Explain3D: Explaining Disagreements in Disjoint Datasets

Xiaolan Wang
University of Massachusetts, Amherst
College of Information and Computer Sciences
xlwang@cs.umass.edu

Alexandra Meliou
University of Massachusetts, Amherst
College of Information and Computer Sciences
ameli@cs.umass.edu

ABSTRACT

Data plays an important role in applications, analytic processes, and many aspects of human activity. As data grows in size and complexity, we are met with an imperative need for tools that promote understanding and explanations over data-related operations. Data management research on explanations has focused on the assumption that data resides in a single dataset, under one common schema. But the reality of today's data is that it is frequently unintegrated, coming from different sources with different schemas. When different datasets provide different answers to semantically similar questions, understanding the reasons for the discrepancies is challenging and cannot be handled by the existing single-dataset solutions.

In this paper, we propose explain3D, a framework for explaining the **d**isagreements across **d**isjoint **d**atasets (3D). Explain3D focuses on identifying the reasons for the differences in the results of two semantically similar queries operating on two datasets with potentially different schemas. Our framework leverages the queries to perform a semantic mapping across the relevant parts of their provenance; discrepancies in this mapping point to causes of the queries' differences. Exploiting the queries gives explain3D an edge over traditional schema matching and record linkage techniques, which are query-agnostic. Our work makes the following contributions: (1) We formalize the problem of deriving optimal explanations for the differences of the results of semantically similar queries over disjoint datasets. Our optimization problem considers two types of explanations, provenance-based and value-based, defined over an evidence mapping, which makes our solution interpretable. (2) We design a 3-stage framework for solving the optimal explanation problem. (3) We develop a smart-partitioning optimizer that improves the efficiency of the framework by orders of magnitude. (4) We experiment with real-world and synthetic data to demonstrate that explain3D can derive precise explanations efficiently, and is superior to alternative methods based on integration techniques and single-dataset explanation frameworks.

PVLDB Reference Format:
Xiaolan Wang, Alexandra Meliou. Explain3D: Explaining Disagreements in Disjoint Datasets. *PVLDB*, 12(7): 779-792, 2019.
DOI: https://doi.org/10.14778/3317315.3317320

Proceedings of the VLDB Endowment, Vol. 12, No. 7
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3317315.3317320

1. INTRODUCTION

Data drives modern applications, analytic processes, and business decisions, heavily influencing many aspects of human activity: from product recommendations and friend connections, to autonomous vehicle decisions and election campaign strategies. Understanding data and the results of processes that operate on data becomes critical in promoting trust in data-driven decisions and in facilitating debugging and repair of errors [52]. Even within the relatively simple setting of relational data and queries, the explosive data sizes, source heterogeneity, and issues of poor data quality make providing explanations a challenging problem.

Existing data management solutions that aim to provide explanations for query results [44, 45, 56] have an important limitation: They focus on a single dataset, where data conforms to a single common schema. However, modern data rarely conforms to this integrated ideal. More often than not, datasets evolve separately, under different schemas, and even datasets from trustworthy sources frequently end up diverging, both in format and content, causing headaches to downstream applications and users. For example, Open Data [40], released by governments and organizations, is typically of high quality, publicly available, and freely used and distributed. Such datasets may be related and overlapping, but their separate production and evolution lead to disagreements that can cause confusion to users and incorrect analyses.

EXAMPLE 1 (ACADEMIC DATA DISAGREEMENT). *We collect two publicly-available academic datasets: the UMass-Amherst dataset on undergraduate programs*[1]*, and the National Center for Education Statistics (NCES) dataset*[2]*. Both data sources are reputable and contain high-quality information. Nevertheless, querying both datasets for the number of undergraduate degree programs at UMass Amherst yields vastly different answers.*

	UMass-Amherst data (D_{UMass})	*NCES data* (D_{NCES})
Schema:	*Major(Major, Degree, School)*	*School(<u>ID</u>, Univ_name, City, Url)* *Stats(<u>ID</u>, Program, bach_degr)*
Query:	Q_1 : `SELECT COUNT(Major) FROM Major;`	Q_2 : `SELECT SUM(bach_degr) FROM School, Stats WHERE Name = 'UMass-Amherst' AND School.ID=Stats.ID;`
Answer:	*113*	*90*

Existing explanation solutions can only be applied with respect to one of these datasets at a time, by asking questions such as "Why is the result of Q_1 (resp. Q_2) high (resp. low)?" But these would not provide meaningful explanations in this case, as each tuple contributes the same to the aggregate of Q_1, and prioritizing tuples with low `bach_degr` *in the provenance of Q_2 would be arbitrary, not grounded on the actual differences with Q_1.*

[1] `https://www.umass.edu/gateway/academics/undergraduate`
[2] `https://nces.ed.gov`: A open dataset presented in simplified schema.

SQL query Q_1:
```
SELECT COUNT(program)
FROM D1;
```
Dataset D_1:

Program	Degree
Accounting	B.S.
CS	B.A.
CS	B.S.
ECE	B.S.
EE	B.S.
Management	B.A.
Design	B.A.

(a) $Q_1(D_1) = 7$

SQL query Q_2:
```
SELECT COUNT(Major)
FROM D2 WHERE Univ='A';
```
Dataset D_2:

Univ	Major
A	Accounting
A	CSE
A	ECE
A	EE
A	Management
A	Design
B	Art

(b) $Q_2(D_2) = 6$

SQL query Q_3:
```
SELECT SUM(Num_bach)
FROM D3;
```
Dataset D_3:

College	Num_bach
Business	2
Engineering	2
Computer Science	1

(c) $Q_3(D_3) = 5$

SQL query Q_4:
```
SELECT SUM(Num_major)
FROM D4;
```
Dataset D_4:

Campus	Num_major
South campus	1
North campus	2
East campus	1

(d) $Q_4(D_4) = 4$

Figure 1: Four queries, operating on disjoint datasets, for answering the question: ***How many undergraduate degree programs are provided by University A?*** **However, all queries yield different answers:** $Q_1(D_1) = 7, Q_2(D_2) = 6, Q_3(D_3) = 5,$ **and** $Q_4(D_4) = 4$**.**

Example 1 illustrates the predicament of dealing with disagreements in disjoint datasets and how single-dataset explanation frameworks fall short. Attempting to use data cleaning [10, 36, 42] and data fusion [9, 21] techniques towards this problem meets similar challenges. These techniques attempt to reconcile the datasets, but are agnostic to the queries of interest, which may very well be contributors to the discrepancy. Ultimately, our goal is not to reconcile the differences between two datasets and consolidate them, but rather to explain the reasons of disagreement between two queries on those datasets, whose results are expected to be the same.

