)$, the estimated precision and 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: $max\_profit \leftarrow 0$
8: **for all** $C \in S \setminus U$ **do**
9: **if** $profit(U \cup \{C\}) > max\_profit$ **then**
10: $C^* \leftarrow C, max\_profit \leftarrow profit(U \cup \{C\})$
11: **if** $precision(U \cup \{C^*\}) > \tau$ **then**
12: $U \leftarrow U \cup \{C^*\}$
13: **else**
14: **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). If there are multiple configurations with the same profit at an iteration, which rarely happens on large datasets, we randomly pick one to break ties. 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, 39] 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 perform extensive empirical testing and find a very good default strategy. We use lowercase for preprocessing, 3-gram for tokenization, and IDF as token weighting, since they are the most commonly used choices in our experiments. For the similarity function parameter, it is more complicated since we need to design a similarity function that can be computed efficiently (without actually performing quadratic comparisons) and can effectively identify the tuple pairs that are unlikely to be matches. *Overlap blocker* is one of the blocking techniques used in Magellan [32]. It uses the number of tokens shared by two records as the similarity function, which can be computed efficiently using inverted index. Inspired by this method, we also use the shared tokens to compute the similarity, but instead of counting the number of tokens, we compute the sum of IDF weights of shared tokens. This can avoid including tuple pairs that share many meaningless frequent tokens, such as "the", "an", which have lower IDF weights. The similarity threshold affects the number of tuple pairs after blocking, but we would not expect users to tune the threshold. Therefore, instead of using a fixed similarity threshold, for every record in R, we simply retain the top-K ($K = \sqrt{|L|}$) records in L with the highest similarity scores to reflect our intuition that K is a larger L can be harder to join and hence we need to retain more record pairs to avoid losing on recall after blocking. We acknowledge that our blocking method may not be the best one. It is also interesting to investigate the "auto-tuned" blocking method as a direction for future research.

Complexity of Algorithm 1. Since the number of 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 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 2(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 (obtained by space tokenization) 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.

Complexity of Algorithm 2. For each tuple pair, the space tokenization (Line 4 and Line 9) and computing word set difference (Line 5 and Line 10) are in O(1) time. With our default blocking, the number of $L-L$ and $L-R$ tuple pairs are $O(|L|\sqrt{|L|})$ and

Algorithm 2 Learning and Applying Negative Rules

Require: Tables L and R, LL and LR
Apply lowercasing, stemming, and removing punctuation for all L and R
$NR \leftarrow \emptyset$
for $l_1, l_2 \in LL$ **do**
  $W_1 \leftarrow$ set of words of l_1, $W_2 \leftarrow$ set of words of l_2
  $\Delta_1 \leftarrow |W_1 \setminus W_2|, \Delta_2 \leftarrow |W_2 \setminus W_1|$
  if $|\Delta_1| = 1$ and $|\Delta_2| = 1$ **then**
    $NR \leftarrow NR \cup (\Delta_1, \Delta_2)$
for $l, r \in LR$ **do**
  $W_1 \leftarrow$ set of words of l, $W_2 \leftarrow$ set of words of r
  $\Delta_1 \leftarrow |W_1 \setminus W_2|, \Delta_2 \leftarrow |W_2 \setminus W_1|$
  if $|\Delta_1| = 1$ and $|\Delta_2| = 1$ and $(\Delta_1, \Delta_2) \in NR$ **then**
    Remove (l, r) from LR
return LR

$O(|R|\sqrt{|L|})$, respectively. Therefore, the total complexity of Algorithm 2 is $O(|L|\sqrt{|L|} + |R|\sqrt{|L|})$.

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 column is given as multiple columns, or when the join column is not explicitly given, in which case our algorithm has to consider all columns.

Figure 4 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.

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 4: Example multi-column fuzzy join: Movies.

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:

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

Algorithm 3 AutoFJ for multiple columns

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

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 $precision(U) \geq \tau$.

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 4, 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, inspired by the forward selection approach well-known in machine learning for feature-selection [16]. At a high-level, our algorithm starts from an empty set of join column, and iteratively expands this set by adding the most important column from the remaining columns (where 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). This adding one-column-at-a-time approach is reminiscent of forward selection [16].

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. Algorithm 3 shows the above process using pseudo-code in detail.

We also note that Algorithm 3 can implicitly infer that columns with many missing values are less useful because missing values are not likely joined with other values (low recall), and hence will assign lower weights to those columns.

Complexity of Algorithm 3. The search algorithm in Algorithm 3 invokes single-column AutoFJ $O(m^2g)$ 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 $O(m^2g(|S||L|\sqrt{|L|} + |S||R|\sqrt{|L|}))$. 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[6] (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" vs. "2012 Wisconsin Badgers football season", in two different snapshots).

