

Fuzzy Integration of Data Lake Tables

Aamod Khatiwada
Northeastern University
Boston, USA
khatiwadaaamod@gmail.com

Roe Shraga
Worcester Polytechnic Institute
Worcester, USA
rshraga@wpi.edu

Renée J. Miller
Northeastern U. & U. of Waterloo
Waterloo, Canada
rjmiller@uwaterloo.ca

Abstract

Data integration is an important step in any data science pipeline where the objective is to unify the information available in different datasets for comprehensive analysis. Full Disjunction, which is an associative extension of the outer join operator, has been shown to be an effective operator for integrating datasets. It fully preserves and combines the available information. Existing Full Disjunction algorithms only consider the equi-join scenario where only tuples having the same value on joining columns are integrated. This, however, does not realistically represent many realistic scenarios where datasets come from diverse sources with inconsistent values (e.g., synonyms, abbreviations, etc.) and with limited metadata. So, joining just on equal values severely limits the ability of Full Disjunction to fully combine datasets. Thus, in this work, we propose an extension of Full Disjunction to also account for “fuzzy” matches among tuples. We present a novel data-driven approach to enable the joining of approximate or fuzzy matches within Full Disjunction. Experimentally, we show that fuzzy Full Disjunction does not add significant time overhead over a state-of-the-art Full Disjunction implementation and also that it enhances the accuracy of a downstream data quality task.

Keywords

Data Integration, Fuzzy Matching, Data Lakes, Full Disjunction

1 Introduction

Data lakes may store an enormous amount of heterogeneous data. Within data lakes, tables are one of the most prevalent data formats [14, 33], and tables are useful for data scientists running complex analyses and making decisions [12, 27]. However, the tables may represent information from different topics, may use inconsistent values (e.g., synonyms or abbreviations) and may have unreliable, incomplete or inconsistent metadata (e.g., table names and column headers) making it difficult for data scientists to find data lake tables that are relevant for their analysis. Consequently, different semantic table search techniques have been proposed [11, 19, 28]. Such techniques generally allow users to search using keyword queries [3, 11, 29] for tables related to the keywords. Alternatively, in the table-as-query paradigm, an existing table is the query [37] and the search is for tables that are related [2], most unionable [10, 15, 18, 28], or most joinable [2, 9, 35, 37]) to the query table.

After discovery, the required information for analysis could be scattered among query and searched tables. So, the next natural step is integration and the generation of a unified view of relevant data [1, 20, 21]. Two major challenges have been considered for integrating a set of discovered tables. The first is to determine which columns should be aligned together in the integrated table. A possible solution could be to align the columns having

T_1				T_2			T_3				
TID	City	Country		TID	Country	City	Vac. Rate (1+ dose)	TID	City	Total Cases	Death Rate (per 100k)
t_1	Berlin	Germany		t_5	CA	Toronto	83%	t_9	Berlin	1.4M	147
t_2	Toronto	Canada		t_6	US	Boston	62%	t_{10}	barcelona	2.68M	275
t_3	Barcelona	Spain		t_7	DE	Berlin	63%	t_{11}	Boston	263K	335
t_4	New Delhi	India		t_8	ES	Barcelona	82%				

FD(T_1, T_2, T_3)

OID	TIDs	City	Country	Vac. Rate (1+ dose)	Total Cases	Death Rate (per 100k)
f_1	{ t_1 }	Berlin	Germany	↓	↓	↓
f_2	{ t_2 }	Toronto	Canada	↓	↓	↓
f_3	{ t_3 }	Barcelona	Spain	↓	↓	↓
f_4	{ t_4 }	New Delhi	India	↓	↓	↓
f_5	{ t_5 }	Toronto	CA	83%	↓	↓
f_6	{ t_6, t_{11} }	Boston	US	62%	263K	335
f_7	{ t_7, t_9 }	Berlin	DE	63%	1.4M	147
f_8	{ t_8 }	Barcelona	ES	82%	↓	↓
f_9	{ t_{10} }	barcelona	↓	↓	2.68M	275

Fuzzy FD(T_1, T_2, T_3)

OID	TIDs	City	Country	Vac. Rate (1+ dose)	Total Cases	Death Rate (per 100k)
f_{10}	{ t_1, t_7, t_9 }	Berlin	Germany	63%	1.4M	147
f_{11}	{ t_2, t_5 }	Toronto	Canada	83%	↓	↓
f_{12}	{ t_3, t_6, t_{10} }	Barcelona	ES	82%	2.68M	275
f_{13}	{ t_4 }	New Delhi	India	↓	↓	↓
f_{14}	{ t_6, t_{11} }	Boston	US	62%	263K	335

Figure 1: Tables about COVID-19 cases in different cities. The column headers are given only for easy reference. However, they may not be available in practice.

the same column headers. However, since data lake tables may have missing, inconsistent, and unreliable column headers, this becomes challenging. Hence, we cannot rely on them for comprehensive integration and to make them consistent, techniques such as schema matching are applied [1, 22].