In this paper, we introduce explain3D,[3] a framework for deriving interpretable explanations for the disagreement in the results of two semantically similar queries.[4] Explain3D leverages the queries in coordination with existing schema matching and entity resolution methods to derive a semantic mapping across the relevant parts of the queries' provenance. It processes this initial mapping to find optimal provenance-based (mismatched tuples) and value-based (mismatched values) explanations and summarizes these explanations to increase understandability. For the disagreement in Example 1, explain3D finds that (1) several tuples in D_{UMass} (such as majors "Equine Management" and "Turfgrass Management") do not correspond to tuples in D_{NCES}, and (2) there is a mismatch of contributions for some tuples—for example, "Computer Science" is counted twice in Q_1 for the distinct B.S. and B.A. degrees, but "Computer Science" has `bach_degr=1` in D_{NCES}. Explain3D further analyzes the common properties of the derived explanations to summarize them as: (1) There is a large portion of mismatches for majors with `Degree`="Associate degree" in D_{UMass}; (2) There are majors with multiple degree types in D_{UMass}, counted multiple times by Q_1, for which `bach_degr=1` in D_{NCES}.

Explain3D addresses the following challenges:

Different schemas. Data sources often adopt different schemas and may thus store their data with different granularities. For example, in Example 1, D_{UMass} lists each degree program as an individual tuple whereas D_{NCES} stores an aggregate of the degrees in each program in the attribute `bach_degr`. Such differences significantly increase the difficulty in determining the mapping relationship between tuples in different datasets.

Missing data mapping. Data mapping or tuple mapping is essential in deriving the explanations. However, existing record linkage and entity resolution techniques [5, 37] typically target mapping entities within the same dataset or datasets with highly similar schemas. In contrast, in our setting, we can leverage the queries to provide us both with the relevant provenance, and clues of the matching attributes.

Distinct queries. Two queries meant to retrieve the same information across two datasets with different schemas are bound to be structurally different. Differences in the queries are confounded with differences in the data and schemas, obscuring the causes of discrepancies and making deriving explanations more challenging.

We make the following contributions.

- We introduce the necessary modeling abstractions and formalize the problem of deriving optimal explanations for the disagreements between the results of two semantically similar queries over two disjoint datasets. We identify explanations as one of two types: provenance-based (indicating mismatched tuples between the two datasets) and value-based (indicating incorrect values in particular tuples). These explanations are defined over an *evidence mapping*, which is an explanation of the explanations themselves, making our method interpretable. (Section 2)
- We introduce explain3D as a 3-stage framework for solving our optimal explanation problem. The first stage leverages the queries and standard schema matching and record linkage methods to derive an initial mapping between the relevant provenance data. The second stage, which is the core of our approach, models the optimization problem as a mixed integer linear program (MILP) and produces a refined *evidence mapping*. This mapping, informed by the queries and the datasets, pinpoints the discrepancies between the two datasets. The third stage relies on standard methods to analyze the common properties of the discrepancies and summarize the explanations. (Section 3)
- We propose a smart-partitioning optimizer that breaks the optimization problem of explain3D's second stage into smaller components, which can be solved separately, increasing the efficiency and scalability of our framework. (Section 4)
- We perform extensive experimental evaluation of explain3D using real-world and synthetic data, comparing it with a state-of-the-art single-dataset explanation framework, state-of-the art entity resolution approaches, and multiple baselines. Our evaluation shows that explain3D is superior in explanation accuracy compared to the alternatives, and the smart-partitioning optimizer is robust to multiple parameter settings and increases efficiency by orders of magnitude with little to no loss of accuracy. (Section 5)

2. EXPLANATIONS FOR DISJOINT DATA

In this section, we use a running example inspired by Example 1 to introduce our concepts and abstractions for modeling explanations for disagreements in disjoint datasets.

[3] Pronounced "explained"

[4] In the context of our work, semantic similarity is subjectively determined by human raters, and assumed as part of the input. This is analogous to the standards of semantic similarity in the natural language processing literature [19].

EXAMPLE 2. *Figure 1 displays four semantically similar queries that answer the question "How many undergraduate programs are provided by University A?" The queries compute the same thing semantically, but they operate on different datasets, with different schemas: D_1 lists the undergraduate programs at University A and Q_1 counts them; D_2 lists the majors at multiple universities and Q_2 selects the ones from University A and counts them; D_3 lists the number of bachelor degrees per college at University A and Q_3 sums them; D_4 lists the number of majors per campus at University A and Q_4 sums them. While all four queries are correct semantically, they ultimately yield different results.*

Manually, one can easily contrast Q_1 and Q_2. The Program and Major attributes are a direct match, and each program in D_1 corresponds to a major in D_2 and vice versa, through a one-to-one mapping: 'Accounting' to 'Accounting', 'CS' to 'CSE', 'ECE' to 'ECE', etc. This reveals that computer science is counted twice in Q_1, for the B.S. and B.A. degrees, but only once in Q_2, which explains the difference in their results. Moreover, the mapping of tuples between the two datasets is an interpretable explanation (evidence) of the explanation itself.

The correspondence between Q_1 and Q_3 is a little less straightforward, because the data is stored at different granularities (list of programs vs aggregates per college). However, the queries are still comparable. The Program attribute semantically maps to the College attribute in a containment relationship: each program typically corresponds to a college—Accounting and Management are part of the Business School, ECE and EE are part of the College of Engineering, and CS is part of the College of Computer Science. This mapping reveals that (1) CS is counted twice in Q_1, for the B.S. and B.A. degrees, but D_3 only lists one bachelor degree in the CS College, and (2) the Design program is missing from D_3.

