DIALITE: Discover, Align and Integrate Open Data Tables

Aamod Khatiwada
Northeastern University
Boston, Massachusetts, USA
khatiwada.a@northeastern.edu

Roee Shraga
Northeastern University
Boston, Massachusetts, USA
r.shraga@northeastern.edu

Renée J. Miller
Northeastern University
Boston, Massachusetts, USA
miller@northeastern.edu

ABSTRACT

We demonstrate a novel table discovery pipeline called DIALITE that allows users to discover, integrate and analyze open data tables. DIALITE has three main stages. First, it allows users to discover tables from open data platforms using state-of-the-art table discovery techniques. Second, DIALITE integrates the discovered tables to produce an integrated table. Finally, it allows users to analyze the integration result by applying different downstreaming tasks over it. Our pipeline is flexible such that the user can easily add and compare additional discovery and integration algorithms.

CCS CONCEPTS

• **Information systems** → **Information integration**.

KEYWORDS

Data Lakes, Data Discovery, Data Integration, Full Disjunction

ACM Reference Format:
Aamod Khatiwada, Roee 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-Companion '23), June 18–23, 2023, Seattle, WA, USA.* ACM, Seattle, WA, USA, 4 pages. https://doi.org/10.1145/3555041.3589732

1 INTRODUCTION

Data discovery has become an important component in data science pipeline. The discovery process uses techniques such as keyword search [13] and table search [4, 7, 9] to discover a set of tables (datasets). Data scientists use such tables to support decision-making processes, train machine learning models, perform statistical analysis, and so on. After discovery, a natural step is to integrate the discovered tables. An integrated table provides a unified view of the data and allows users to run queries and analyses that go beyond a single table.

While integrating tables, we intend to combine tuples from different tables in a maximal way such that the integrated tuples carry as much information as possible. This enriches the analysis and decision-making process after integration and also improves the quality of downstream applications. The widely known outer-join [12] operator is not associative and does not aim to maximize the connections among the integrated tuples [6]. Accordingly, *Full Disjunction (FD)* [6] has been understood as a natural way of assembling partial pieces of information (facts) such that it maximizes the connections among these facts [11]. FD can be viewed as an associative version of outer join [2] and has been used to integrate information across relational tables [2, 10] and web tables [10]. In a recent paper [8], we proposed ALITE, a new algorithm based on Full Disjunction to integrate tables discovered in data lakes. ALITE was shown to be correct and faster than the existing FD algorithms [2, 10] while integrating real open data lake tables in practice [8]. Additional details on other integration operators and their comparison against FD are available in the full ALITE paper [8] where FD was shown to be a better semantics for integration because it produces a result over which a downstream task like entity-resolution performs more accurately.

In this demonstration, we propose DIALITE, a novel system that discovers, aligns, and integrates open data tables. It extends the aforementioned ALITE [8], which does not perform discovery but instead takes a set of tables as input, aligns the matching columns using holistic schema matching, and applies the FD to get an integrated table. DIALITE offers state-of-the-art systems for different table discovery tasks [7, 14] before applying ALITE to integrate them. Furthermore, DIALITE also offers new downstream analytics (that have not been previously considered) to evaluate the quality of integration. Specifically, DIALITE allows users to

(1) Upload a table or randomly generate one using GPT-3 as a query and discover related tables from a given data lake that are unionable [7] or joinable [14] with the query table. Apart from the available table discovery algorithms, DIALITE also allows users to add new algorithms for table search based on their preference.
(2) Integrate the discovered tables (or upload a set of tables to be integrated) using a novel table integration system called ALITE [8]. Besides ALITE, we allow users to add new integration operators within our extendible architecture.
(3) Analyze and compare the table integrated using ALITE against alternative integration techniques by performing downstream applications. In our demo, we consider outer join as an alternative integration technique (or other queries/methods added by the user) and present data analytics (common aggregations and statistics) and entity-resolution as downstream applications. Both, when applied over real tables (which can be incomplete) will show the dramatic difference between maximally integrating information using FD vs. using outer joins.