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

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, namely, those records in the right table that are exactly the same as some record in the left table. These 50 data sets and their sizes are shown in the leftmost two columns of Table 2. We will release this benchmark dataset, which has been included in this submission as supplemental materials [8].

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 scores or probability of matching for each tuple pairs, and ask users to pick the right similarity or probability threshold for the best precision/recall trade-off. To compare apple-to-apple, for each method compared, we perform an automated procedure to search for the similarity (probability) threshold that would produce a precision score closest to but not greater than AutoFJ. We report the corresponding recall score, which we call the *adjusted recall* (AR). We then compare the recall of AutoFJ and the AR of other methods. Note that, by construction, if the recall of AutoFJ is higher than the AR of other methods, it also indicates that AutoFJ has a higher F1-measure. However, if the recall of AutoFJ is smaller, it does not necessarily mean that AutoFJ has a lower F1-measure since its precision is higher. Hence, the AR score is favorable for baseline methods compared. When there are ties in precision (i.e., two thresholds produce the same precision), we report AR using the highest possible recall, again making it favorable for baselines.

For example, suppose AutoFJ produces fuzzy-join with precision 0.91, recall 0.72. For an existing method, we try all different thresholds to obtain different (P, R) values as $(0.8, 0.8)$, $(0.9, 0.7)$, $(0.92, 0.6)$, $(0.95, 0.5)$. The adjusted recall (AR) for this method will be 0.7, as its corresponding precision is closest to but not greater than the 0.91 produced by AutoFJ.

In addition to the adjusted recall, we also measure the quality of fuzzy-joins using Precision-Recall AUC score (PR-AUC). The PR-AUC is defined as the area under the Precision-Recall curves. A higher PR-AUC score generally indicates a higher recall (or F-measure) under different precision levels. For AutoFJ, we obtain the precision-recall curves by varying the target precision τ; for other baseline methods, we obtain the precision-recall curves by varying the similarity (probability) thresholds.

5.1.3 Single-Column Fuzzy Join Algorithms.

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

• 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 1). Though the configuration space is exponential in the number of parameters, in practice this is very small. This is because we only have a limited number of parameters and each parameter has only a limited number of choices; in addition, some parameter value combinations need not be enumerated (e.g., different tokenization choices have no impact if character-based distance functions are chosen). 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 [7]. 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).

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

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

• Best Static Join Function (BSJ). In this method, we evaluate the join quality of each join function in our space. 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 [8]. 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 [9]. It produces a score for every tuple pair based on an adapted and fine-tuned version of the edit distance.
• ZeroER [42]. This is a state-of-the-art unsupervised entity resolution (ER) approach that achieves comparable performance to supervised methods but requires zero labeled examples. It uses a generative model that is variant of a Gaussian Mixture Model to predict the probability of a tuple pair being a match. The features, i.e., similarity functions, used in ZeroER is automatically 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) [43]: This is a set similarity join algorithm that employs several filtering techniques to improve the efficiency. We use an implementation [10] that uses 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) [38]. 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.
• 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.

[8] https://www.microsoft.com/en-us/download/details.aspx?id=15011
[9] https://github.com/seatgeek/fuzzywuzzy
[10] https://github.com/usc-isi-i2/ppjoin

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* configuration. 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. We also perform upper-tailed T-Test over 50 datasets to verify the statistical significance of our results. The alternative hypothesis (H_1) assumes the mean of AutoFJ's recall is greater than the mean of baseline's AR. As shown in the second last row of Table 2, the p-values of all baselines are smaller than 0.003, which shows the significance of our results.

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 means that AutoFJ can generally outperforms other baseline methods under different precision levels besides the precision level of 0.9. The details of PR-AUC scores on each dataset can be found in Table 5 in Appendix B, where we show that AutoFJ outperforms all other methods 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 generally performs worse than other unsupervised methods. This is because ECM binarizes features, which can lose information. PPJoin simply joins records with Jaccard similarity, which may not be able to capture string variations, such as typos.

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 is unsatisfactory, which is not surprisingly as deep learning approaches usually require many labeled examples to perform well.

It is also worth noting that AutoFJ (and Excel) can even outperform the best supervised baseline with 50% all joins as training data, which we can attribute to two reasons. First, AutoFJ essentially performs a data-driven feature engineering process to select the best distance function, while existing supervised methods assume predetermined feature set. Second, the performance of supervised

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.

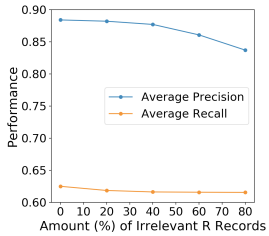
methods depends heavily on the amount of labeled examples, which is even more critical in deep learning methods like DM.