While we can reason about the differences of Q_1, Q_2, and Q_3, we cannot compare them with Q_4 because the Campus attribute does not meaningfully correspond in a direct or containment relationship with the other datasets.

This example highlights several concepts: (1) attribute matches and their implications to (2) comparability of queries, (3) explanations as mismatched tuples or mismatched values, and (4) evidence mappings that support the derived explanations. We proceed to formalize these concepts and define the problem of deriving explanations for disagreements in the results of semantically similar queries over disjoint datasets.

2.1 Problem input: Queries, data, and matches

In this paper, we focus on queries of the general relational algebra form $Q = \pi_o\sigma_C(X)$, where X can be a single relation or an arbitrary query, allowing joins, unions, and subqueries; C also allows any operators, except UDFs. We restrict the projection, o, to be either a set of attributes, $o = \mathcal{A} \subseteq attr(X)$, or one of the five main SQL aggregate functions (SUM, COUNT, AVERAGE, MAX, MIN), $o = aggr(A_i), A_i \in attr(X)$. Compared to prior work on explanations over a single database [13, 45, 49, 56], which mostly focus on flattened queries in select-project-join (SPJ) and select-project-join-aggregate (SPJA) format, our framework supports a wider range of queries.

As Example 2 showed, some queries are not comparable (Q_1 and Q_4). Reasoning about these cases would require external information, not derivable by standard matching and linking methods. We cannot derive explanations for these cases—this appears impossible without external information—and we focus on comparable queries. As Example 2 highlighted, comparability is determined by semantic mappings that match attributes of the queries. We formalize these *attribute matches* below.

Notation	Description
$Q = \Pi_o\sigma_c(R)$	A query over relation R in database D.
$\mathcal{M}_{attr}(Q_1, Q_2) = (\mathcal{A}_i\phi\mathcal{A}_j)$	Attribute matches.
$\mathcal{M}_{tuple} = \{(t_i, t_j, p), ...\}$	Tuple matches.
$P(A_1, ..., A_k, I)$ or P	The provenance relation of query Q.
T	Canonical tuples of query Q.
$t.I$	The impact of a tuple t.
$E = (\Delta, \delta \vert \mathcal{M}^*_{tuple})$	Explanations and their evidence.
$\Delta = \{t, ...\} \in E$	Provenance-based explanations.
$\delta = \{t.I \mapsto t.I^*\} \in E$	Value-based explanations.
$\mathcal{M}^*_{tuple} \subseteq \mathcal{M}_{tuple}$	Evidence of a set of explanations.

Figure 2: Summary of notations.

DEFINITION 2.1 (ATTRIBUTE MATCHES). *Given two queries Q_1 over relation R_1 and Q_2 over relation R_2, we represent the semantic mapping among their attributes as **attribute matches**, denoted with the matching function $\mathcal{M}_{attr}$:*

$$\mathcal{M}_{attr}(Q_1, Q_2) = (\mathcal{A}_i\phi\mathcal{A}_j)$$

where $\mathcal{A}_i$, $\mathcal{A}_j$ are sets of categorical attributes in R_1 and R_2, respectively, and $\phi \in \{\equiv, \sqsubseteq, \sqsupseteq\}$ is the semantic relation between two sets of attributes [26].

In our definition of matching attributes, we borrow the notion of the semantic relation ϕ from prior work [26]. A set of attributes A_i can be *semantically equivalent* to A_j ($\mathcal{A}_i \equiv \mathcal{A}_j$), corresponding to a one-to-one mapping between instantiations of A_i and A_j, *less general* than A_j ($\mathcal{A}_i \sqsubseteq \mathcal{A}_j$), corresponding to a many-to-one mapping, or *more general* than A_j ($\mathcal{A}_i \sqsupseteq \mathcal{A}_j$), corresponding to one-to-many mapping. Note that semantic equivalence does not imply a condition on cardinality and two semantically equivalent sets of attributes may in fact have arbitrary overlap. For example, the sets $A_i = (address, city, state, zip)$ in R_1 and $A_j = (address)$ in R_2 can be semantically equivalent ($A_i \equiv A_j$). In our running example, $\mathcal{M}_{attr}(Q_1, Q_2) =$ (program) $\equiv$ (major), and $\mathcal{M}_{attr}(Q_1, Q_3) =$ (program) $\sqsubseteq$ (college). The attribute matches can be derived from standard schema matching techniques [2, 6, 26, 57]. Deriving these matches is not a focus in our work, and we treat them as part of our input.

One can consolidate or separate matches over sets of attributes, e.g., (zip, city) $\sqsubseteq$ (county) becomes (zip) $\sqsubseteq$ (county) and (city) $\sqsubseteq$ (county). Our framework applies to both cases. From here on, for ease of exposition, we will assume that the attribute matches are on a single attribute from each relation, and we will simply denote them with $\mathcal{M}_{tuple}$ when the queries are clear from the context.

If there exists at least one attribute match between two queries, we can derive explanations for their differences (comparable queries); otherwise, the queries are not comparable (Q_1 and Q_4).

DEFINITION 2.2 (COMPARABLE QUERIES). *Two queries Q_1 over relation R_1 and Q_2 over relation R_2 are comparable if and only if $\mathcal{M}_{attr}(Q_1, Q_2) \neq \emptyset$.*

We focus on comparable queries in this work, and from here on we will assume that the queries we discuss are comparable. To derive explanations for query disagreements, we need to analyze the contents of the two datasets and reason about their correspondence. We do not need to do so for the entire datasets, but rather for the parts that contribute to the queries (provenance). For example, in Q_2 only the tuples in D_3 with Univ='A' are part of the provenance. To facilitate exposition, we derive a *provenance relation*.

DEFINITION 2.3 (PROVENANCE RELATION). *Given a query $Q = \pi_o\sigma_c(R)$ over relation $R(A_1, \ldots)$, we derive a **provenance***

relation *$P(A_1, \dots, I)$ as follows: For each tuple $t \in \sigma_c(R)$, we create a tuple $t' = (t, I)$ in P, where $t'.I = \Pi_{o'}(t)$, with $o' = 1$ if Q is a non-aggregate query, and $o' = o$ otherwise. The impact of a tuple measures its statistical contribution to the result of query Q.*

In our running example, the provenance relation of Q_1 has 7 tuples (same as D_1), each with impact 1; the provenance relation of Q_3 has 3 tuples (same as D_3), with impacts 2, 2, and 1, same as the corresponding values of the Num_bach attribute.

