Overlap-Based Duplicate Table Detection

(Discussion Paper)

Luca Zecchini[1,*], Tobias Bleifuß[2], Giovanni Simonini[1], Sonia Bergamaschi[1] and Felix Naumann[2]

[1]*University of Modena and Reggio Emilia, Italy*
[2]*Hasso Plattner Institute, University of Potsdam, Germany*

Abstract

Both the Web and data lakes contain much redundant data in the form of largely overlapping pairs of tables. In many cases, this overlap is not accidental and provides meaningful information about the relatedness of the tables. In particular, we focus on the *largest overlap* between two tables, i.e., their largest common subtable. The largest overlap can help us discover multiple coexisting versions of the same table, which possibly differ in the completeness and correctness of the conveyed information. Automatically detecting these highly similar, *duplicate* tables would allow us to guarantee their consistency through data cleaning or change propagation, but also to eliminate redundancy to free up storage space or to save additional work for the editors. Unfortunately, detecting the largest overlap is a computationally challenging problem, requiring to carefully permute columns and rows.

We introduce therefore Sloth, our solution to efficiently detect the largest overlap between two tables. As we experimentally demonstrate on real-world datasets, Sloth is not only effective in solving this task, but can impact on multiple additional use cases, such as detecting potential copying across sources or automatically discovering candidate multi-column joins.

Keywords

Table Overlap, Table Matching, Related Tables

1. Overlapping Tables

The Web contains a huge amount of structured data in tabular form. In 2008, it was already possible to retrieve more than 14 billion tables [1], and nowadays more than 2 million tables can coexist in the English version of Wikipedia alone [2].

Web tables often have a very dynamic existence. A particularly representative case is that of Wikipedia [2], where tables are frequently edited or updated, moved within their page or to another page, copied to related pages or elsewhere, with frequent episodes of carelessness, conflicts among editors [3], and even vandalism [4]. Because of this dynamism and the heterogeneity of the community of Wikipedia editors, it can be very difficult to guarantee data quality, which is fundamental for an encyclopedia, whose content should always be correct, complete, and updated.

SEBD 2024: 32nd Symposium on Advanced Database Systems, June 23-26, 2024, Villasimius, Sardinia, Italy
*Corresponding author.
luca.zecchini@unimore.it (L. Zecchini); tobias.bleifuss@hpi.de (T. Bleifuß); giovanni.simonini@unimore.it (G. Simonini); sonia.bergamaschi@unimore.it (S. Bergamaschi); felix.naumann@hpi.de (F. Naumann)
0000-0002-4856-0838 (L. Zecchini); 0009-0006-9517-7707 (T. Bleifuß); 0000-0002-3466-509X (G. Simonini); 0000-0001-8087-6587 (S. Bergamaschi); 0000-0002-4483-1389 (F. Naumann)

Team	City	Stadium	Capacity
Arsenal	London	Emirates	60,704
Aston Villa	Birmingham	Villa Park	42,657
Liverpool	Liverpool	Anfield	53,394
Manchester	Manchester	Old Trafford	74,310

Stadium	Opened	Capacity
Anfield	1884	60,704
Craven Cottage	1896	22,384
Emirates	2006	60,704
Old Trafford	1910	74,310
Villa Park	1897	42,657

(a) A pair of tables about football teams and stadiums.

Liverpool	Liverpool	Anfield	53,394	
Arsenal	London	Emirates	60,704	2006
Aston Villa	Birmingham	Villa Park	42,657	1897
Manchester	Manchester	Old Trafford	74,310	1910
		Anfield	60,704	1884
		Craven Cottage	22,384	1896

Largest Overlap

(b) The largest overlap between the two tables.

Figure 1: Two example tables and their largest overlap.

Among the 2.13 million tables existing in Wikipedia at the time of our latest snapshot, we surprisingly discovered that about 6.5 million pairs of tables present an overlap equal to at least half of the area of the smaller table, for an estimated redundancy of 63.49 MB. Even more surprisingly, we detected 5.9 million pairs of coexisting tables with identical content, highlighting the massive diffusion of copy-and-paste practices in Wikipedia [5].