Related Work. To the best of our knowledge, DIALITE is the first system that enables a full table search pipeline starting from

SIGMOD-Companion '23, June 18–23, 2023, Seattle, WA, USA

ACM ISBN 978-1-4503-9507-6/23/06…$15.00
https://doi.org/10.1145/3555041.3589732

table discovery, followed by table integration and downstream applications over the integrated table. An earlier system called COCOA [3] focuses on finding joinable tables with correlated attributes that expand the query table. After finding the tables, COCOA applies LEFT JOIN between the query table and the discovered table as the integration operator. DIALITE, on the other hand, finds a set of related tables that can be integrated with the query table (including unionable and joinable tables). Auctus [1] also discovers related tables and integrates table pairs by applying inner join or union. Unlike Auctus, DIALITE does not limit the number of tables to be integrated. In addition, DIALITE uses FD to integrate the tables, which has been shown in theory to maximize the connections among the tuples in the tables [11].

Existing table discovery techniques search for joinable, unionable, or related tables. **JOSIE** [14] and **LSH Ensemble** [15], for instance, take a query table as input with a query column marked by the user and returns a set of top-k tables that are joinable with the query table. **SANTOS** [7] takes a query table and discovers a set of top-k semantically unionable tables as output. Other table search techniques [4, 9] also focus on table discovery from large repositories. But the important question of how to integrate the discovered tables is not addressed. **ALITE** [8] provides a solution to this question. As the data lake tables may lack consistent and meaningful column headers, ALITE applies holistic schema matching over the set of searched tables and assigns a dummy column header called an Integration ID to the set of matching columns. Then, it applies a natural FD over the integration IDs to integrate the tables. The integration process outputs an integrated table with maximally integrated tuples [6, 11].

2 SYSTEM DESCRIPTION

DIALITE, outlined in Fig. 1, has three stages (discover, align & integrate, and analyze) which are included in our demo plan (Sec. 3).

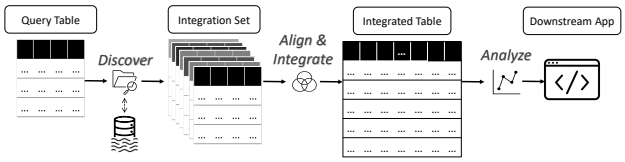

Figure 1: An overview of DIALITE.

DIALITE offers various options for discovery, alignment, integration, and analysis, including the ability for users to implement their own methods. The pipeline begins with a user-provided query table Q. However, we also allow the user to randomly generate a table using GPT-3 [5]. We now detail the components.

2.1 Discover

Given a query table Q, DIALITE uses a data discovery method to find tables in a data lake (table repository) $\mathcal{D}$ that are unionable [7], joinable [14], or simply semantically similar. DIALITE allows the user to choose among existing discovery algorithms including SANTOS [7] and LSH Ensemble [15]. Alternatively, a user can use their own discovery algorithm by implementing a similarity method between two tables. As we apply ALITE [8] for alignment and integration, the discovery phase is agnostic to the type of search. ALITE, aiming to maximize the connections among the tables, decides how the tables should be integrated. The output of discovery is an *integration set* $D \subseteq \mathcal{D}$, a set of tables to be integrated, including the query table. Our discovery techniques allow users to control the number of tables returned, and our system permits users to select a subset of the discovered tables to be integrated.

2.2 Align and Integrate

Given an integration set D, DIALITE uses ALITE [8] to integrate the tables. The integration set can be derived from the discovery part (Sec. 2.1) or provided as input by the user. The latter represents a traditional data integration scenario where the integration set is given. As shown in Fig. 1, this stage outputs an integrated table.

ALITE is composed of two main parts, namely align and integrate. Note that we do not require reliable column headers in the integration set. The first part of ALITE (Align) applies holistic schema matching to identify common columns in the integration set. The matched columns are given the same integration ID. Using these ids as attribute names, the integrate part applies (natural) FD. Specifically, ALITE uses a new holistic schema matching algorithm that was shown to outperform state-of-the-art matchers. ALITE uses a novel algorithm to compute the FD shown to be correct and efficient on real tables (including tables having nulls) [8]. DIALITE also allows users to add alternative integration operators, e.g., outer join, which is included for demonstration.

2.3 Analyze

Given an integrated table (that can also be uploaded as a file by the user), DIALITE allows the user to explore the benefits of integration. Specifically, the user can choose a downstream application to apply over the integrated table. A simple application is an aggregation query that can be applied over the integrated table as we illustrate in Sec. 3.1. In Sec. 3.2 we also present a more complex downstream application, entity-resolution (ER).

2.4 Implementation

DIALITE is implemented in Python 3.8 and the demonstration uses a web application.[1] Within our pipeline, we use SANTOS [2], LSH Ensemble [3] and ALITE [4] using their publicly available code. Also, we use the *py_entitymatching* package to show ER as a downstream application.[5] We allow users to interact with the system after each step so that they can validate the intermediate results.