After determining the aligned columns, the second challenge is to find an integration operator (or query) to merge the tuples and generate an integrated table. Basic integration operators such as inner join, union, outer join, and so on, may not be effective as they may not retain all the information during integration [13, 17, 30]. For instance, the inner join operator, when integrating a set of tables (an *integration set*), does not retain a tuple if it has no joining partner tuple even in a single table. This can be problematic, particularly when we have a large integration set. Outer join solves the inner join problem by retaining tuples without join partners. However, the outer join is not an associative operator and different orders of applying outer join over a set of tables generate different sets of partially integrated tuples [5, 32]. Note that it is possible that no order of applying outer join can produce complete tuples [5]. Consequently, Galindo-Legaria [13] introduced the Full Disjunction (FD) operator, which is an associative version of the outer join operator. Importantly, FD joins each tuple in the tables to be integrated in a maximal way such that no information is lost (even if there is no joining tuple) and the incompleteness in tuples is minimized [32]. Hence, FD has been considered an optimal way of integrating information present in different tables [32] for decades. We refer to the literature for further details on Full Disjunction [5, 32], including its scalable [20] and parallelized implementations [30].

EDBT '26, Tampere (Finland)

© 2025 Copyright held by the owner/author(s). Published on OpenProceedings.org under ISBN 978-3-98318-102-5, series ISSN 2367-2005. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

Notice however that the existing Full Disjunction definition and algorithms only consider joining tuples on equal values [5, 13, 20]. In reality, data lake tables come with inconsistencies such as abbreviations, synonyms, and more. So, relying on equi-joins impacts Full Disjunction’s ability to integrate the tables well and also impacts the usage of integrated tables for downstream tasks. Instead, we need “fuzzy” matching between the values. Fuzzy Join [8], also known as similarity join, is an extension of the traditional equi-join operator to produce record pairs that approximately match. In this paper, we address the aforementioned generalization of the join operator within Full Disjunction and show that this extension addresses the noise and inconsistencies that data lakes exhibit [27].

We adopt an existing table integration method (ALITE) [20] that implements Full Disjunction and propose an extension that accounts for fuzzy matches between the values. Our solution first resolves the inconsistencies between the column cells representing the same values. After making the values consistent, it applies the FD operator to integrate the tuples. We selected to extend ALITE rather than other Full Disjunction algorithms because ALITE allows the input tables to be incomplete (something that occurs often in data lakes), it handles arbitrary join patterns (even cyclic joins), and it has been shown to outperform other algorithms on large tables (and large integration sets).

EXAMPLE 1. Consider the three tables about COVID-19 cases in Fig. 1. (simplified versions of data taken from COVID-19 open datasets).^{a b} The columns TID and OID are used for illustration to clearly indicate which tuples (in TID column) were integrated to produce this new tuple (OID column). For simplicity, columns that align are given the same name in the three tables and are highlighted in the same colors. Table $FD(T_1, T_2, T_3)$ shows the Full Disjunction result using an equi-join. Since T_1 has a typo in Tuple t_1 (Berlinn), Full Disjunction does not integrate it with other tuples about Berlin (t_7 and t_9) forming separate tuples f_1 and f_7 . Furthermore, as two aligned Country Columns in Tables T_1 and T_2 contain the full names and codes of countries respectively, FD does not integrate Tuples t_2 and t_5 and Tuples t_3 and t_8 . Moreover, Tuples t_3 and t_{10} are both about Barcelona but they are not integrated by FD as they are represented in different cases (Barcelona in t_3 and barcelona in t_{10}). On the other hand, Fuzzy $FD(T_1, T_2, T_3)$ shows tuples integrated using our proposed algorithm where the tuples are integrated maximally without redundancy.

^a<https://catalog.data.gov/dataset/covid-19-outcomes-by-vaccination-status>, <https://catalog.data.gov/dataset/provisional-covid-19-death-counts-rates-and-percent-of-total-deaths-by-jurisdiction-of-res>

^b Please note that open data is not a sole use case and our method also works for relational and web tables.

Next, we summarize our contributions.

- **Fuzzy Full Disjunction:** To the best of our knowledge, we are the first to propose fuzzy integration of tuples using the Full Disjunction Operator. Specifically, our method first identifies fuzzy matches, makes them consistent, and then applies Full Disjunction.
- **Empirical Evaluation:** We show experimentally that our novel Fuzzy Full Disjunction method, without significantly increasing runtime, enhances integration effectiveness in a down-stream data quality task.
- **Open Source Code and Benchmark:** Our code, models, and datasets are publicly available: https://github.com/northeastern-datalab/fuzzy_fd

2 Related Work

Galindo-Legaria [13] introduced FD as an associative alternative to the outer join operator which is computed by applying outer join in all possible orders of the input tables. Next, the results are outer unioned to generate all possible FD tuples. A subsumption operator is then applied to eliminate a tuple that is contained in another tuple (i.e., tuples containing partial information). Rajaraman and Ullman [32] stated that FD is the right semantics for data integration and several others [5, 17, 30] propose algorithms to compute FD faster in practice. Recently, Khatiwada et al. [20] proposed ALITE, which uses the Full Disjunction operator to integrate data lake tables discovered using different table search techniques. Since data lake tables have inconsistent and unreliable column headers [14, 26], ALITE first determines the matching columns in the tables to be integrated by applying holistic schema matching [34] over column-based pre-trained embeddings. After that, ALITE uses the (Natural) Full Disjunction [13], over the matched columns to produce an integrated table. Notice that all the prior work considers equi-join integration of tuples, i.e., the values of the tuples to be integrated are consistent and can be matched using the equality operator. We relax this assumption and propose a novel way of applying Full Disjunction over data lake tables using approximate or fuzzy joins.