In particular, we focus on the *largest overlap* between the two tables [5], i.e., their largest common *rectangular* subtable, as depicted in Figure 1. In many cases, this overlap is not accidental, but gives meaningful insights about the relatedness of the tables and the quality of their content. The nature of tabular data allows changing the order of columns and rows (Figure 1b), making the detection of the largest overlap computationally challenging.

The ability to detect the largest overlap between two tables, and in particular to retrieve pairs of highly similar tables, defined as *duplicate* or *matching* tables, can lead to several benefits, such as verifying the consistency of the information conveyed by the tables, pointing out cases of incompleteness or inconsistencies. This is not only relevant for Web tables: every scenario where a table can be duplicated at a certain point in time, with an independent development for the different copies, is prone to the insurgence of inconsistencies. For instance, when data scientists retrieve datasets from the enterprise's data lake, perform transformations (e.g., join, wrangling, etc.) for their analysis, then store back the new datasets into the data lake [6].

Depending on the context, a user might desire to ensure the consistency of the information present in duplicate tables through operations of *data cleaning* [7] and *change propagation* [8], or it might be more convenient to directly prevent the rise of inconsistencies by eliminating this redundancy. In fact, avoiding redundancy not only allows to save disk space, but also to lighten the workload for website editors, who would just have to focus on a single consistent table instead of performing every editing multiple times, exposing to the risk of inconsistencies or missed updates. Whatever the purpose, one must first detect such duplicate tables.

While the existing literature widely recognizes the importance of discovering *related* tables [9, 10, 11] on the Web or in data lakes for enriching the conveyed information, proposing many approaches for the efficient detection of *unionable* tables [12, 13, 14] or *joinable* tables [15, 16, 17, 18, 19], the task of detecting duplicate tables is only investigated in some specific or limited scenarios (e.g., to detect the subsequent versions of a table throughout the history of a Wikipedia page [20] or restricted to the basic cases of perfect duplicates and row containment [21]).

To fill this gap, we recently proposed Sloth [5], a novel solution to determine the largest overlap between a given pair of tables, i.e., the maximal contiguous rectangular area of identical cells that can be achieved by reordering columns and rows of both tables.

More formally, given a bijective attribute mapping $M : X_M \subseteq X \rightarrow Y_M \subseteq Y$ defined between two tables $R(X)$ and $S(Y)$, we refer to the *table overlap* $O_M = R[X_M] \uplus S[Y_M]$ as the intersection under the bag semantics (i.e., which allows duplicates) between the bags of tuples obtained through the projection of $R(X)$ on X_M and $S(Y)$ on Y_M. Defined $\mathcal{O}$ as the set of all possible overlaps between the two tables and the *overlap area* $A_M = |X_M| \cdot |O_M|$ as the number of cells contained in the overlap O_M, the set of the *largest overlaps* $\mathcal{O}^* = \{O_{M^*} \in \mathcal{O} \mid A_{M^*} \geq A_M, \forall O_M \in \mathcal{O}\}$ is composed of the overlaps with the maximum area, in most cases just one. All details of our formalization can be found in the full research paper [5].

At the core of Sloth lies the first algorithm designed to detect the largest overlap between two tables. First, our algorithm detects the pairs of attributes across the two tables that share some cell values. By combining these pairs of attributes, it is possible to obtain a complete overview of all potentially non-empty overlaps existing between the tables (i.e., the candidates to be the largest one) in the form of a *lattice* [22]. The combined pairs determine an upper bound for the area of the candidates. Thus, our algorithm aims to exploit this bounding mechanism to prioritize candidates and detect the largest overlap as soon as possible, minimizing the number of candidates for which we need to compute the actual area.