3 DEMONSTRATION PLAN

Next, we illustrate our demonstration. The link to a demonstration video is available in the github repository.[1] Our demo contains two parts. First, we present the use case of the pipeline as described in Sec. 2. This part is actually composed of three demonstration items, that can be demonstrated independently. Then, we demonstrate DIALITE's extensibility to new algorithms for discovery, integration, and analysis.

[1] https://github.com/northeastern-datalab/dialite
[2] https://github.com/northeastern-datalab/santos
[3] https://github.com/ekzhu/datasketch
[4] https://github.com/northeastern-datalab/alite
[5] https://github.com/anhaidgroup/py_entitymatching

3.1 DIALITE Use case

Discovery. The users of DIALITE would be able to upload a query table in CSV format from an existing pool of tables.[6] Note that we will provide a data lake for the users to use in the demonstration. The tables in this data lake are real tables from open data and are currently preprocessed for our discovery algorithms [7, 15]. Specifically, the indexes used in SANTOS [7] and LSH Ensemble [15] are built offline, i.e., they are already available for the user to use. The users can easily preprocess and link their own data lake.

The set of discovered tables by all the table discovery systems is stored for the next step. As there may be an overlap in unionable and joinable search results, we persist *the set of tables* found by all techniques to form an integration set.

T_1 (Query Table)

TID	Country	City	Vaccination Rate (1+ dose)
t_1	Germany	Berlin	63%
t_2	England	Manchester	78%
t_3	Spain	Barcelona	82%

T_2 (Retrieved unionable table)

TID	Country	City	Vaccination Rate (1+ dose)
t_4	Canada	Toronto	83%
t_5	Mexico	Mexico City	±
t_6	USA	Boston	62%

T_3 (Retrieved joinable table)

TID	City	Total Cases	Death Rate (per 100k residents)
t_7	Berlin	1.4M	147
t_8	Barcelona	2.68M	275
t_9	Boston	263k	335
t_{10}	New Delhi	2M	158

Figure 2: Tables detailing COVID-19 cases in different places. The symbol ± represents null values present in the input tables ("*missing nulls*").

Example 1. *Consider tables T_1, T_2 and T_3 about COVID-19 cases in different places shown in Fig. 2. Here, TID (Tuple ID) is not a real data column and it is added only to refer to the tuples. Let, T_1 be the query table, and tables T_2 and T_3 reside in the data lake repository. In our demo, a user selects* `City` *as an intent column and query column to search for a unionable table using SANTOS [7] and a joinable table using LSH Ensemble [15] respectively. Please see respective papers for additional details on the search algorithms. Let, T_2 and T_3 be results of the unionable and joinable searches respectively. So the result of the discovery step is an integration set of tables T_1, T_2, and T_3. Note that the names of columns are presented for simplicity and are not used by the discovery techinques, which are designed for the ambiguty of data lakes, i.e., unreliable/missing metadata.*

Align and Integrate. Once the integration set is formed, DIALITE allows user to apply ALITE's holistic schema matching to generate integration IDs. Over such IDs, we apply ALITE's FD.

Example 2. *Consider the integration set of tables T_1, T_2 and T_3 shown in Fig. 2 is formed after table discovery step as illustrated in Ex. 1. DIALITE applies ALITE's integration algorithm over these tables that returns an integrated table as shown in Fig. 3. The integration semantics is explained in ALITE paper [8].*

[6]For the demonstration itself, we also allow users to randomly generate a query table (see Sec. 3.2). Also, a user may choose to upload their query table noting that it may be off-topic wrt the data lake, which may yield no results.

$FD(T_1, T_2, T_3)$ (Integrated Table by ALITE)

OID	TIDs	Country	City	Vaccination Rate (1+ dose)	Total Cases	Death Rate (per 100k residents)
f_1	$\{t_1, t_7\}$	Germany	Berlin	63%	1.4M	147
f_2	$\{t_2\}$	England	Manchester	78%	⊥	⊥
f_3	$\{t_3, t_8\}$	Spain	Barcelona	82%	2.68M	275
f_4	$\{t_4\}$	Canada	Toronto	83%	⊥	⊥
f_5	$\{t_5\}$	Mexico	Mexico City	±	⊥	⊥
f_6	$\{t_6, t_9\}$	USA	Boston	62%	263k	335
f_7	$\{t_{10}\}$	⊥	New Delhi	⊥	2M	158

Figure 3: Result of applying ALITE over the tables in Fig. 2. The symbol ⊥ represents the null values produced during the integration due to missing information ("*produced nulls*").