Previous work has addressed the challenge of identifying fuzzy inner joins. Zhu et al. [36] determined fuzzy matches by employing string transformations such as matching n-grams of cell values and concatenating cell values. Li et al. [23] determined fuzzy matches between the values within a column pair by selecting suitable parameters for the given input table. They framed fuzzy matching as an optimization task with an objective of maximizing matching recall under a precision constraint. In our work, we address fuzzy Full Disjunction rather than inner join. For comparison, we have used the latest of these [23] as a baseline.

3 Fuzzy Full Disjunction

In this section, we describe our proposed method for integrating data lake tables using Full Disjunction, considering fuzzy matches of values in the join columns. Our system takes as input a set of tables that may have inconsistent column headers and inconsistent value representations and outputs a maximally integrated table. The system is illustrated in Fig. 2. We first use ALITE’s holistic column matcher to determine the aligned columns in the input tables. Aligned columns are annotated with the same *integration id*. The tables after column annotation are denoted as *Annotated Tables* in Fig. 2. Over the aligned columns of annotated tables, we apply our novel value matching component which addresses inconsistencies among join values. Once value matches are identified, the matching values are replaced with a single consistent value before applying the FD operator.

3.1 ALITE’s Holistic Column Matcher

Before explaining how we implement the value matching component, we briefly describe ALITE’s column matcher[20]. ALITE considers a set of tables to be integrated as input that can have inconsistent column headers and we cannot rely on them for integration. So first, ALITE determines matching columns among all the input columns by applying holistic schema matching over them. Specifically, ALITE first represents each input column using pre-trained embeddings over their values, and over such embeddings, it applies hierarchical clustering to get distinct sets

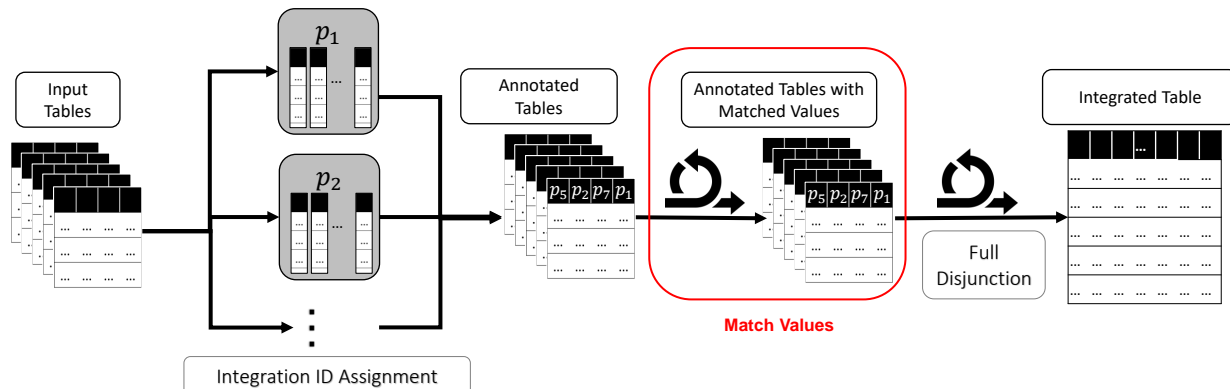


Figure 2: Block Diagram of the proposed system for Fuzzy Integration of Data Lake Tables.

of matching columns. The matched columns are assigned with a unique column integration ID. Because ALITE uses embeddings of column values, its matcher is robust to tables with inconsistent value representations and can be applied in our new setting of fuzzy joins. The ALITE column matcher ensures each column of a table T_1 in the integration set matches with at most one column from any other table T_2 (and does not match columns within a table).

3.2 Value Matching

In its current form, ALITE only considers semantic inconsistencies in aligning columns as described above. However, as motivated in our running example (see Fig. 1), inconsistencies can present themselves also on a value level. That means that two values, such as Canada and CA, that do not have an exact match, should and will be considered when integrating the tables via Full Disjunction (right-most part of Fig. 2).

Now, we discuss how we determine the fuzzy matching values within the aligning columns. Following the literature on the problem of entity matching (EM) [24], we consider a clean-clean value matching scenario [4], i.e., there is no inconsistency of value representations within a column and we want to match potentially inconsistent representations across different (matched) columns. This means, for example, that when referring to Canada within a single column, the value “Canada” is used consistently, but in other tables this value might appear as “CA” or “CAN”. Note that, different from the EM problem, here we consider single column values (though these values may be phrases containing many words) for matching rather than full rows (tuples). Furthermore, we consider de-duplicated column values for our discussion. However, it is trivial to handle duplicate values in a column. The matching process can be extended by first de-duplicating the values within a column while retaining a representative row for each value group and keeping track of the removed duplicates. Once matching is performed on the de-duplicated rows, the removed rows can be reinstated by associating them with their representatives.