Since this task is computationally challenging, Sloth also relies on a greedy variant for the algorithm based on *beam search* [23, 24] to deal with critical pairs for which the exact technique struggles to produce a result in a reasonable time for the user. Section 2 provides an overview of both algorithms, which are described in detail in the full research paper [5].

Beyond the efficiency of Sloth and the quality of the results produced by the greedy algorithm, through our experimental evaluation we were able to highlight multiple relevant real-world use cases, such as the detection of highly overlapping tables in Wikipedia, the recognition of potential copying across tables from different sources [25], and the automated discovery of candidate multi-column joins in a corpus of relational tables. An excerpt from the full research paper [5] is reported in Section 3, while Section 4 briefly compares to the existing literature. Finally, Section 5 presents the future directions of our research and concludes the paper.

2. Largest Overlap Detection

Figure 2 illustrates the high-level design of Sloth. Implemented in Python, Sloth[1] considers as input two tables $R(X)$ and $S(Y)$ (Figure 2a) and returns the set of their largest overlaps $\mathcal{O}^*$ (Figure 2d). The exact algorithm is our default choice (Figure 2b), with a timeout mechanism

[1] https://github.com/dbmodena/sloth

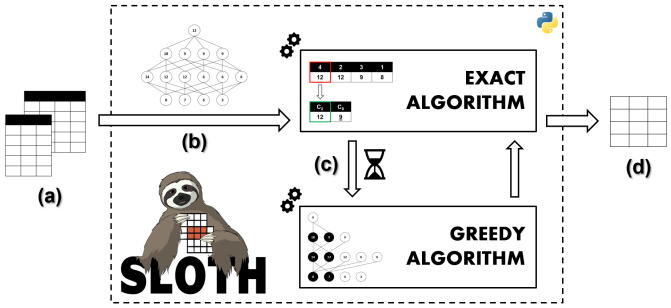


Figure 2: An overview of the Sloth workflow, from a pair of tables (a) to their largest overlap (d) through our detection algorithm (b) and its greedy variant for critical cases (c).

defined to spot critical cases that would not provide a result in a reasonable time, activating its greedy variant (Figure 2c). Both the timeout and the parameter for the greedy algorithm (i.e., the beam width β) can be edited by the users according to their needs (e.g., a faster computation or a better accuracy). Additional parameters can define the minimum area Δ to consider the largest overlap as relevant and even minimum/maximum width/height thresholds.

Our exact algorithm (Algorithm 1) needs to consider all mappings that can potentially determine the largest overlap, denoted as *candidates*. To identify the candidates, our algorithm first considers all possible *single-attribute mappings*, i.e., those mappings for which X_M is represented by a single attribute x, checking the area of the overlap between $R[x]$ and $S[M(x)]$. We call *seeds* those single-attribute mappings whose area is greater than zero, and we collect them in a dedicated list (Line 2) sorted by descending area, as depicted in Figure 3b. All details about the invoked functions can be found in the full research paper [5].

Since every *multi-attribute mapping* is a combination of some single-attribute mappings, the candidates can be considered as the combinations of the seeds and modeled as the nodes of a lattice, as depicted in Figure 3b, where every level n contains the combinations of n seeds. Due to the bijectivity of the mapping, some seeds cannot be combined (e.g., S_0 and S_4 in Figure 3b, both considering the same attribute City from X), hence possibly producing a *semilattice*.

Moving up within the lattice increases the width of the overlap, but not necessarily its area, as its height may decrease as new columns are added. In particular, the seed with the minimum area (equal to its height) in the combination defines an upper bound for the height of the candidate, and therefore for its area. This bounding mechanism can be exploited both to prune the lattice and to prioritize the candidates based on their potential area. We define therefore the *pruning threshold* θ (Line 3), which always contains the maximum between Δ and the maximum actual area of a candidate that we know so far (initially the area of the first seed in the list, then possibly updated every time we verify a new candidate).