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 quality of AutoFJ without negative rules. We compute the adjusted recall (AR) of AutoFJ-NR by selecting a precision target for each dataset such that the resulting actual precision is closest to but not greater than AutoFJ. As we can see, without negative-rules, the average adjusted recall 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 test 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 5(a) shows the average precision and recall over 50 datasets with different amount 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.

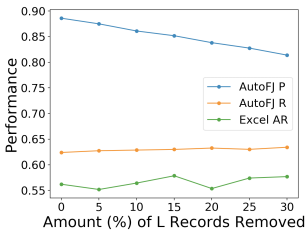
Robustness Test (2): Zero Fuzzy Joins. We construct a second adversarial test, where the L and R are taken from different entity-type that are entirely unrelated (e.g., L from "Satellites" joins with R from "Hospitals"), thus any joins are false positives. We construct



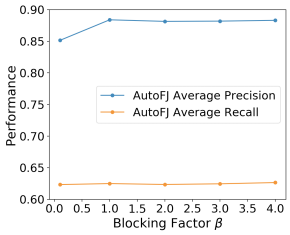
(a) Irrelevant R Records

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

10 such cases. Figure 5(b) shows the false positive rate (defined as the number of false positives divided by 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 that the reference table L is complete, and we took this into account when estimating precision (c.f. Equation (9)). However, an extremely sparse L does affect our estimate. To test the robustness of our estimate, we make the already incomplete L even more sparse by randomly removing records in L. Figure 5(c) shows the average performance of AutoFJ and Excel across 50 datasets with different amount of records removed from L tables. Note that the performance numbers are reported with respect to the ground-truth of a particular L and R after removing records. 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 5(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 6(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 6(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 usually 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 usually 2-3 times faster than AL, but in practice AL can take longer time because we do not include the time for human effort.

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 6(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 6(d) shows the running time of each component of AutoFJ as we vary the space size. 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 6(d), the pre-computation for precision only 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

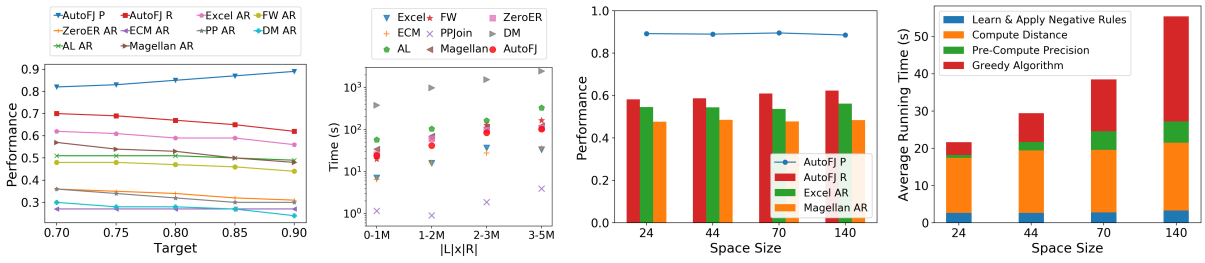
5.2.1 Multi-Column Datasets. For multi-column fuzzy joins, we use 8 benchmark datasets in the entity resolution literature [32, 33, 38], as shown in Table 3.

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



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

Figure 6: Varying Target Precision (a), Efficiency Comparison (b) and Varying Space Size (c)(d).

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
	R	AR	AR
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.

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 do not automatically infer join-keys, 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 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, except for AL, the p-values of all other baselines are smaller than 0.034, which shows the significance of our results.

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 Table 7 in Appendix B, where we show that AutoFJ outperforms other datasets on 2 out of 8 datasets.

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. We also observe that AutoFJ will not select or will assign lower weights to those columns with missing values. Among all the columns selected in Table 4, only the column "pages" in ABN dataset and the "company struct" column in BB dataset have few missing values, which have lower weights compared to other selected 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 fuzzy matching and entity resolution, is a long-standing problem in data integration [27, 28].

The state-of-the-art supervised methods typically first compute a similarity vector for every record pair, and then train a classifier to determine matches/non-matches [28, 38, 38]. Many similarity measures have been used, including domain independent ones such as edit distance [41] and Jaccard distance [19] and domain specific ones such as Jaro distance for person names [31]. Given a dataset, picking suitable similarity measures is often a decision implicitly made by humans. Our work, however, aims at selecting distance functions and thresholds automatically. In addition, supervised methods typically need labeled examples. For example, the state-of-the-art results are achieved using deep learning models [38] that are label-hungry. In contrast, our work requires no labeled examples.

AutoEM [44] is a transfer learning EM approach that uses pre-trained EM models to build ER systems. This method relies on pre-trained matchers for various entities types (e.g., names, organizations). For a newly given EM task, it still requires labeled training data to fine-tune the models when the given attribute types are not pre-trained. In contrast, AutoFJ does not need any domain knowledge or labeled training data.