Given two queries, the tuples of their provenance relations can be associated through mappings such as the ones described in Example 2. We formalize the tuple mapping below.

DEFINITION 2.4 (TUPLE MAPPING). *Given relations R_1 and R_2, the* ***tuple mapping*** *between R_1 and R_2 is a set of tuple matches:*

$$\mathcal{M}_{tuple} = \{(t_i, t_j, p), ...\}$$

where $t_i \in R_1$ and $t_j \in R_2$ are two tuples, and $p \in (0, 1]$ is the probability that tuple t_i and tuple t_j correspond to the same or associated (with respect to containment) entities.

In Example 2, a possible tuple mapping between Q_1 and Q_2 can be (omitting superfluous attributes for simplicity) $\mathcal{M}_{tuple} =$ {(Accounting, Accounting, 1.0), (CS, CSE, 0.9), (ECE, ECE, 1.0), (EE, EE, 1.0), (Management, Management, 1.0), (Design, Design, 1.0)}. Deriving such matches can be done with traditional record linkage techniques [5, 8, 16, 20, 54]. We use such techniques as blackbox components in our framework to derive an initial tuple mapping. This initial mapping is typically crude, with many possible tuple matches of varied probabilities, and it needs to be refined into the correct mapping $\mathcal{M}^*_{tuple}$. This refinement is a core part of our framework, which we will discuss in Section 3.

2.2 Problem output: The explanations

Example 2 highlighted the two generic types of explanations we derive: (1) provenance-based explanations, indicating mismatched tuples between the two datasets, and (2) value-based explanations, indicating incorrect values or contributions for particular tuples. We formalize these explanations below.

DEFINITION 2.5 (EXPLANATIONS). *Given two queries Q_1 and Q_2 and their provenance relations P_1 and P_2, the explanations of their differences include two generic types:*

- ***A provenance-based explanation*** *is a tuple $t \in P_1$ (resp. $t \in P_2$) such that t does not map to a $t' \in P_2$ (resp. $t' \in P_1$). We use Δ to denote a set of provenance-based explanations.*
- ***A value-based explanation*** *specifies an impact value change, $t.I \mapsto t.I^*$, for a tuple $t \in P_1 \cup P_2$, meaning that t should have impact $t.I^*$ rather than $t.I$. We use δ to denote a set of value-based explanations.*

Example 2 highlights a provenance-based explanation for the disagreement of Q_1 and Q_3 (the Design program is missing from D_3), and a value-based explanation (D_3 only lists one bachelor degree in the CS College, when it should be two). The derived explanations are tightly coupled with the tuple mapping. In comparing Q_2 and Q_3, a mapping that matches CSE with the Computer Science College, will produce different explanations than a mapping that matches CSE to the College of Engineering. Typically, the initial mappings ($\mathcal{M}_{tuple}$) derived from standard entity resolution and linkage techniques are probabilistic, and would assign the two possible matches for CSE with two distinct probabilities. Our goal is to discover the right mapping $\mathcal{M}^*_{tuple}$, leading to the correct (optimal) set of explanations; we call this refined mapping the *evidence mapping* (or evidence for short). The evidence mapping is a subset of the initial mapping ($\mathcal{M}^*_{tuple} \subseteq \mathcal{M}_{tuple}$), and needs to conform to certain properties discussed in Section 3.

The final product of our framework is a set of explanations and their evidence, reported as $E = (\Delta, \delta | \mathcal{M}^*_{tuple})$. The evidence $\mathcal{M}^*_{tuple}$ is an explanation of the explanations themselves, making our result fully interpretable.

2.3 Optimal explanations for 3D

We now define the problem of deriving optimal explanations for disagreements in disjoint data, which we will refer to as EXP-3D.

PROBLEM 1 (THE EXP-3D PROBLEM). *Given two queries Q_1 and Q_2 with provenance relations P_1 and P_2, respectively, and a set of initial tuple matches $\mathcal{M}_{tuple}$, our goal is derive a set of explanations, $E = (\Delta, \delta | \mathcal{M}^*_{tuple})$ that* ***maximize the probability****:*

$$Pr(E | P_1, P_2, \mathcal{M}_{tuple})$$

More informally, we are looking for the set of explanations and their evidence mapping that are the most likely, given the queries' provenance and the initial probabilistic tuple mapping. In our running example, suppose that the initial mapping for Q_2 and Q_3 assigns two possible matches for CSE, Computer Science and Engineering, each with some probability. This indicates two possible cases for $\mathcal{M}^*_{tuple}$, mapping CSE to Computer Science in one case and Engineering in the other. The former choice results in a single provenance-based explanation (the tuple with major='Design' in D_2 does not have a match in D_3). The latter choice, results in the same explanation and, in addition, that the tuple with College='Computer Science' in D_3 does not have a match in D_2, and that the Num_bach value of the Engineering tuple in D_3 is wrong. Clearly, the former choice is better. Intuitively, a particular tuple mapping identifies specific discrepancies, which we map to explanations, and fewer discrepancies are typically preferred.

In Section 3.1, we analyze the calculation of the objective function of Problem 1, and reduce it to a simpler scoring function that is both tractable and theoretically-grounded. In Section 3.2, we describe a framework for deriving the explanations and evidence mapping through a translation to Mixed Integer Linear Programs (MILP). Then, in Section 3, we describe a smart-partitioning optimizer that improves the efficiency of our basic approach by several orders of magnitude.

3. DERIVING EXPLANATIONS

In this section, we present explain3D, a 3-stage framework that solves Problem 1. The first stage (Section 3.1) refines the provenance data into a canonical form that is easier to analyze. With data in this canonical form, we define essential properties for evidence mappings and explanations, and use them to simplify the objective function of Problem 1. The second stage (Section 3.2), which is the core of our approach, models the optimization problem as a mixed integer linear program (MILP) and produces a refined *evidence mapping* and the corresponding explanations. The third stage (Section 3.3) relies on standard methods to analyze the common properties of the discrepancies and summarize the explanations.