Our algorithm exploits the upper bound defined by the seeds to manage two priority queues (i.e., max heap structures), aiming to minimize both the number of candidates that need to be materialized and those among them whose actual area needs to be computed: *(i) Levels* (Line 4), containing the representations of the levels of the lattice, used to generate the candidates incrementally by decreasing potential area; *(ii) Candidates* (Line 5), containing the generated candidates, used to progressively verify their actual area and detect the largest overlap.

Algorithm 1: Largest overlap detection algorithm

Input: Two tables $R(X)$ and $S(Y)$; minimum area Δ (default 0); minimum/maximum width/height
Output: The set of the largest overlaps $\mathcal{O}^*$

1 $\mathcal{O}^* \leftarrow \emptyset$ // largest overlaps
2 *Seeds* $\leftarrow$ **findSeeds**(R, S)
3 $\theta \leftarrow max(\Delta, Seeds[0].A)$ // pruning threshold
4 *Levels* $\leftarrow$ **initLevels**(*Seeds*) // priority queue to generate candidates
5 *Candidates* $\leftarrow$ *maxHeap*$(\emptyset, key = A)$ // priority queue to verify candidates
6 **while** *Candidates* $\neq \emptyset$ **or** *Levels* $\neq \emptyset$ **do**
7 **while** *Candidates.head*$().A <$ *Levels.head*$().A$ **do**
8 *Levels*, *Candidates* $\leftarrow$ **genCand**(*Levels*, *Candidates*, *Seeds*) // generate more candidates
9 **if** *Candidates* $\neq \emptyset$ **then**
10 *topC* $\leftarrow$ *Candidates.pop*() // top candidate
11 **if** *topC.O* $\neq \emptyset$ **then**
12 $\mathcal{O}^* \leftarrow \mathcal{O}^* \cup$ *topC.O* // largest overlap found!
13 **else**
14 *Levels*, *Candidates* $\leftarrow$ **verCand**$(R, S,$ *topC*, *Levels*, *Candidates*$)$ // verify candidate
15 **return** $\mathcal{O}^*$

In particular, we iterate on the priority queues until both of them are emptied (Line 6), terminating early as soon as all largest overlaps are detected. At each iteration, first we need to ensure that at least one of the candidates with the potential largest overlap has been generated and inserted into *Candidates* (Lines 7-8), then we can check the candidate at the top of *Candidates* (Line 10). If it has already been verified (i.e., its actual overlap has already been computed), none of the other candidates (among both the ones already generated and the ones yet to generate) can present a greater area, hence it is one of the largest overlaps, and we can add it to the result set (Line 12); otherwise, we need to compute its overlap (and therefore its actual area) and reinsert it into the priority queue if it can still be part of the result set (Line 14).

Since our exact algorithm generates the candidates by combining the seeds, the detection of a very large number of seeds (e.g., for a pair of wide tables with some values repeated across several columns in both) may produce a huge lattice, making it sometimes impossible to generate the candidates in a reasonable amount of time. For a result as close as possible to the exact largest overlap, we designed therefore a greedy variant inspired by beam search, a heuristic search algorithm that performs a breadth-first search in a tree by only expanding the β most promising nodes at each level, where the parameter β is denoted as beam width.

Our greedy algorithm is designed to bottom-up traverse the lattice generated from the seeds. At the beginning, we consider a maximum of β seeds with the greatest area. To find candidates for the second level, we combine each of them with every other seed and drop repeated and invalid combinations. After this generating step, we verify all new candidates and again select only the β candidates with the greatest actual area among them. For the third and every further level, we combine the selected candidates of the previous level with every seed that is not already part of the candidate and then again verify their area and limit their number to β, while the pruning threshold θ is updated and can determine an early stopping.