In terms of unsupervised fuzzy-join methods, our evaluation shows that the carefully-tuned default setting in Excel significantly outperforms all other methods, and is the state-of-the-art. It employs a variant of the generalized fuzzy similarity [17], which is a weighted combination of multiple distance functions. The weight functions, as well as pre-processing parameters, were extensively tuned on English data. In comparison, our evaluation shows that other fuzzy-join solutions (e.g., FuzzyWuzzy) employ one default configuration and is often sub-optimal on different data sets.

ZeroER [42] is a recent unsupervised method that also uses a predetermined set of features. It uses a variant of Gaussian Mixture Model to decide matches, and is included in experimental evaluation.

7 CONCLUSIONS

In this paper, we propose to automatically select fuzzy join configurations without using labeled examples. We formalized this as an optimization problem of maximizing recall while meeting a given precision threshold, and developed efficient AutoFJ algorithms. We showed that AutoFJ significantly outperforms existing methods.

REFERENCES

[1] 2019.07.12. Alteryx: Fuzzy Match Documentation. https://help.alteryx.com/2018.2/FuzzyMatch.htm.
[2] 2019.07.12. Benchmark datasets for entity resolution. https://dbs.uni-leipzig.de/en/research/projects/object_matching/fever/benchmark_datasets_for_entity_resolution.
[3] 2019.07.12. Excel: Fuzzy Lookup Add-In. https://www.microsoft.com/en-us/download/details.aspx?id=15011. (2019.07.12).
[4] 2019.07.12. Fuzzy Lookup in SQL Server. https://docs.microsoft.com/en-us/sql/integration-services/data-flow/transformations/fuzzy-lookup-transformation.
[5] 2019.07.12. OpenRefine Fuzzy Reconciliation. https://github.com/OpenRefine/OpenRefine/wiki/Reconciliation.
[6] 2019.07.12. Trifacta: Join Panel. https://docs.trifacta.com/display/SS/Join+Panel.
[7] 2019.7.12. Duplicate Detection, Record Linkage, and Identity Uncertainty: Datasets. http://www.cs.utexas.edu/users/ml/riddle/data.html.
[8] 2020.07.06. Supplemental materials for AutoFJ, with a anonymous URL. https://www.dropbox.com/sh/myiees5wv716n2f/AAAPx5AVDxt4kXJoKXnskSxa?dl=0.
[9] Foto N. Afrati, Anish Das Sarma, David Menestrina, Aditya G. Parameswaran, and Jeffrey D. Ullman. 2012. Fuzzy Joins Using MapReduce. In *Proceedings of ICDE*.
[10] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. 2006. Efficient Exact Set-Similarity Joins. In *Proceedings of VLDB*.
[11] Arvind Arasu, Michaela Götz, and Raghav Kaushik. 2010. On active learning of record matching packages. In *SIGMOD*. 783–794.
[12] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. 1999. *Modern information retrieval*. Vol. 463. ACM press New York.
[13] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. 2007. Scaling up all pairs similarity search. In *Proceedings of WWW*.
[14] Kedar Bellare, Suresh Iyengar, Aditya G Parameswaran, and Vibhor Rastogi. 2012. Active sampling for entity matching. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1131–1139.
[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] Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. 2018. Feature selection in machine learning: A new perspective. *Neurocomputing* 300 (2018), 70–79.
[17] 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.
[18] 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.
[19] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. 2006. A Primitive Operator for Similarity Joins in Data Cleaning. In *Proceedings of ICDE*.
[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] Matthew A Jaro. 1980. *UNIMATCH, a record linkage system: users manual*. Bureau of the Census.
[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] Guoliang Li, Dong Deng, Jiannan Wang, and Jianhua Feng. 2011. PASS-JOIN: A Partition-based Method for Similarity Joins. In *Proceedings of VLDB*.
[36] Charles T Meadow, Donald H Kraft, and Bert R Boyce. 1999. *Text information retrieval systems*. Academic Press, Inc.
[37] 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*.
[38] 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.
[39] 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.
[40] 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.
[41] Jiannan Wang, Guoliang Li, and Jianhua Feng. 2010. Trie-Join: Efficient Trie-based String Similarity Joins with Edit-Distance Constraints. In *Proceedings of VLDB*.
[42] 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.
[43] 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.
[44] 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.

A AUTO-FJ: A NEGATIVE RESULT

In this paper we consider an unsupervised approach to Fuzzy-Join without using any labeled examples, but with the help of a structured constraint in the form of reference tables (i.e., the L table in our problem that has few or no duplicate records).

Our key observation is that for unsupervised fuzzy-joins, some form of constraints (in reference tables or otherwise) would be necessary, or otherwise the problem becomes under-specified for unsupervised approaches. To show why this is the case, we construct an adversarial example inspired by real fuzzy-join/entity-resolution applications.