3.1 Stage 1: Canonicalization and Simplification

The provenance relation P_1 of Q_1 has two tuples for the CS program, one for the B.S. and one for the B.A. degree. The degree information is not relevant to the comparison with Q_2, and it is not part of the mapping between Q_1 and Q_2. Thus the two CS tuples in P_1 are indistinguishable with respect their role in the disagreement between Q_1 and Q_2. This indicates that the provenance relation contains redundancy. We consolidate redundant tuples and their impact through *canonicalization*. Canonicalization groups tuples with the same values for the matching attributes and sums their impacts. Canonicalization does not change the provenance relations of queries that require a strict one-to-one mapping (queries with AVG/MAX/MIN aggregation). The canonical relation of Q_1 has 6 tuples (instead of 7 in P_1), and CS is represented by a single tuple with impact 2 (Figure 3a).

DEFINITION 3.1 (CANONICAL RELATION). *Given a provenance relation P of a query Q, and attribute matches $\mathcal{M}_{attr}$, the canonical relation T of P is derived with the query:*

$$T = \pi_{\mathcal{A},I}({}_{\mathcal{A}}\mathcal{G}_{SUM(I)}(P))$$

Where $\mathcal{A}$ is a set of matching attributes that appear in $\mathcal{M}_{attr}$; ${}_{\mathcal{A}}\mathcal{G}_{SUM(I)}$ is the Group By operation over attributes $\mathcal{A}$ with aggregate function SUM on the impact attribute I.

EXAMPLE 3. *Figure 3 shows the canonical relations of Q_1 and Q_2 based on the attribute matches $\mathcal{M}_{attr} = (program \equiv major)$. The canonical relation of Q_1 is constructed with the query:*
`SELECT` *program, COUNT(I)* `AS` *I* `FROM` P_1 `GROUP BY` *program*

Canonicalization simplifies the datasets without losing information necessary for the reasoning on disagreements. It further allows us to identify and formalize essential properties for explanations and evidence mapping, which we analyze next.

Explanation properties

Completeness. Explanations define refinements on the canonical relations. A provenance-based explanation indicates the removal of tuples, and a value-based explanation modifies a tuple's impact. Our goal is to identify a set of explanations that is *complete*: if one performs all the refinements defined by the explanations, the queries would return the same result. We evaluate completeness through the properties of valid mapping and equal impact. In the following, we denote $T_1^* = \delta(T_1 \setminus \Delta)$ and $T_2^* = \delta(T_2 \setminus \Delta)$ as the refined tuples of the canonical relations.

Mapping validity. The attribute matches ($\mathcal{M}_{attr}$) between two queries imply the cardinality of the tuple matches between the two canonical relations. If two attributes have an equivalence match, e.g., program $\equiv$ major, then the canonical relations should have a one-to-one mapping of their tuples. Thus, in Figure 3, each tuple in T_1 should map to one tuple in T_2. If it is a less general match, e.g., program $\sqsubseteq$ college, then the mapping should be many-to-one (many programs map to one college). We can never have many-to-many mappings.

Initial tuple mapping, however, typically do not conform to the required cardinality, as they frequently assign several probabilistic matches for each tuple. For example, the CSE major in Q_2 may be mapped to two colleges in Q_3, Engineering and Computer Science, which violates the many-to-one cardinality requirement for two relations. Our goal is to produce a refined mapping $\mathcal{M}_{tuple}^*$ that conforms to the cardinality requirements of the attribute matches $\mathcal{M}_{tuple}^*$; we call such a mapping *valid*.

DEFINITION 3.2 (VALID MAPPING). *Given attribute matches $\mathcal{M}_{attr} = (\mathcal{A}_i \phi \mathcal{A}_j)$, and two sets of refined tuples, T_1^* and T_2^*, the mapping $\mathcal{M}_{tuple}^*$ is* ***valid*** *if and only if the following are true:*

- *If $\mathcal{A}_i \sqsubseteq \mathcal{A}_j$, then $\forall t \in T_1^*, |\{t|(t,t',p) \in \mathcal{M}_{tuple}^*\}| \leq 1$*
- *If $\mathcal{A}_i \sqsupseteq \mathcal{A}_j$, then $\forall t \in T_2^*, |\{t|(t',t,p) \in \mathcal{M}_{tuple}^*\}| \leq 1$*
- *If $\mathcal{A}_i \equiv \mathcal{A}_j$, then both the above conditions hold.*

rowID	Program	I
p_1	Accounting	1
p_2	CS	2
p_3	ECE	1
p_4	EE	1
p_5	Management	1
p_6	Design	1

(a) T_1: Canonical relation for Q_1

rowID	Major	I
m_1	Accounting	1
m_2	CSE	1
m_3	ECE	1
m_4	EE	1
m_5	Management	1
m_6	Design	1

(b) T_2: Canonical relation for Q_2

Figure 3: Canonical relations for queries Q_1 and Q_2 of Figure 1. I **denotes the impact of the tuples.**

Impact equality. Tuples of the canonical relations and their mapping form a bipartite graph. In a valid mapping, where the matches can only be one-to-one, one-to-many, or many-to-one, the graph separates into connected components. Each component contains the tuples that correspond to each other semantically. When the two query results agree, the total impact on each side of the bipartite graph is the same within each connected component. Thus, our goal is to find a set of explanations, such that the refined canonical relations T_1^* and T_2^* demonstrate such impact equality.