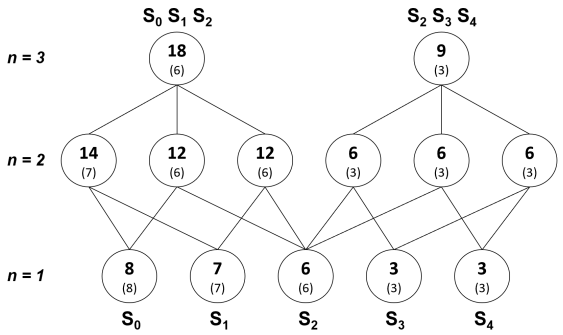
Team	City	Stadium	Capacity
Arsenal	London	Emirates Stadium	60,704
Barcelona	Barcelona	Camp Nou	99,354
Bayern Munich	Munich	Allianz Arena	75,000
Inter Milan	Milan	San Siro	80,018
Liverpool	Liverpool	Anfield	53,394
Manchester United	Manchester	Old Trafford	74,310
Milan	Milan	San Siro	80,018
Real Madrid	Madrid	Santiago Bernabéu	81,044

Team	City	Country	Stadium	Founded
Real Madrid	Madrid	Spain	Santiago Bernabéu	1902
Milan	Milan	Italy	San Siro	1899
Bayern Munich	Munich	Germany	Allianz Arena	1900
Liverpool	Liverpool	England	Anfield	1892
Barcelona	Barcelona	Spain	Camp Nou	1899
Ajax	Amsterdam	Netherlands	Johan Cruyff Arena	1900
Inter Milan	Milan	Italy	Giuseppe Meazza	1908
Manchester United	Manchester	England	Old Trafford	1878
Chelsea	London	England	Stamford Bridge	1905
Juventus	Turin	Italy	Juventus Stadium	1897

(a) Two input tables $R(X)$ and $S(Y)$ about football teams.

Seeds

seed	S_0	S_1	S_2	S_3	S_4
M	City ↓ City	Team ↓ Team	Stadium ↓ Stadium	Team ↓ City	City ↓ Team
A	8	7	6	3	3



(b) The sorted list of detected seeds and the valid candidates in the semilattice generated from the seeds. For every node we report the upper bounds for its area and its height (between round brackets).

Figure 3: Two input tables, the sorted list of detected seeds, and the lattice of valid candidates.

3. Experimental Evaluation

Our experiments, whose configurations and results are reported and discussed in detail in the full research paper [5], aim to evaluate the performance of Sloth (considering both the exact algorithm and its greedy variant) and its application to three real-world use cases.

In Table 1, we present the datasets used in our evaluation. We denote as wiki_history the Wikipedia table matching dataset from the IANVS project[2], which captures the evolution of all 3.5M tables present in the English Wikipedia throughout its entire history (until September 1, 2019). We separately consider the most recent snapshot from this dataset, denoted as wiki_latest. Beyond the Wikipedia scenario, we employ uni_dwh [16], a real-world university data warehouse, and two datasets[3] reporting the information about stock symbols and flights captured from different sources across multiple days [25], both in their original version (i.e., *raw*) and in the one obtained through schema alignment (i.e., *clean*).

First, we evaluate the performance of Sloth on a representative subset of the wiki_history dataset and the uni_dwh dataset, chosen to cover both the case of Web tables and the one of relational database tables in our analysis. Our experiments [5] confirm that the main factor

[2]https://hpi.de/naumann/projects/data-profiling-and-analytics/change-exploration.html
[3]https://lunadong.com/fusionDataSets.htm

Table 1
Statistics about the number and the size of the tables appearing in the used datasets.

Dataset	#D	Width (#columns)			Height (#rows)		
		MIN	MAX	AVG	MIN	MAX	AVG
wiki_history	55.97M	1	5694	5.92	1	17.38k	26.63
wiki_latest	2.13M	1	883	5.23	1	4670	11.47
uni_dwh	158	1	55	9.48	2	151.78k	5604.79
stock_raw	1.15k	4	69	16.18	221	1000	987.58
stock_clean	1.15k	3	17	12.49	221	1000	987.58
flight_clean	1.17k	3	7	5.75	6	1309	662.17

Table 2
Types of overlapping tables in Wikipedia.