Consider the fuzzy join task in Figure 7(a), where we are given a task of Auto-FuzzyJoin between L and R without any additional constraints. This pair of tables in Figure 7 have attributes like product names, colors, and capacity.

This task is under-specified, for which there are multiple possible "ground-truth" matches. To see why this is the case, suppose an exact match for r_1 is missing in L, and we want to determine whether r_1 should fuzzy-join with l_1 or l_2 or none at all. In the first possible-world we call W_1, it is reasonable to join r_1 with l_1, because there are scenarios in retail where product colors are important distinguishing features for customers, while disk capacities are considered minor variants that customers can select after clicking through. Conversely, in the second possible-world W_2, it is equally plausible to join r_1 with l_2, as there are scenarios where storage capacities are considered as important product features (since they determine prices), while colors as minor features. Finally in W_3, it is also possible that r_1 should not join with either l_1 or l_2, if in some applications one considers both color and capacity are important distinguishing features for products.

The ground-truth for the three equally plausible interpretations of the fuzzy join tasks are W1: $\{(r_1, \{l_1\})\}$, W2: $\{(r_1, \{l_2\})\}$, W3:$\{(r_1, \emptyset)\}$, respectively. Since we removed the reference table constraint, a right tuple can join with an arbitrary number of left tuples. On this small data set of $L = \{l_1, l_2, l_3\}$, and $R = \{r_1\}$, any deterministic algorithm A would produce a match decision $A(L, R) = \{(r_1, m(r_1)) | m(r_1) \in 2^L\}$, where $2^L = \{\emptyset, \{l_1\}, \{l_2\}, \{l_3\}, \{l_1, l_2\}, \{l_1, l_3\}, \{l_2, l_3\}, \{l_1, l_2, l_3\}\}$. Let $F(A(L, R), W)$ be the F1-score of output $A(L, R)$ evaluated against the ground-truth in W. It can be shown through enumeration, that given any possible $A(L, R) = \{(r_1, m(r_1)) | m(r_1) \in 2^L\}$, there exists a $W \in \{W_1, W_2, W_3\}$, such that $F(A(L, R)) = 0$ (because either precision is 0, or recall is 0).

We conclude that on this fuzzy join data set, no deterministic algorithm can have F1 score > 0, thus proving the negative result.

Now let us consider L to be a reference table. As shown in Figure 7(b), this constraint allows us to infer that neither (l_1, r_1) or (l_2, r_1) should join, without asking labeled data from users to clarify the intent. The reference table L constrains this task enough to allow only one interpretation – both colors and capacity are important features (implicit from L), so that neither (l_1, r_1) or (l_2, r_1) should join, and we converge to W_3 discussed above.

While reference table is simple to specify yet powerful in constraining the space of possible matches, we do not claim it to be the only good constraint that is both easy to specify and sufficient for Auto-FuzzyJoin. Exploring additional such constraints is an interesting direction of future work.

L-id	L-name	L-color	L-capacity
l_1	iPhone 9	White	64 GB
l_2	iPhone 9	Gold	128 GB
l_3	iPhone 9	Gold	64 GB
l_4	...	...	...

R-id	R-name	R-color	R-capacity
r_1	iPhone 9	White	128 GB
r_2	...	...	...

(a) Without constraints like L being the reference table, (l_1, r_1), (l_2, r_1) are equally plausible joins. The Auto-FuzzyJoin problem is under-specified.

L-id	L-name	L-color	L-capacity
l_1	iPhone 9	White	64 GB
l_2	iPhone 9	Gold	128 GB
l_3	iPhone 9	Gold	64 GB
l_4	...	...	...

R-id	R-name	R-color	R-capacity
r_1	iPhone 9	White	128 GB
r_2	...	...	...

(b) With L being the reference table that is free of duplicate records, the Auto-FuzzyJoin problem is better constrained. Specifically, we know that neither (l_1, r_1) or (l_2, r_1) should join, because both l_2 and l_3 are in reference table L (indicating that records with different "capacities" are considered distinct entities, thus l_1 should not join r_1), and both l_1 and l_3 are in reference table L (indicating that records with different "colors" are considered as distinct entities, thus l_2 should not join r_1).

Figure 7: Examples product data set. The question is which l should r_1 join, in the absence of an exact match to r_1 "iPhone 9, White, 128GB".