DEFINITION 3.3 (IMPACT EQUALITY). *Given canonical relations T_1^* and T_2^*, and a bipartite graph G formed by a valid mapping $\mathcal{M}_{tuple}^*$ between T_1^* and T_2^*, the impact equality property is satisfied if and only if for all connected components (T_1', T_2') of G:*

$$\sum_{t \in T_1'} (t.I) = \sum_{t \in T_2'} (t.I)$$

DEFINITION 3.4 (COMPLETE EXPLANATIONS). *A set of explanations $E = (\Delta, \delta | \mathcal{M}_{tuple}^*)$ over canonical relations T_1 and T_2 is* ***complete*** *if $\mathcal{M}_{tuple}^*$ is a valid mapping and $T_1^* = \delta(T_1 \setminus \Delta)$ and $T_2^* = \delta(T_2 \setminus \Delta)$ satisfy the impact equality property.*

Explanation problem revisited

The objective function of Problem 1 maximizes the probability $Pr(E|P_1, P_2, \mathcal{M}_{tuple})$. This probability can be equally and more efficiently computed over the canonical relations, which are a (lossless, for the purposes of this problem) summary of the provenance relations: $Pr(E|P_1, P_2, \mathcal{M}_{tuple}) = Pr(E|T_1, T_2, \mathcal{M}_{tuple})$.

From Bayesian inference, this is proportional to the product of three probabilities:

$$\begin{aligned} &Pr(E|T_1, T_2, \mathcal{M}_{tuple}) \\ &\qquad \propto Pr(T_1, T_2|E) Pr(\mathcal{M}_{tuple}|T_1, T_2, E) Pr(E) \end{aligned} \tag{1}$$

We next consider each of the three probabilities separately.

$\mathbf{Pr(T_1, T_2|E)}$. Assuming that tuples are independent, we have:

$$Pr(T_1, T_2|E) = \prod_{t \in T_1 \cup T_2} Pr(t|E) \tag{2}$$

We use α and β to denote the a priori probabilities that $t \in T_1 \cap T_2$ and that t has correct impact $t.I$, respectively. Intuitively, $\alpha, \beta \in (0.5, 1]$, as a tuple is more likely to be covered by both queries and have correct impact than not.[5] We then compute the probabilities of the different cases of t's inclusion in a set of explanations E as:

$$\begin{aligned} Pr(t|t \notin \Delta, t \notin \delta) &= \alpha\beta; \quad Pr(t|t \notin \Delta, t \in \delta) = \alpha(1-\beta); \\ Pr(t|t \in \Delta, t \notin \delta) &= 1 - \alpha; Pr(t|t \in \Delta, t \in \delta) = 0. \end{aligned} \tag{3}$$

$Pr(T_1, T_2|E)$ is then derived from Equations (2)-(3). Larger Δ and δ lead to lower probabilities, thus the computation prioritizes smaller provenance- and value-based explanations.

[5]For simplicity, we assume the same α and β for all tuples, but our framework can handle different values across tuples.

$\mathbf{Pr(\mathcal{M}_{tuple}|T_1, T_2, E)}$. Assuming independence in tuple matches:

$$Pr(\mathcal{M}_{tuple}|T_1, T_2, E) = \prod_{m \in \mathcal{M}_{tuple}} Pr(m|T_1, T_2, E) \quad (4)$$

In addition, for a tuple match $m = (t_i, t_j, p)$, the probability that tuples t_i and t_j match is p, thus:

$$\begin{aligned}
&Pr(m|m \in \mathcal{M}_{tuple}*, t_i, t_j \in T_1 \cup T_2) = p;\\
&Pr(m|m \notin \mathcal{M}_{tuple}*, t_i, t_j \in T_1 \cup T_2) = 1 - p;\\
&Pr(m|t_i, t_j \notin T_1 \cup T_2) = 0. \quad (5)
\end{aligned}$$

$Pr(\mathcal{M}_{tuple}|T_1, T_2, E)$ is then derived from Equations (4)-(5).

The probability computation prioritizes tuple matches with higher probabilities in the evidence mapping.

$\mathbf{Pr(E)}$. In this paper, we simply set the prior probability of a set of explanations E, based on whether it is complete (Definition 3.4). If E is complete, then $Pr(E) = 1$; otherwise, $Pr(E) = 0$. These priors force our framework to only consider explanations that resolve all disagreements.

We can then compute the objective function from Equation (1). In practice, to improve efficiency we calculate and later optimize the probability in the logarithmic space:

$$\begin{aligned}
&\log(Pr(E|T_1, T_2, \mathcal{M}_{tuple})) \propto\\
&\qquad \log(Pr(T_1, T_2|E)) + \log(Pr(\mathcal{M}_{tuple}|T_1, T_2, E)). \quad (6)
\end{aligned}$$

Through a reduction from the Exact Cover problem[6], we can prove that Problem 1 is NP-complete [53].

THEOREM 3.5. EXP-3D *(Problem 1) is NP-complete.*

3.2 Stage 2: MILP transformation

In this section, we show how stage 2 of explain3D transforms the EXP-3D problem into a mixed integer linear program (MILP). This transformation allows explain3D to use modern constrained optimization solvers to derive the optimal explanations. Later, in Section 4, we show how to optimize computation in this stage, through a smart-partitioning optimizer.

To translate an instance of the EXP-3D problem into a MILP problem, we first convert tuples, their tuple matches, and the associated explanations into linear constraints; we then express the explanation completeness properties, using linear constraints; we complete the translation process by formalizing a linear expression for the probability of the explanations.

Expressing explanations

To express the explanations, we first introduce a binary variable for each tuple $t_i \in T_1 \cup T_2$ and a binary variable for each tuple match (t_i, t_j, p); we then translate the changes suggested by the explanations into linear constraints.

Tuple: Given a tuple $t_i = (t_i.A_1, ..., I)$, there are two types of explanations that may be associated with this tuple: (1) a provenance-based explanation ($t_i \in \Delta$); (2) a value-based explanation ($t_i \in \delta$). We use a binary variable x_i to indicate whether tuple t_i is included in an provenance-based explanation; To express the value-based explanation, we use a integer variable $t.I^*$ for tuple t's refined impact and a binary variable y_i representing whether the tuple's refined impact is the same as its original impact ($y_i = 1$) or not ($y_i = 0$). The binary variable y_i should satisfy the following constraint.

$$y_i \quad = \quad (t.I^* = t.I) \quad (7)$$

When $x_i = 1$, the tuple $t_i \in \Delta$ is selected as a provenance-based explanation; when $x_i = 0$, the tuple t_i remains in the canonical relation and its impact is set to $t.I^*$.