Let T be a table in the integration set (left-most part of Fig. 2). We denote a set of aligning columns using C and a specific i^{th} column in the set using c_i . Similarly, we represent a j^{th} value in a list of values V using v_j . Furthermore, $c_i.v_j$ represents the j_{th} value of Column c_i . Next, we formally define the Fuzzy Value Match problem.

DEFINITION 2 (FUZZY VALUE MATCH PROBLEM). *We are given a set of aligned columns $c_1, c_2 \dots c_n$, a list of their values*

$[c_i.v_j | i \in 1 \dots n; j \in 1, 2, \dots]$, a matching threshold θ , and a distance function $\text{dist}(\cdot)$. The Fuzzy Value Match Problem is to find a disjoint set of values $V_1, V_2, \dots V_k$ such that $\forall u, v \in V_i$ ($1 \leq i \leq k$) input, $\text{dist}(u, v) < \theta$.

The definition, which aims to obtain sets of values that represent the matching meanings, uses two key components, namely a *distance function* and a *threshold*, which we address in the next section.

3.3 Match Values Implementation

Now, we explain how we implement the Match Values component that determines the fuzzy matches between the values.

Embed Column Values: We first represent each column value in a fixed-dimension embedding space to ensure that matched values are close to each other in this embedding space. For instance, cells referring to Canada with values “Canada” and “CA” in T_1 and T_2 (Fig. 1), respectively, are embedded near each other, while “Germany” and “CA” are embedded farther apart. Similarly, we aim to embed “Berlinn” and “Berlin” close to each other as they both represent Berlin. In our system, we embed each cell using Mistral-7B-Instruct model,¹ a recent large language model that we used in our experimental analysis (detailed in Sec. 4).

Determine Fuzzy Matches: In the context of the clean-clean scenario [4, 31], the values within each column are consistent (that is, two values have the same meaning iff they are identical). In what follows, our approach identifies fuzzy matches between values across aligned columns. We initiate this process by selecting a pair of aligned columns and determining the fuzzy matches between their respective sets of values. To determine these matches, we compute the cosine distances between the embeddings of cell values from the first column and those from the second column. Based on these distances, we perform bipartite matching between the values of the column pairs. Specifically, we apply a linear sum assignment algorithm [6] that identifies an optimal bipartite match between the values, minimizing the total distance between the matched values. Note that we do not allow matches whose distance is higher than the threshold θ .

EXAMPLE 3. *Consider the Country columns of Tables T_1 and T_2 in Fig. 1 that are aligned. We apply bipartite matching between their sets of values. Based on the embeddings, Germany is matched with DE, Canada is matched with CA, and Spain is matched with ES. Bipartite matching matches India in T_1 with US in T_2 but their*

¹https://huggingface.co/docs/transformers/main/en/model_doc/mistral

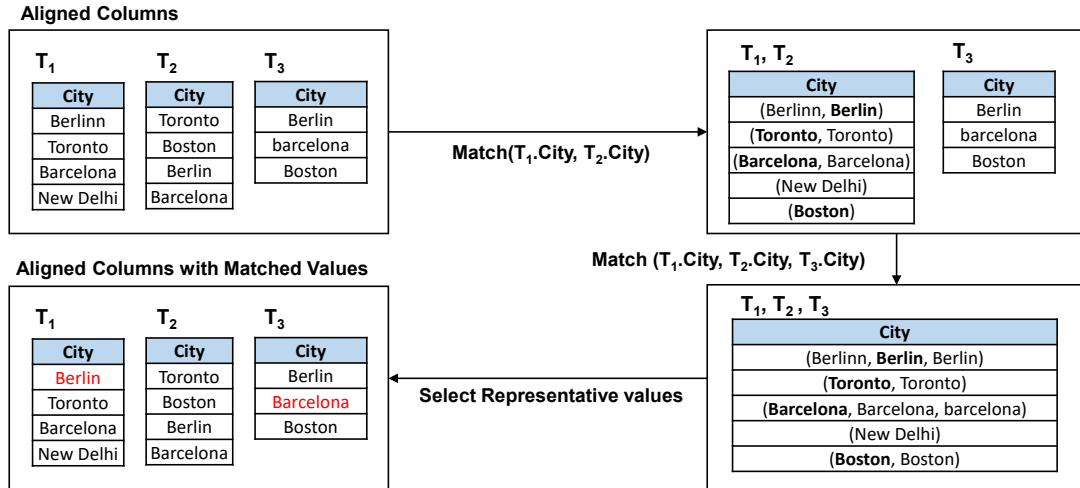


Figure 3: Applying Match Values component over the aligning columns for Fuzzy Integration of Data Lake Tables.

match score is above the threshold. So, this match is discarded and these values are placed in separate value sets.

Once we determine a match between the values of two columns, we outer join the columns to generate a combined column. If a value in one column is not matched with a value in another column, it is left in a singleton set represented by its embedding. If two values match, we select the most representative value embedding, i.e., the one that appears most frequently in the list of all values from the aligned columns. In the case of a tie, we select a value from the first table among the two matching tables each time, to keep the assignment consistent. This process produces a combined column, which we then use for bipartite matching with another aligned column. We continue producing the combined column and matching it with another aligned column until all fuzzy matches in the set of aligned columns are determined.