Type	#Table Pairs	Estimated Size
Perfect duplicates	5.91M (85.521%)	39.55 MB
Containment	351.24k (5.085%)	6.05 MB
Additional rows	59.75k (0.865%)	4.35 MB
Additional columns	289.97k (4.198%)	1.91 MB
Additional rows and columns	1.53k (0.022%)	0.56 MB
Partial overlaps	648.91k (9.394%)	39.16 MB
$\geq$ *50% of the smallest table*	252.80k (3.660%)	35.52 MB
$<$ *50% of the smallest table*	396.12k (5.735%)	6.60 MB
Total ($\geq$ 50%)	**6.51M (94.265%)**	**63.49 MB**

leading to timeouts is the number of seeds, which directly affects the size of the lattice, hence the number of candidates that potentially need to be generated. This aspect is strictly correlated to the width of the tables and the amount of repeated cell values across them. Further, we analyze the impact on the runtime of the different tasks (i.e., seed detection, candidate generation, and candidate verification) for both the exact and the greedy algorithm, and also of the beam width for the latter. We also show the impact of the table height on the seed detection runtime and how our similarity estimation significantly differs from two widely adopted similarity metrics based on set semantics, such as Jaccard similarity and overlap set similarity [17, 26]. Finally, we evaluate the accuracy of the greedy algorithm, showing that, even in absence of approximation guarantees on the quality of its result, it is generally able to detect largest overlaps of the same area as those discovered by the exact algorithm.

Moving to the real-world use cases, first we use Sloth to quantify the amount of largely overlapping pairs of tables coexisting in Wikipedia, using wiki_latest. The results, reported in Table 2, highlight the redundancy of the information conveyed by Wikipedia tables and the diffusion of copy-and-paste practices in the encyclopedia. Then, we show that Sloth can be used to detect potential copying across multiple sources, allowing us to retrieve all clusters of sources with declared copying dependencies in the stock and flight datasets [25] with a minimum effort, without the need for schema alignment, also leading to the discovery of meaningful additional sources. Finally, we use Sloth to automatically detect candidate multi-column joins in uni_dwh, a task not supported by any of the existing solutions [5], as highlighted by showing the limitations of a simple adaptation of Josie [17] in such a scenario.

4. Related Work

Even if a plethora of algorithms have been designed to efficiently discover related tables [9, 10, 11], especially unionable [12, 13, 14] and joinable ones [15, 16, 17, 18, 19], none of them can exactly compute the largest overlap between two tables.

For instance, Josie [17] considers a join column from a query table and finds the top-k columns whose set of cell values presents the largest intersection with the one of the join column. Since it operates on single columns using the set semantics, it is impossible to use it for detecting the largest overlap. At most, an adaptation considering entire tables under the bag semantics would produce an upper bound for the largest overlap, so it might be used to enhance scalability by passing only the most promising pairs to Sloth. Similarly, Mate [19] is the only system supporting the discovery of multi-column joins (using a dedicated hashing function named XASH), but it requires to provide a set of columns as input. Since the set that yields the largest overlap is not known up-front, it cannot be easily employed as an alternative to Sloth.

Moving to the previous approaches to duplicate table detection, Bleifuß et al. [20] aim to find matching tables across subsequent versions of a Wikipedia page. Their solution (i.e., a multi-stage matching process based on Jaccard similarity) is designed for one-to-one matches among a limited number of tables and exploits specific aspects such as the position of the tables inside the page, hence it is not generalizable. Koch et al. [21] use instead XASH and define two tables as duplicates if they contain the same set of tuples, only tackling the cases of perfect duplicates or row containment.

5. Conclusion and Future Work

We presented Sloth, a method to efficiently determine the largest overlap between two tables, allowing to detect duplicate tables. This leads to several benefits both on the Web and in data lakes. For instance, it allows spotting and solving common data quality issues, such as inconsistent or incomplete information. Also, it helps eliminate redundancy to free up storage space or to save additional work for the editors, preventing the insurgence of data quality problems. Through our experimental evaluation we assessed the performance of Sloth in real-world scenarios, considering Web tables from Wikipedia and relational tables from a data warehouse, up to use cases such as the detection of potential copying across multiple sources and the discovery of candidate multi-column joins in a corpus of relational tables.