Based on the binary variables and Equation (3), we express the probability of the explanations being associated with tuple t_i as:

$$\log(Pr(t_i)) = x_i \otimes a + \underline{(1 - x_i) \otimes ((1 - y_i) \otimes b + y_i \otimes c)}$$

In the above expression, $\otimes$ represents regular multiplication; we prefer to use $\otimes$ to indicate that it is the semi-module multiplication by scalars; $a = \log(1-\alpha)$, $b = \log(\alpha)+\log(\beta)$, and $c = \log(\alpha)+\log(1-\beta)$ as three constant values. Note that the above Equation is quadratic due to the underlined expression: $P_i = (1 - x_i) \otimes ((1 - y_i) \otimes b + y_i \otimes c)$. We linearize P_i, with the help of two constant numbers L and U as follows [3].

$$\begin{aligned}
&P_i \geq L \otimes (1 - x_i)\\
&P_i \leq U \otimes (1 - x_i)\\
&P_i \geq (1 - y_i) \otimes b + y_i \otimes c - U \otimes x_i\\
&P_i \leq (1 - y_i) \otimes b + y_i \otimes c - L \otimes x_i\\
&\log(Pr(t_i)) = x_i \otimes p_1 + P_i \quad (8)
\end{aligned}$$

The constant number L (or H) cannot be greater than the lower bound (or smaller than the upper bound) of P_i.

Tuple Match: Given a tuple match $m = (t_i, t_j, p)$, we use a binary variable $z_{i,j}$ to express whether it is a true match: When $z_{i,j} = 1$, we include it in the evidence mapping. The probability of this match is computed as follows.

$$\begin{aligned}
&z_{i,j} \leq (1 - x_i); \qquad z_{i,j} \leq (1 - x_j)\\
&\log(Pr(m)) = z_{i,j} \otimes \log(p) + (1 - z_{i,j}) \otimes log(1 - p) \quad (9)
\end{aligned}$$

Where x_i and x_j are the binary variables associated with t_i and t_j, respectively.

Expressing explanation completeness

We use the explanation variables to express the mapping validity and impact equality properties as linear constraints.

Valid Mapping: As required by Definition 3.2, the refined tuple matches $\mathcal{M}^*_{tuple}$ should follow the valid mapping property, which essentially restricts the degree for some of the tuples to be less than or equal to 1. If t_i is such a tuple, then we add the constraints:

$$\sum_{(t_i, t_j, p) \in M} z_{i,j} \leq 1 \quad (10)$$

Equal Impact: Valid mappings between the canonical tuples T_1 and T_2 can never have many-to-many cardinality. Therefore, in the bipartite graph between T_1 and T_2 under a valid mapping, at least one of T_1 or T_2 is guaranteed to have only tuples with maximum degree of 1. This observation allows us to simplify the specification of the connected components in the bipartite graph and the corresponding impact calculations. Suppose that all tuples in T_1 have maximum degree 1. Then the set of connected components is:

$$\mathcal{S} = \{(\eta(t_j), t_j, M)|t_j \in T_2\}$$

where $\eta(t_j)$ is the set of T_1 tuples that are adjacent to $t_j \in T_2$. Consider one connected component $(\eta(t_j), t_j, M) \in \mathcal{S}$, the total impact of T_1 in the component is $I_l = \sum_{t_i \in \eta(t_j)} z_{i,j} \otimes t_i.I^*$; and

[6]The Exact Cover problem is one of Karp's 21 NP-complete problems [30].

Algorithm 1: The basic solution

Input : Two sets of canonical tuples (T_1, T_2) and acquired tuple matches $(\mathcal{M}_{tuple})$
Output: A set of explanations
1 $milp\_vars, milp\_cond, prob\_expr \leftarrow \emptyset$;
2 **foreach** *tuple* t *in* $T_1 \cup T_2$ **do**
3     $milp\_vars \leftarrow milp\_vars \cup$ DefineTupleVariables(t);
4     $milp\_cond \leftarrow milp\_cond \cup$ TupleImpactCondition(t);
5     $prob\_expr \leftarrow prob\_expr \cup$ TupleProbability(t);
6 **foreach** *mapping* m *in* $\mathcal{M}$ **do**
7     $milp\_vars \leftarrow milp\_vars \cup$ DefineMappingVariables(m);
8     $prob\_expr \leftarrow prob\_expr \cup$ MappingProbability(m);
9 $milp\_cond \leftarrow milp\_cond \cup$ FormConditions($milp\_vars$);
10 $milp \leftarrow$ FormMILP($milp\_cond, prob\_expr$);
11 $solved\_vars \leftarrow$ SolveMILP($milp$);
12 $E \leftarrow$ DecodeVariables($solved\_vars$);
13 **return** E;

the total impact of T_2 tuples is $I_r = t_j.I^*$. Here, we linearize the quadratic equation I_l using the same method as Equation (8).

$$
\begin{aligned}
I_i &\leq U \otimes z_{i,j} \\
I_i &\geq L \otimes z_{i,j} \\
I_i &\leq t_i.I^* - L \otimes (1 - z_{i,j}) \\
I_i &\geq t_i.I^* - U \otimes (1 - z_{i,j})
\end{aligned}
\tag{11}
$$

Where $I_i = z_{i,j} \otimes t_i.I^*$ is an element in I_l; L and U are two constants that cannot be greater than the lower bound (or smaller than the upper bound) of a tuple's impact.

Finally, the equal impact property requires:

$$\sum_{t_i \in \eta(t_j)} I_i = I_l \tag{12}$$

Formalizing the objective function

The EXP-3D problem aims to derive a set of complete explanations such that the probability of the explanations is maximized. The MILP formulation creates variables for all provenance-based (tuples) and all value-based (impact) explanations. Our objective function can be formulated as a linear expression over the explanation variables in a fashion similar to the constraints of the explanation properties:

$$\log(Pr(E|\mathcal{T}, \mathcal{M})) = \sum_{t \in \mathcal{T}} \log(Pr(t)) + \sum_{m \in \mathcal{M}} \log(Pr(m)) \tag{13}$$

Where $\mathcal{T} = T_1 \cup T_2, \mathcal{M} = \mathcal{M}_{tuple}$; $\log(Pr(t))$ and $\log(Pr(m))$ are formulated by Equation (8) and Equation (9) respectively.