B EXPERIMENTAL RESULTS

Dataset	AutoFJ	Unsupervised						Supervised			AutoFJ
		BestStaticJF	Excel	FW	ZeroER	ECM	PP	Megallan	DM	AL	24 configurations
Amphibian	0.543	0.52	0.537	0.518	0.507	0.264	0.485	0.807	0.595	0.861	0.536
ArtificialSatellite	**0.519**	0.451	0.447	0.399	0.063	0.1	0.219	0.22	0.004	0.099	0.402
Artwork	0.857	0.86	0.872	0.692	0.585	0.234	0.665	0.813	0.727	0.713	0.864
Award	**0.503**	0.466	0.453	0.387	0.327	0.111	0.333	0.385	0.318	0.171	0.477
BasketballTeam	**0.732**	0.638	0.701	0.465	0.031	0.321	0.445	0.528	0.182	0.446	0.699
Case	**0.983**	0.943	0.934	0.763	0.77	0.383	0.569	0.964	0.94	0.982	0.960
ChristianBishop	**0.836**	0.82	0.788	0.64	0.635	0.162	0.553	0.791	0.507	0.768	0.837
CAR	0.83	0.747	**0.862**	0.555	0.311	0.108	0.501	0.72	0.554	0.408	0.813
Country	**0.666**	0.582	0.65	0.477	0.348	0.119	0.327	0.579	0.351	0.426	0.644
Device	**0.753**	0.696	0.741	0.623	0.471	0.113	0.388	0.614	0.464	0.404	0.718
Drug	0.414	0.145	0.409	0.392	0.376	0.046	0.078	**0.618**	0	0.523	0.407
Election	0.725	0.67	0.64	0.339	0.291	0.099	0.229	**0.782**	0.712	0.352	0.753
Enzyme	**0.679**	0.541	0.593	0.573	0.502	0.133	0.539	0.447	0.012	0.327	0.675
EthnicGroup	0.861	0.774	0.824	0.665	0.482	0.11	0.06	0.826	0.721	**0.876**	0.846
FootballLeagueSeason	0.757	0.656	0.711	0.598	0.502	0.048	0.383	**0.896**	0.535	0.447	0.712
FootballMatch	**0.901**	0.858	0.658	0.592	0.648	0.126	0.611	0.898	0.056	0.596	0.895
Galaxy	0.369	0.302	**0.404**	0.293	0.002	0.096	0.142	0.291	0.044	0.123	0.409
GivenName	**0.972**	0.755	0.892	0.264	0.001	0.126	0.002	0.935	0.152	0.898	0.972
GovernmentAgency	0.676	0.63	**0.68**	0.544	0.465	0.152	0.444	0.603	0.64	0.473	0.678
HistoricBuilding	**0.824**	0.803	0.81	0.675	0.556	0.161	0.233	0.748	0.682	0.618	0.821
Hospital	**0.624**	0.618	0.609	0.418	0.308	0.156	0.254	0.462	0.355	0.189	0.625
Legislature	0.831	0.799	**0.838**	0.706	0.55	0.114	0.136	0.78	0.635	0.762	0.844
Magazine	**0.845**	0.821	0.823	0.624	0.251	0.148	0.453	0.629	0.507	0.554	0.849
MemberOfParliament	**0.869**	0.812	0.776	0.588	0.19	0.143	0.24	0.829	0.758	0.757	0.861
Monarch	0.72	0.509	**0.725**	0.535	0.404	0.193	0.247	0.675	0.264	0.585	0.680
MotorsportSeason	0.907	0.94	0.94	0.418	0.928	0.09	0.512	0.989	0.961	**0.994**	0.902
Museum	**0.68**	0.646	0.659	0.512	0.451	0.099	0.291	0.5	0.491	0.322	0.662
NCAATeamSeason	**1**	0.676	0.544	0.237	0.412	0.007	0.555	0.97	0.043	0.562	1.000
NFLS	**1**	**1**	0.848	0.5	0.5	0.12	0.929	0.962	0	0.681	1.000
NaturalEvent	**0.627**	0.597	0.483	0.418	0.388	0.3	0.529	0.11	0.173	0.039	0.652
Noble	0.739	0.717	**0.741**	0.556	0.591	0.221	0.418	0.653	0.456	0.56	0.732
PoliticalParty	**0.532**	0.441	0.496	0.329	0.388	0.101	0.244	0.519	0.287	0.341	0.514
Race	0.369	0.354	**0.387**	0.279	0.161	0.105	0.215	0.328	0.069	0.149	0.369
RailwayLine	**0.683**	0.553	0.631	0.437	0.421	0.15	0.312	0.616	0.218	0.443	0.642
Reptile	0.966	0.944	0.964	0.934	0.924	0.384	0.901	0.969	0.94	**0.985**	0.964
RugbyLeague	**0.588**	0.487	0.548	0.373	0.244	0.224	0.395	0.244	0.317	0.169	0.550
ShoppingMall	0.857	0.895	**0.915**	0.587	0.504	0.305	0.537	0.701	0.649	0.659	0.843
SoccerClubSeason	**0.971**	0.884	0.81	0.372	0.934	0.188	0.961	0.915	0.404	0.756	0.971
SoccerLeague	**0.449**	0.425	0.432	0.266	0.304	0.084	0.253	0.352	0.118	0.156	0.434
SoccerTournament	**0.88**	0.818	0.689	0.395	0.503	0.066	0.714	0.859	0.635	0.797	0.848
Song	0.928	0.823	0.891	0.626	0.666	0.19	0.535	0.935	0.884	**0.952**	0.924
SportFacility	0.464	0.412	0.443	0.344	0.295	0.107	0.216	**0.529**	0.226	0.297	0.450
SportsLeague	0.41	0.417	**0.437**	0.279	0.301	0.086	0.283	0.347	0.251	0.165	0.409
Stadium	0.446	0.39	0.43	0.327	0.283	0.107	0.213	**0.492**	0.277	0.277	0.434
TelevisionStation	0.554	0.473	0.284	0.202	0.089	0.043	0.079	0.567	0.316	**0.631**	0.380
TennisTournament	**0.816**	0.595	0.568	0.522	0.475	0.308	0.614	0.798	0.178	0.415	0.814
Tournament	0.73	0.668	0.588	0.365	0.379	0.079	0.54	**0.732**	0.476	0.619	0.702
UnitOfWork	**0.983**	0.943	0.934	0.763	0.768	0.41	0.569	0.965	0.945	0.973	0.960
Venue	**0.606**	0.57	0.6	0.476	0.458	0.148	0.181	0.584	0.368	0.426	0.606
Wrestler	0.28	0.263	0.277	0.233	0.004	0.086	0.159	**0.477**	0.131	0.332	0.277
Average	**0.715**	0.647	0.658	0.481	0.419	0.156	0.394	0.659	0.411	0.521	0.700