Analyze. After integration, the next feature that ALITE offers is the analysis of the integrated table. We show the use of an aggregation query over the integrated table.

Example 3. *With the integrated table, we now allow the user to use queries that go beyond the single tables. For example, over the integrated table (Fig. 3), the user can find that Boston is the city with the lowest vaccination rate and Toronto has the highest. Trying to understand the reason for that, the user may explore the relationship between vaccination rates (given in T_1 and T_2), number of cases and death rates (given in T_3). For example, the user can compute the correlation between vaccination and death rates that shows a positive (pearson) correlation of 0.16 and (somewhat surprising) correlation of 0.9 between case numbers and vaccination rates. While a bit counter-intuitive, the analysis reveals an interesting insight about the nature of vaccinations, suggesting that in cities with higher death rates and more cases, the government is focusing on vaccination programs and the people are more willing to vaccinate.*

3.2 DIALITE Extendibility

As described in Sec. 2, the user can extend DIALITE by implementing their own alternative components in the pipeline. Specifically, we aim to demonstrate the ability of users to implement (using python code) new discovery algorithms, integration methods and perform their required analysis.

Example 4. *For illustration, a sample code snippet to add a new joinable table discovery algorithm is provided in Fig. 4. The user basically implements a similarity function between two datasets (df1 and df2) that is used by DIALITE for table discovery.*

```
def new_joinability_discovery_algorithm(df1, df2):
    join_df = pd.merge(df1, df2, how ='inner')
    return len(join_df)/max(len(df1), len(df2))
```

Figure 4: Implementing user-defined discovery algorithm based on inner join.

We also consider a scenario where the user may not have a query table to start the analysis. So, DIALITE allows users to use simple prompts to generate the query table for the analysis. We demonstrate this feature using GPT-3 based implementation [5] and allow the user to generate a query table based on prompt as

illustrated in Fig. 5. Here, we generate a query table about COVID-19 cases that has 5 columns and 5 rows.

```
query_table = dialite.randomly_generate_query_table(
    'a table about covid with 5 columns and 5 rows'
    )
query_table.head(5)
```

✓ 10.1s

	Country	Cases	Deaths	Recovered	Active
0	USA	5742812	178701	2633567	2930544
1	Brazil	3713876	116476	2788841	808559
2	India	3444061	61529	2643788	738744
3	Russia	982822	16841	745930	219051
4	Mexico	704016	73814	442309	187893

Figure 5: A code snippet to generate query table using GPT-3.

Furthermore, the users can add an alternative integration operator over the default integration system (ALITE). For example, Fig. 6 shows a user-defined code snippet that implements the commonly used outer-join operator for integration. In the demonstration, we illustrate the benefit of using ALITE over the standard outer join as shown in the following example.

```
def new_outer_join_integration_algorithm(integration_set):
        table1_loc = integration_set.pop()
        table1 = pd.read_csv(table1_loc)
        for table2_loc in integration_set:
                table2 = pd.read_csv(table2_loc)
                table1 = table1.merge(table2, how = "outer")
        return table1
```

Figure 6: Implementing outer join as an integration operator.

Example 5. *Consider tables T_4, T_5 and T_6 shown in Fig. 7 (a) forms an integration set after table discovery where, the tables describe the COVID-19 vaccines, their country of origin, and the regulatory agency that approved the vaccines. For sake of illustration, assume that a user used outer-join as an alternative integration algorithm (see Fig. 6). The results of applying outer-join and ALITE (FD) over these tables are shown in Fig. 8 (a) and Fig. 8 (b) respectively. Now let us assume that the user wants to apply Entity Resolution (ER) as a downstream application by applying py_entitymatching.[5] This analysis over outer join and FD results are shown in Fig. 8(c) and Fig. 8(d), respectively. Outer join produces more output tuples than FD; yet, it does not produce any tuple containing the agency that approved the Johnson & Johnson (J&J) vaccine. FD, on the other hand, produces an output tuple f_{13} that provides this information (as it can be produced using t_{13} and t_{15}). Furthermore, since outer join produces incomplete tuples, ER can not resolve f_9 and f_{10}. This shows an advantage of using the default ALITE operator instead of outer join to integrate the tables.*

ACKNOWLEDGMENTS

This work was supported in part by NSF under award numbers IIS-1956096 and IIS-2107248.