Moving beyond the results presented in this paper, we plan to broaden our research in multiple directions. First, we want to design updatable indexes to enable overlap-based duplicate table detection at scale, allowing users to provide a table as a query and retrieve the top-k tables presenting the largest overlap with it. Then, we want to enable Sloth to detect not only the largest, but also the *best* overlap between two tables, defining quality metrics to capture how meaningful an overlap is. Finally, we plan to evaluate the use of table embeddings [27] to estimate the largest overlap and to analyze the impact of table deduplication on the performance of tabular language models [28], similarly to what has already been demonstrated for their textual counterparts [29].

References

[1] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, Y. Zhang, WebTables: Exploring the Power of Tables on the Web, Proceedings of the VLDB Endowment (PVLDB) 1 (2008) 538–549. doi:10.14778/1453856.1453916.

[2] T. Bleifuß, L. Bornemann, D. V. Kalashnikov, F. Naumann, D. Srivastava, The Secret Life of Wikipedia Tables, in: Proceedings of the Workshop on Search, Exploration, and Analysis in Heterogeneous Datastores (SEA Data @ VLDB), 2021, pp. 20–26. URL: https://ceur-ws.org/Vol-2929/paper4.pdf.

[3] S. Bykau, F. Korn, D. Srivastava, Y. Velegrakis, Fine-Grained Controversy Detection in Wikipedia, in: Proceedings of the IEEE International Conference on Data Engineering (ICDE), 2015, pp. 1573–1584. doi:10.1109/ICDE.2015.7113426.

[4] M. Potthast, B. Stein, R. Gerling, Automatic Vandalism Detection in Wikipedia, in: Proceedings of the European Conference on Information Retrieval (ECIR), 2008, pp. 663–668. doi:10.1007/978-3-540-78646-7_75.

[5] L. Zecchini, T. Bleifuß, G. Simonini, S. Bergamaschi, F. Naumann, Determining the Largest Overlap between Tables, Proceedings of the ACM on Management of Data (PACMMOD) 2 (2024) 48:1–48:26. doi:10.1145/3639303.

[6] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, P. C. Arocena, Data Lake Management: Challenges and Opportunities, Proceedings of the VLDB Endowment (PVLDB) 12 (2019) 1986–1989. doi:10.14778/3352063.3352116.

[7] I. F. Ilyas, X. Chu, Data Cleaning, 2019. doi:10.1145/3310205.

[8] T. Bleifuß, L. Bornemann, T. Johnson, D. V. Kalashnikov, F. Naumann, D. Srivastava, Exploring Change: A New Dimension of Data Analytics, Proceedings of the VLDB Endowment (PVLDB) 12 (2018) 85–98. doi:10.14778/3282495.3282496.

[9] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, C. Yu, Finding Related Tables, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), 2012, pp. 817–828. doi:10.1145/2213836.2213962.

[10] A. Bogatu, A. A. A. Fernandes, N. W. Paton, N. Konstantinou, Dataset Discovery in Data Lakes, in: Proceedings of the IEEE International Conference on Data Engineering (ICDE), 2020, pp. 709–720. doi:10.1109/ICDE48307.2020.00067.

[11] Y. Zhang, Z. G. Ives, Finding Related Tables in Data Lakes for Interactive Data Science, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), 2020, pp. 1951–1966. doi:10.1145/3318464.3389726.

[12] M. J. Cafarella, A. Halevy, N. Khoussainova, Data Integration for the Relational Web, Proceedings of the VLDB Endowment (PVLDB) 2 (2009) 1090–1101. doi:10.14778/1687627.1687750.

[13] O. Lehmberg, C. Bizer, Stitching Web Tables for Improving Matching Quality, Proceedings of the VLDB Endowment (PVLDB) 10 (2017) 1502–1513. doi:10.14778/3137628.3137657.