EXAMPLE 4. Figure 3 illustrates three aligned City columns from T_1 , T_2 , and T_3 in Fig. 1. In the first step, we match the City columns from T_1 and T_2 . This results in Berlinn, Toronto, and Barcelona from T_1 being matched with Berlin, Toronto, and Barcelona from T_2 , respectively. New Delhi remains unmatched. Since Berlin appears twice and Berlinn appears once across all three columns, we select Berlin for the combined column (bold in Table T_1, T_2 on the top-right corner). For the other two values (Toronto and Barcelona), as they are identical, we simply select one for the combined column. New Delhi with no other matches is also added to the combined column. Next, we match the combined column from T_1 and T_2 with the City column from T_3 . This process results in the final combined column containing the values Berlin, Toronto, Barcelona, New Delhi, and Boston (bottom-right corner).

The column values in the final combined column are selected as the representative values for each set of matched values. Then, we replace all of the values across the aligned columns with their respective representative value. For instance, continuing Ex. 4, as shown in the bottom-left corner of Fig. 3, we replace Berlinn and barcelona with their representative values Berlin and Barcelona.

After matching the values in each set of aligned columns, all value level inconsistency are resolved. Consequently, we apply the equi-join Full Disjunction operator from ALITE [20] and it

integrates the tuples without missing the integration on fuzzy values.

Finally, we note that the threshold, representing the sensitivity of value matching, is set empirically. This highlights an important trade-off that comes from the literature on fuzzy/similarity joins. Specifically, setting the threshold very low, even if the embedding is effective, might not match together semantically similar values that should be considered for join. On the other end, setting the value to be very high may end up with too many matches. For example, the value Barcelona from T_1 (Fig. 3) and the value Berlin from T_2 are semantically similar (major cities in Europe) and thus have a non-negligible value matching score. However, the assumption is that this value should be higher than that of Berlinn and Berlin. Even if the distance between Barcelona and Berlin is less than an overly high threshold, the bipartite matching would correctly match Berlin with Berlinn, rather than Barcelona. To select the optimal threshold, we conducted a grid search experiment, varying the threshold value between 0 and 1, aiming to maximize matching effectiveness. Our results showed that the threshold has minimal impact on the effectiveness of our method, as it is applied after bipartite matching, which already determines the best possible matches. We report our results with a threshold of 0.7 and provide results with other thresholds in our GitHub repository.²

4 Experiments

Now we empirically evaluate our Fuzzy Full Disjunction method. In our evaluation, we aim to answer the following questions:

- (1) How effective is our method in determining the fuzzy matches between the values?
- (2) How efficient is our proposed method in integrating the tables considering fuzzy matches?

Specifically, we evaluate several embedding models to assess their performance for value matching. Our primary point of comparison is state-of-the-art (equi-based) full disjunction algorithms [5, 20].

²https://github.com/northeastern-datalab/fuzzy_fd

4.1 Experimental Setup

We run all our experiments using Python 3.10 on a server having Intel(R) Xeon(R) Gold 6346 CPU @ 3.10GHz and NVIDIA A40 GPU.

We implement the ALITE Full Disjunction operator [20] using the publicly available code³ and use scipy’s implementation of the linear sum assignment algorithm to perform bipartite matching.⁴

Benchmarks. We run our experiments over a publicly available fuzzy join benchmark and an integration benchmark from the literature.

(i) *Auto-Join Benchmark.* **Auto-Join** is a publicly available⁵ fuzzy tuple matching Benchmark that comes with 31 integration sets of tables (each with about 4 columns on average) covering 17 topics such as songs, government official details, etc. [36]. Each integration set contains sets of aligned columns (having around 150 values per column on average) that can be joined in a fuzzy manner under a clean-clean scenario [4]. We run our experiments over such joining columns. We use this benchmark to evaluate the accuracy of our fuzzy matching algorithm over different embedding methods.

(ii) *ALITE EM Benchmark.* Khatiwada et al. [20] created a set of datasets using open data tables to evaluate the effectiveness and efficiency of table integration methods. We use their Entity Matching Benchmark to evaluate effectiveness. This benchmark includes tables on football players, their positions, teams, and related team information such as home stadium, capacity, and opening date. To create the entity matching scenario, they augment an original table with new tuples that can be fuzzily matched to existing ones, incorporating abbreviations, typos, and similar variations. After this augmentation, the original table is partitioned into smaller tables, which are then integrated using different methods during the experiment. After integration, entity matching is applied as a downstream task to evaluate integration quality.

(iii) *ALITE Movie Benchmark.* Khatiwada et al. [20] also created a (non-fuzzy) benchmark based on **IMDB** movie dataset containing about 106M tuples distributed in 6 tables, to study the efficiency of different Full Disjunction implementations.⁶ Specifically, they sampled the rows from the IMDB tables to create integration sets containing 5K to 30K input tuples and study FD’s runtime over them. Although this is an equi-join benchmark, as the Match Values component still needs the same time to check for the fuzzy matches even if they do not exist, we use this benchmark to study the efficiency of our method against the baselines.