Table 5: PR-AUC Scores on 50 single-column fuzzy join datasets

Dataset	AutoFJ		Unsupervised					Supervised		
	24 configurations		Excel	FW	ZeroER	ECM	PPJoin	Megellan	DM	AL
	P	R	AR	AR	AR	AR	AR	AR	AR	AR
Amphibian	0.794	0.522	0.514	0.513	0.504	0.372	0.485	0.787	0.589	**0.861**
ArtificialSatellite	0.773	0.236	**0.375**	0.236	0.042	0.194	0.125	0.199	0.011	0.142
Artwork	0.920	0.845	**0.886**	0.731	0.592	0.371	0.518	0.588	0.385	0.715
Award	0.890	0.380	**0.409**	0.365	0.042	0.237	0.245	0.227	0.110	0.168
BasketballTeam	0.881	0.578	**0.645**	0.018	0.042	0.398	0.042	0.245	0.089	0.411
Case	0.978	0.929	0.887	0.763	0.642	0.529	0.092	0.848	0.928	**0.983**
ChristianBishop	0.948	**0.777**	0.650	0.591	0.387	0.283	0.500	0.622	0.281	0.755
CAR	0.904	0.747	**0.900**	0.421	0.095	0.221	0.389	0.567	0.221	0.408
Country	0.891	0.533	**0.553**	0.464	0.247	0.244	0.275	0.290	0.066	0.403
Device	0.902	0.518	**0.675**	0.503	0.222	0.198	0.299	0.122	0.205	0.337
Drug	0.761	0.325	0.401	0.376	0.401	0.045	0.070	0.526	0.008	**0.541**
Election	0.959	**0.618**	0.298	0.138	0.072	0.177	0.110	0.612	0.325	0.341
Enzyme	0.838	**0.646**	0.583	0.583	0.375	0.208	0.500	0.267	0.033	0.307
EthnicGroup	0.962	0.784	0.026	0.513	0.463	0.225	0.015	0.717	0.455	**0.876**
FootballLeagueSeason	0.855	0.486	0.671	0.575	0.468	0.132	0.282	**0.890**	0.316	0.437
FootballMatch	1.000	0.698	0.321	0.340	0.415	0.208	0.623	**0.715**	0.052	0.466
Galaxy	1.000	**0.353**	**0.353**	0.118	0.059	0.235	0.235	0.229	0.044	0.217
GivenName	0.979	**0.909**	0.487	0.078	0.013	0.442	0.286	0.552	0.060	0.828
GovernmentAgency	0.916	**0.611**	0.585	0.464	0.305	0.261	0.343	0.357	0.381	0.466
HistoricBuilding	0.952	**0.779**	0.758	0.549	0.389	0.236	0.066	0.492	0.191	0.602
Hospital	0.866	**0.529**	0.529	0.444	0.226	0.292	0.230	0.100	0.146	0.149
Legislature	0.960	**0.787**	0.769	0.704	0.431	0.208	0.023	0.604	0.261	0.748
Magazine	0.944	**0.796**	0.788	0.420	0.179	0.281	0.318	0.123	0.296	0.532
MemberOfParliament	0.952	0.664	0.608	0.308	0.018	0.205	0.008	0.617	0.476	**0.742**
Monarch	0.856	0.393	**0.653**	0.095	0.236	0.351	0.095	0.429	0.099	0.572
MotorsportSeason	0.941	0.869	0.943	0.843	0.920	0.196	0.938	0.985	0.963	**0.994**
Museum	0.915	**0.567**	0.554	0.370	0.236	0.193	0.193	0.115	0.103	0.225
NCAATeamSeason	1.000	0.382	0.059	0.588	0.412	0.118	0.294	**0.928**	0.059	0.503
NFLS	1.000	0.500	0.500	0.500	0.500	0.200	**1.000**	0.933	0.000	0.633