[14] F. Nargesian, E. Zhu, K. Q. Pu, R. J. Miller, Table Union Search on Open Data, Proceedings of the VLDB Endowment (PVLDB) 11 (2018) 813–825. doi:10.14778/3192965.3192973.

[15] E. Zhu, F. Nargesian, K. Q. Pu, R. J. Miller, LSH Ensemble: Internet-Scale Domain Search, Proceedings of the VLDB Endowment (PVLDB) 9 (2016) 1185–1196. doi:10.

14778/2994509.2994534.
[16] R. Castro Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, M. Stonebraker, Aurum: A Data Discovery System, in: Proceedings of the IEEE International Conference on Data Engineering (ICDE), 2018, pp. 1001–1012. doi:10.1109/ICDE.2018.00094.
[17] E. Zhu, D. Deng, F. Nargesian, R. J. Miller, JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), 2019, pp. 847–864. doi:10.1145/3299869.3300065.
[18] Y. Dong, K. Takeoka, C. Xiao, M. Oyamada, Efficient Joinable Table Discovery in Data Lakes: A High-Dimensional Similarity-Based Approach, in: Proceedings of the IEEE International Conference on Data Engineering (ICDE), 2021, pp. 456–467. doi:10.1109/ICDE51399.2021.00046.
[19] M. Esmailoghli, J. Quiané-Ruiz, Z. Abedjan, MATE: Multi-Attribute Table Extraction, Proceedings of the VLDB Endowment (PVLDB) 15 (2022) 1684–1696. doi:10.14778/3529337.3529353.
[20] T. Bleifuß, L. Bornemann, D. V. Kalashnikov, F. Naumann, D. Srivastava, Structured Object Matching across Web Page Revisions, in: Proceedings of the IEEE International Conference on Data Engineering (ICDE), 2021, pp. 1284–1295. doi:10.1109/ICDE51399.2021.00115.
[21] M. Koch, M. Esmailoghli, S. Auer, Z. Abedjan, Duplicate Table Detection with Xash, in: Proceedings of the Conference on Database Systems for Business, Technology and Web (BTW), 2023, pp. 367–390. doi:10.18420/BTW2023-18.
[22] G. Birkhoff, Lattice Theory, 1940. doi:10.1090/coll/025.
[23] B. T. Lowerre, The HARPY Speech Recognition System, Ph.D. thesis, Carnegie Mellon University, 1976.
[24] X. Huang, J. Baker, R. Reddy, A Historical Perspective of Speech Recognition, Communications of the ACM (CACM) 57 (2014) 94–103. doi:10.1145/2500887.
[25] X. Li, X. L. Dong, K. Lyons, W. Meng, D. Srivastava, Truth Finding on the Deep Web: Is the Problem Solved?, Proceedings of the VLDB Endowment (PVLDB) 6 (2012) 97–108. doi:10.14778/2535568.2448943.
[26] D. Deng, Y. Tao, G. Li, Overlap Set Similarity Joins with Theoretical Guarantees, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), 2018, pp. 905–920. doi:10.1145/3183713.3183748.
[27] R. Cappuzzo, P. Papotti, S. Thirumuruganathan, Creating Embeddings of Heterogeneous Relational Datasets for Data Integration Tasks, in: Proceedings of the ACM International Conference on Management of Data (SIGMOD), 2020, pp. 1335–1349. doi:10.1145/3318464.3389742.
[28] G. Badaro, M. Saeed, P. Papotti, Transformers for Tabular Data Representation: A Survey of Models and Applications, Transactions of the Association for Computational Linguistics (TACL) 11 (2023) 227–249. doi:10.1162/tacl_a_00544.
[29] K. Lee, D. Ippolito, A. Nystrom, C. Zhang, D. Eck, C. Callison-Burch, N. Carlini, Deduplicating Training Data Makes Language Models Better, in: Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), 2022, pp. 8424–8445. doi:10.18653/v1/2022.acl-long.577.