Baselines. To embed the column values, we evaluate different embedding baselines. Specifically, we implement a publicly available word embedding model (**FastText** [16]).⁷ We also implement two pre-trained language models (**BERT** [7] and **RoBERTa** [25]) and two large language models: **Mistral** (Mistral-7B-Instruct-v0.3) and **Llama3** (Meta-Llama-3-8B-Instruct) available in Hugging Face library.⁸ For each language model, we pass the cell values through its layers and extract the embeddings of the last hidden layer. Next, we implement Auto-FuzzyJoin (**AutoFJ**) [23] baseline using its public implementation. AutoFJ takes as input a

pair of tables and a user-defined precision target between 0 and 1. It then maximizes recall while ensuring the specified precision target is met. Following the original paper, we use a default precision target of 0.9 for our experiments and provide results for other precision targets in our GitHub repository.² In addition, we use (non-fuzzy) **ALITE** as an integration baseline.³ We also use **BICOMNLOJ**, another non-fuzzy FD algorithm [5], for an efficiency experiment.

Evaluation Metrics. To report the value match effectiveness, we use the standard Precision (P), Recall (R), and F_1 -Score (F_1). We also report runtime to compare efficiency. If M_G is the set of value matches in the ground truth and M_M is the set of value matches given by a method,

$$P = \frac{M_G \cap M_M}{M_M}, R = \frac{M_G \cap M_M}{M_G}, F_1 = \frac{2 \cdot P \cdot R}{P + R} \quad (1)$$

4.2 Results

Now we discuss the results of our experiments.

4.2.1 Fuzzy Matching Effectiveness. First, we report the performance of different embedding methods in determining the fuzzy matches. Table 1 shows the average performance of each embedding baseline over 31 sets of aligning columns in Auto-Join Benchmark. It is seen that although being smaller in size (7B parameters) than the second best model Llama3 (8B parameters), Mistral outperforms all the language models in terms of Precision, Recall, and F_1 -Score. Other models, FastText, BERT, and RoBERTa, show lower performance by at least 8% in terms of all metrics than Mistral. This shows that pre-trained embeddings of large language models can be used to embed the column values for fuzzy matches. As Mistral performs better than Llama3, even being slightly smaller than Llama3 in terms of the number of parameters, we use it in our system and for all our experiments. Another baseline, AutoFJ, is the most precise system since it is designed to meet a user-defined precision target. However, this comes at the cost of lower recall, which reduces its overall F_1 -Score. We also experimented with different precision target values, but AutoFJ did not outperform the top two methods in terms of F_1 -Score for any of them.

Table 1: Value Matching effectiveness of different models in Auto-Join Benchmark. The best score along each column is in bold; the second best score is underlined.

Model	Precision	Recall	F_1 -Score
FastText	0.70	0.67	0.66
BERT	0.72	0.76	0.73
RoBERTa	0.73	0.77	0.74
AutoFJ	0.84	0.65	0.68
Llama3	<u>0.81</u>	<u>0.85</u>	<u>0.81</u>
Mistral	<u>0.81</u>	0.86	0.82

4.2.2 Downstreaming Task Effectiveness. Next, we report the effectiveness of integration using Fuzzy FD over regular FD in the ALITE EM Benchmark. Specifically, we perform entity matching over the integrated table created using our fuzzy FD and over the table created using regular FD (ALITE) and report the effectiveness. For reference, we also pick an arbitrary join order and perform entity matching over the result of outer (equi) join over the tables. The results are reported in Table 2. Entity resolution over our Fuzzy FD integration is better than that over the regular

³<https://github.com/northeastern-datalab/alite>

⁴https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear_sum_assignment.html

⁵<https://github.com/Yeye-He/Auto-Join>

⁶<https://datasets.imdbws.com/>

⁷<https://github.com/facebookresearch/fastText>

⁸<https://huggingface.co/>

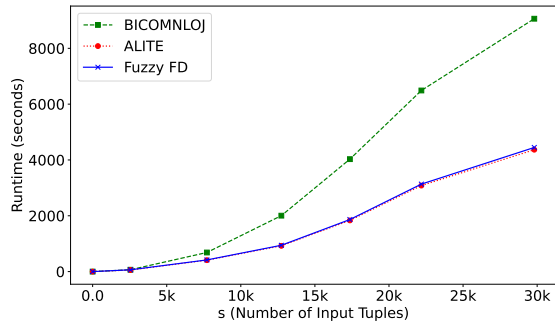


Figure 4: Runtime comparison of Regular Full Disjunction Algorithms against Fuzzy FD in IMDB Benchmark.

FD in terms of all three metrics: Precision, Recall, and F_1 -Score (by over 7%, 2%, and 5% respectively). Specifically, precision improves with Fuzzy FD because it eliminates the false negatives (tuples that are not correctly resolved) that regular FD produces due to unmatched values. Additionally, Fuzzy FD’s better integration of tuples provides more information for the entity matching algorithm, which as a result, retains more true positive tuples, increasing recall.

Table 2: Entity Resolution over tables that are integrated using Regular Full Disjunction against Fuzzy Full Disjunction. The best scores are bolded.