NaturalEvent	0.806	**0.569**	0.314	0.510	0.275	0.412	0.118	0.100	0.241	0.054
Noble	0.934	0.387	**0.654**	0.426	0.234	0.363	0.033	0.125	0.065	0.441
PoliticalParty	0.846	0.356	0.378	**0.408**	0.200	0.204	0.069	0.253	0.067	0.312
Race	0.781	0.326	**0.349**	0.223	0.143	0.194	0.160	0.211	0.034	0.100
RailwayLine	0.883	0.480	**0.581**	0.117	0.252	0.285	0.289	0.216	0.061	0.310
Reptile	0.771	0.964	0.941	0.932	0.925	0.527	0.893	0.968	0.937	**0.985**
RugbyLeague	0.931	**0.466**	0.224	0.259	0.052	0.276	0.259	0.139	0.221	0.145
ShoppingMall	0.795	0.805	**0.849**	0.642	0.308	0.509	0.547	0.546	0.317	0.686
SoccerClubSeason	0.972	0.686	0.922	0.549	0.647	0.275	**0.961**	0.825	0.331	0.668
SoccerLeague	0.806	0.366	**0.374**	0.067	0.218	0.168	0.197	0.171	0.084	0.116
SoccerTournament	0.934	0.738	0.517	0.597	0.403	0.176	0.579	0.782	0.320	**0.797**
Song	0.968	0.902	0.880	0.632	0.207	0.280	0.327	0.854	0.530	**0.954**
SportFacility	0.865	**0.402**	0.378	0.362	0.216	0.201	0.146	0.146	0.043	0.214
SportsLeague	0.771	0.343	**0.403**	0.289	0.154	0.179	0.214	0.057	0.084	0.110
Stadium	0.855	**0.380**	0.367	0.339	0.210	0.200	0.139	0.182	0.096	0.281
TelevisionStation	0.662	0.259	0.247	0.211	0.048	0.154	0.073	0.528	0.297	**0.642**
TennisTournament	0.941	0.593	0.444	0.556	0.370	0.556	0.519	**0.674**	0.257	0.433
Tournament	0.877	0.588	0.423	0.468	0.275	0.207	0.495	**0.661**	0.194	0.606
UnitOfWork	0.975	0.929	0.895	0.763	0.682	0.550	0.300	0.854	0.819	**0.976**
Venue	0.897	0.544	**0.555**	0.490	0.380	0.214	0.133	0.480	0.080	0.423
Wrestler	0.804	0.256	0.241	0.222	0.006	0.164	0.080	**0.406**	0.069	0.317
Average	0.892	**0.582**	0.546	0.433	0.303	0.267	0.303	0.477	0.246	0.499

Table 6: Precision and Recall on 50 single-column datasets with 24 configurations

Dataset	AutoFJ	Unsupervised					Supervised		
		Excel	FW	ZeroER	ECM	PP	Magellan	DM	AL
RI	0.971	0.775	0.596	0.992	0.947	0.893	0.997	0.898	**0.998**
AB	**0.800**	0.559	0.008	0.281	0.125	0.382	0.544	0.268	0.384
BB	**0.675**	0.414	0.396	0.028	0.524	0.476	0.639	0.411	0.611
BR	0.913	0.801	0.740	0.492	0.666	0.754	0.913	0.813	**0.962**
ABN	0.851	0.894	0.871	0.918	0.852	0.830	0.984	0.870	**0.988**
DA	0.978	0.965	0.688	0.942	0.060	0.974	0.987	0.969	**0.999**
FZ	0.760	0.998	0.761	0.933	0.097	0.960	0.998	0.954	**1.000**
IA	0.824	0.871	0.606	0.821	0.629	0.682	0.974	0.650	**0.969**
Average	0.847	0.785	0.583	0.676	0.487	0.744	**0.879**	0.729	0.864

Table 7: PR-AUC Scores on 8 multi-column fuzzy join datasets