T_4

TID	Vaccine	Approver
t_{11}	Pfizer	FDA
t_{12}	JnJ	±

T_5

TID	Country	Approver
t_{13}	United States	FDA
t_{14}	USA	±

T_6

TID	Vaccine	Country
t_{15}	J&J	United States
t_{16}	JnJ	USA

Figure 7: An integration set of Tables about COVID-19 vaccines, their country of origin and their approvers.

$T_4 \bowtie T_5 \bowtie T_6$

OID	TIDs	Vaccine	Approver	Country
f_8	$\{t_{11}, t_{13}\}$	Pfizer	FDA	United States
f_9	$\{t_{12}\}$	JnJ	±	⊥
f_{10}	$\{t_{14}\}$	⊥	±	USA
f_{11}	$\{t_{15}\}$	J&J	⊥	United States
f_{12}	$\{t_{16}\}$	JnJ	⊥	USA

(a) Output Table generated using outer join

Vaccine	Approver	Country
Pfizer	FDA	United States
JnJ	±	⊥
⊥	±	USA
J&J	⊥	United States

(c) Entity Resolution over outer join result

FD(T_4, T_5, T_6)

OID	TIDs	Vaccine	Approver	Country
f_8	$\{t_{11}, t_{13}\}$	Pfizer	FDA	United States
f_{12}	$\{t_{16}\}$	JnJ	⊥	USA
f_{13}	$\{t_{13}, t_{15}\}$	J&J	FDA	United States

(b) Output Table generated using FD (ALITE)

Vaccine	Approver	Country
Pfizer	FDA	United States
J&J	FDA	United States

(d) Entity Resolution over FD (ALITE) result

Figure 8: Integrating tables in Fig. 7 using outer join and FD.

REFERENCES

[1] Sonia Castelo, Rémi Rampin, Aécio S. R. Santos, Aline Bessa, Fernando Chirigati, and Juliana Freire. 2021. Auctus: A Dataset Search Engine for Data Discovery and Augmentation. *Proc. VLDB Endow.* 14, 12 (2021), 2791–2794.

[2] Sara Cohen, Itzhak Fadida, Yaron Kanza, Benny Kimelfeld, and Yehoshua Sagiv. 2006. Full Disjunctions: Polynomial-Delay Iterators in Action. In *VLDB 2006*. ACM. http://dl.acm.org/citation.cfm?id=1164191

[3] Mahdi Esmailoghli, Jorge-Arnulfo Quiané-Ruiz, and Ziawasch Abedjan. 2021. COCOA: COrrelation COefficient-Aware Data Augmentation. In *EDBT 2021*. 331–336. https://doi.org/10.5441/002/edbt.2021.30

[4] 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. *Proc. VLDB Endow.* 16, 7 (2023), 1726–1739. https://doi.org/10.14778/3574245.3574274

[5] Luciano Floridi and Massimo Chiriatti. 2020. GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds Mach.* 30, 4 (2020), 681–694.

[6] César A. Galindo-Legaria. 1994. Outerjoins as Disjunctions. In *SIGMOD Conference 1994*. ACM, 348–358. https://doi.org/10.1145/191839.191908

[7] Aamod Khatiwada, Grace Fan, Roee 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. https://doi.org/10.1145/3588689

[8] Aamod Khatiwada, Roee Shraga, Wolfgang Gatterbauer, and Renée J. Miller. 2022. Integrating Data Lake Tables. *Proc. VLDB Endow.* 16, 4 (2022), 932–945. https://doi.org/10.14778/3574245.3574274

[9] 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. https://doi.org/10.14778/3192965.3192973

[10] Matteo Paganelli, Domenico Beneventano, Francesco Guerra, and Paolo Sottovia. 2019. Parallelizing Computations of Full Disjunctions. *Big Data Research* 17 (2019), 18–31. https://doi.org/10.1016/j.bdr.2019.07.002

[11] Anand Rajaraman and Jeffrey D. Ullman. 1996. Integrating Information by Outerjoins and Full Disjunctions (Extended Abstract). In *PODS 1996*. ACM.

[12] Raghu Ramakrishnan and Johannes Gehrke. 2003. *Database management systems (3. ed.)*. McGraw-Hill.

[13] Roee Shraga, Haggai Roitman, Guy Feigenblat, and Mustafa Canim. [n.d.]. Web Table Retrieval using Multimodal Deep Learning. In *SIGIR conference 2020*. ACM.

[14] 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. https://doi.org/10.1145/3299869.3300065

[15] 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. https://doi.org/10.14778/2994509.2994534