Integration Method	Precision	Recall	F_1 -Score
Outer Join	0.33	0.39	0.36
Regular FD (ALITE)	0.79	0.83	0.81
Fuzzy FD	0.86	0.85	0.86

4.2.3 Efficiency. We compare the runtime of ALITE’s regular FD Operator [20] and an alternative algorithm BICOMNLOJ [5] against our Fuzzy FD over the IMDB Benchmark. Recall this benchmark has no fuzzy value matches, but we run our fuzzy matching and confirm that it correctly outputs just singleton value matches. For a comprehensive evaluation, we report the total number of input tuples considered for integration on the X-axis and the runtime on the Y-axis. As shown in Fig. 4, the lines for ALITE (fastest among regular FD algorithms) and fuzzy FD methods almost overlap throughout the graph, showing that our fuzzy FD algorithm, although performing an additional value matching step, does not add significant additional time overhead to the regular Full Disjunction.

5 Discussion and Future Work

Finally, we discuss some limitations, potential extensions, and tradeoffs of using Fuzzy FD in data integration. First, Full Disjunction has exponential complexity in the number of input tables, limiting its scalability. As a result, it is practical only for a modest number of tables. This fits our scenario where we only deal with around 10 to 15 tables retrieved using table discovery techniques for integration. If we have hundreds of tables, it may not be feasible to apply FD. Second, the Integration ID Assignment phase may not always achieve perfect precision, which can affect the quality of downstream integration. An interesting future direction would be to add an LLM-based feedback loop that can help to automatically correct any imperfect alignment to enhance integration effectiveness. Moreover, we currently embed each cell

independent of the rest of the table context in the value match phase; this could be enhanced using more table context.

We presented an extension of the Full Disjunction algorithm to integrate a set of tables considering fuzzy matches between the values. Experimentally, we showed that our Fuzzy Full Disjunction algorithm does not add significant runtime overhead while it does improve integration effectiveness. To show the latter, we applied a downstream entity-matching algorithm over the result of FD and Fuzzy FD and showed a 5% improvement in F_1 -Score for Fuzzy FD.

Acknowledgments

This work was supported in part by NSF under award numbers IIS-1956096, IIS-2107248, and IIS-2325632. We acknowledge the support of the Canada Excellence Research Chairs (CERC) program. Nous remercions le Chaires d’excellence en recherche du Canada (CERC) de son soutien.

6 Artifacts

The source code, data, and/or other artifacts have been made available at: https://github.com/northeastern-datalab/fuzzy_fd.

References

- [1] Jens Bleiholder and Felix Naumann. 2009. Data Fusion. *ACM Comput. Surv.* 41, 1, Article 1 (Jan. 2009), 41 pages. doi:10.1145/1456650.1456651
- [2] Alex Bogatu, Alvaro A. A. Fernandes, Norman W. Paton, and Nikolaos Konstantinou. 2020. Dataset Discovery in Data Lakes. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 709–720. doi:10.1109/ICDE48307.2020.00067
- [3] Dan Brickley, Matthew Burgess, and Natasha Noy. 2019. Google Dataset Search: Building a Search Engine for Datasets in an Open Web Ecosystem. In *The World Wide Web Conference*. ACM, 1365–1375. doi:10.1145/3308558.3313685
- [4] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2021. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.* 53, 6 (2021), 127:1–127:42. doi:10.1145/3418896
- [5] Sara Cohen, Itzhak Fadida, Yaron Kanza, Benny Kimelfeld, and Yehosha Sagiv. 2006. Full Disjunctions: Polynomial-Delay Iterators in Action. In *VLDB 2006*. ACM. <http://dl.acm.org/citation.cfm?id=1164191>
- [6] David Frederic Crouse. 2016. On implementing 2D rectangular assignment algorithms. *IEEE Trans. Aerosp. Electron. Syst.* 52, 4 (2016), 1679–1696. doi:10.1109/TAES.2016.140952
- [7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv abs/1810.04805* (2019).
- [8] AnHai Doan, Alon Halevy, and Zachary Ives. 2012. *Principles of data integration*. Elsevier.
- [9] Yuyang Dong, Kunihiko Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient Joinable Table Discovery in Data Lakes: A High-Dimensional Similarity-Based Approach. In *37th IEEE International Conference on Data Engineering, ICDE 2021*. IEEE, 456–467. doi:10.1109/ICDE51399.2021.00046
- [10] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *PVLDB* 16, 7 (2023), 1726–1739.
- [11] Raul Castro Fernandez, Ziawasch Abedjan, Famen Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1001–1012.
- [12] Sainyam Galhotra and Udayan Khurana. 2020. Semantic Search over Structured Data. In *CIKM 2020*. Association for Computing Machinery, 3381–3384. doi:10.1145/3340531.3417426
- [13] César A. Galindo-Legaria. 1994. Outerjoins as Disjunctions. In *SIGMOD Conference 1994*. ACM, 348–358. doi:10.1145/191839.191908
- [14] Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. 2023. Data Lakes: A Survey of Functions and Systems. *IEEE Trans. Knowl. Data Eng.* 35, 12 (2023), 12571–12590. doi:10.1109/TKDE.2023.3270101
- [15] Xuming Hu, Shen Wang, Xiao Qin, Chuan Lei, Zhengyuan Shen, Christos Faloutsos, Asterios Katsifodimos, George Karypis, Lijie Wen, and Philip S. Yu. 2023. Automatic Table Union Search with Tabular Representation Learning. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*. Association for Computational Linguistics, 3786–3800. <https://aclanthology.org/2023.findings-acl.233>
- [16] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* (2016).

- [17] Yaron Kanza and Yehoshua Sagiv. 2003. Computing Full Disjunctions. In *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '03)*. ACM, 78–89. doi:10.1145/773153.773162
- [18] Aamod Khatiwada, Grace Fan, Roe Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J Miller, and Mirek Riedewald. 2023. SANTOS: Relationship-based Semantic Table Union Search. *Proc. ACM Manag. Data* 1, 1 (2023), Article 9. doi:10.1145/3588689
- [19] Aamod Khatiwada, Harsha Kokel, Ibrahim Abdelaziz, Subhajit Chaudhury, Julian Dolby, Oktie Hassanzadeh, Zhenhan Huang, Tejaswini Pedapati, Horst Samulowitz, and Kavitha Srinivas. 2025. TabSketchFM: Sketch-Based Tabular Representation Learning for Data Discovery over Data Lakes. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1523–1536. doi:10.1109/ICDE65448.2025.00118
- [20] Aamod Khatiwada, Roe Shraga, Wolfgang Gatterbauer, and Renée J. Miller. 2022. Integrating Data Lake Tables. *Proc. VLDB Endow.* 16, 4 (2022), 932–945. doi:10.14778/3574245.3574274
- [21] Aamod Khatiwada, Roe Shraga, and Renée J. Miller. 2023. DIALITE: Discover, Align and Integrate Open Data Tables. In *Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023*. ACM, 187–190. doi:10.1145/3555041.3589732
- [22] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine: Evaluating Matching Techniques for Dataset Discovery. In *37th IEEE International Conference on Data Engineering, ICDE 2021*. IEEE, 468–479. doi:10.1109/ICDE51399.2021.00047
- [23] Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. 2021. Auto-FuzzyJoin: Auto-Program Fuzzy Similarity Joins Without Labeled Examples. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 1064–1076. doi:10.1145/3448016.3452824
- [24] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (2020), 50–60. doi:10.14778/3421424.3421431
- [25] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR abs/1907.11692* (2019). arXiv:1907.11692 <http://arxiv.org/abs/1907.11692>
- [26] Renée J. Miller. 2018. Open Data Integration. *Proc. VLDB Endow.* 11, 12 (2018), 2130–2139. doi:10.14778/3229863.3240491
- [27] Fatemeh Nargesian, Erkang Zhu, Renée J. Miller, Ken Q. Pu, and Patricia C. Arocena. 2019. Data Lake Management: Challenges and Opportunities. *Proc. VLDB Endow.* 12, 12 (2019), 1986–1989. doi:10.14778/3352063.3352116
- [28] Fatemeh Nargesian, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2018. Table Union Search on Open Data. *Proc. VLDB Endow.* 11, 7 (2018), 813–825. doi:10.14778/3192965.3192973
- [29] Paul Ouellette, Aidan Sciortino, Fatemeh Nargesian, Bahar Ghadiri Bashardoost, Erkang Zhu, Ken Q. Pu, and Renée J. Miller. 2021. RONIN: Data Lake Exploration. *Proc. VLDB Endow.* 14, 12 (2021), 2863–2866. doi:10.14778/3476311.3476364
- [30] Matteo Paganelli, Domenico Beneventano, Francesco Guerra, and Paolo Sotovia. 2019. Parallelizing Computations of Full Disjunctions. *Big Data Research* 17 (2019), 18–31. doi:10.1016/j.bdr.2019.07.002
- [31] George Papadakis, Ekaterini Ioannou, and Themis Palpanas. 2020. Entity Resolution: Past, Present and Yet-to-Come. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020*. OpenProceedings.org, 647–650. doi:10.5441/002/edbt.2020.85
- [32] Anand Rajaraman and Jeffrey D. Ullman. 1996. Integrating Information by Outerjoins and Full Disjunctions (Extended Abstract). In *PODS 1996*. ACM.
- [33] Franck Ravat and Yan Zhao. 2019. Data Lakes: Trends and Perspectives. In *Database and Expert Systems Applications - 30th International Conference, DEXA 2019, Linz, Austria, August 26-29, 2019, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 11706)*, Sven Hartmann, Josef Küng, Sharma Chakravarthy, Gabriele Anderst-Kotsis, A Min Tjoa, and Ismail Khalil (Eds.). Springer, 304–313. doi:10.1007/978-3-030-27615-7_23
- [34] Weifeng Su, Jiying Wang, and Frederick H. Lochovsky. 2006. Holistic Schema Matching for Web Query Interfaces. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Proceedings (Lecture Notes in Computer Science, Vol. 3896)*. Springer, 77–94. doi:10.1007/11687238_8
- [35] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *SIGMOD Conference 2019*. ACM, 847–864. doi:10.1145/3299869.3300065
- [36] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-Join: Joining Tables by Leveraging Transformations. *Proc. VLDB Endow.* 10, 10 (2017), 1034–1045. doi:10.14778/3115404.3115409
- [37] Erkang Zhu, Fatemeh Nargesian, Ken Q. Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *Proc. VLDB Endow.* 9, 12 (2016), 1185–1196. doi:10.14778/2994509.2994534