The algorithm

Algorithm 1 provides the pseudocode implementing the MILP transformation described in this section. The algorithm first iterates over all tuples in the input to define variables, construct constraints, and express the tuple probabilities in Lines 3-5. The algorithm then iterates over all the tuple matches and formalizes the probability expression in Line 8 according to Equation (9). Next, the algorithm constructs constraints for the completeness requirement, as in Equations (10)-(12), by a *FormConditions* function (Line 9). With the variables and constraints, the algorithm completes the MILP problem formulation and calls a MILP solver to get a solution (Line 10-11). We derive the final explanations from the MILP solution by including an explanation or evidence (tuple match) if the solve value of the corresponding binary variable is 1 (Line 12).

3.3 Stage 3: Summarization

The product of stage 2 of explain3D is a set of explanations and their evidence mapping. But when the discrepancies between two datasets are extensive, the derived explanations could involve a large number of tuples and values. Reviewing such explanations can be tedious. Stage 3 of our framework is tasked with summarizing and abstracting the explanations to reduce their size and increase their understandability. As in Example 1, we may use `Degree=`*"Associate degree" in D_{UMass}* to summarize the common patterns of the derived explanations, which is easier to understand than presenting the explanations individually.

Different summarization methods are possible. Explain3D marks tuples associated with explanations as a "target" and then uses existing techniques, such as Data Auditor [27] and Data X-Ray [50] to identify common patterns for the target tuples. Alternatively, "target" tuples could be treated as examples by QBE (Query-By-Example) techniques [17,23,43,46], which can then generate SQL queries that precisely describe them. Developing novel summarization methods is not a focus of our work in this paper, and thus stage 3 relies on existing tools. Detailed stage 2 explanations are still available through explain3D, for users who prefer to peruse the more precise and detailed causes of the disagreement.

4. PARTITIONING OPTIMIZATION

A critical problem with stage 2 of the explain3D framework is that it does not scale for problems with a large number of tuples and tuple matches. The problem is that the generated MILP grows to sizes that can stump even state-of-art solvers. To improve the efficiency of the basic algorithm, we can split the bipartite graph $G = (T_1, T_2, \mathcal{M}_{tuple})$ into its maximal connected components and solve the problem in each component separately. This method requires linear time, $O(|T_1| + |T_2| + \mathcal{M}_{tuple})$, to derive the connected components and it does not sacrifice the accuracy. However, it fails to achieve any efficiency or scalability guarantees, as in the worst case, G may be connected.

Inspired by the connected components approach, we propose a method to divide the original problem into a collection of sub-problems with bounded sizes such that each sub-problem is guaranteed to be small enough to solve. Our partitioning method is based on the Graph Partitioning Problem (GPP) [11, 31, 34, 41], which aims to minimize the total weight of the edge cuts[7].

PROBLEM 2 (THE GRAPH PARTITIONING PROBLEM). *Given a number $k \in \mathbb{N}_{>1}$, a bipartite graph $G = (T_1, T_2, \mathcal{M}_{tuple})$ formed by tuples and their matches, and an upper bound L_{max} for the maximum partition size, we seek a partition Π of $T_1 \cup T_2$ with disjoint collections of tuples $\Pi = \{(T_{1,1}, T_{2,1}), ..., (T_{1,k}, T_{2,k})\}$ such that:*

- $T_{1,1} \cup ... \cup T_{1,k} = T_1$ *and* $T_{2,1} \cup ... \cup T_{2,k} = T_2$;
- $|T_{1,i}| + |T_{2,j}| \leq L_{max}$;
- *EdgeCutSum*$(\Pi) = \sum_{(t_i,t_j) \in E} w(t_i, t_j)$ *is minimized.*

Where $E = \{(t_i, t_j), ...\}$ denotes the set of edges across partitions; $w(t_i, t_j)$ denotes the weight of edge (t_i, t_j); $|T_{1,i}| + |T_{2,j}| \leq L_{max}$ is the balancing constraint over the maximum size of one partition.

In our setting, a naïve way to assign the edge weights is by using the tuple matches' probabilities: $w(t_i, t_j) = p$. However, this setting is ill-suited for our problem: According to our objective function (Problem 1), cutting a high probability tuple match tends to hurt our objective value much more than cutting multiple lower

[7] Edge cuts refer to edges across partitions.

Algorithm 2: The pre-partitioning algorithm

Input : A bipartite graph $G = (T_1, T_2, \mathcal{M}_{tuple})$ and thresholds θ_l, θ_h, R
Output: A simplified graph $G_c = (C_1, C_2, \mathcal{M}_c)$

1 $C_1, C_2, \mathcal{M}_c \leftarrow \emptyset$;
2 **foreach** *tuple t in* $T_1 \cup T_2$ **do**
3   **if** *t.isVisited* **then**
4   continue
5     $(T'_1, T'_2) \leftarrow$ FindHighProbTuplesDFS(t, G, θ_h);
6     $(C'_1, C'_2) \leftarrow$ MergeTuples(T'_1, T'_2);
7     $(C_1, C_2) \leftarrow$ UpdateMergedTuples(C'_1, C'_2);
8 **foreach** *mapping* (t_i, t_j, p) *in* $\mathcal{M}_{tuple}$ **do**
9     $(C'_i, C'_j) \leftarrow$ FindMergedTuples(C_1, C_2, t_i, t_j);
10     $\mathcal{M}_c \leftarrow$ UpdateEdgeWeight(C'_i, C'_j, p, R)
11 **return** $G_c = (C_1, C_2, \mathcal{M}_c)$;

probability tuple matches with equal or even higher total probabilities. For example, let us assume that we cut a tuple match, with 0.9 probability, that is part of the optimal explanation ($\mathcal{M}^*_{tuple}$). The objective value, $Pr(E)$, would drop by 9 times[8] as the probability of this tuple match, $Pr(m|m \in \mathcal{M}^*_{tuple})$, would change from 0.9 to 0.1. This objective value loss is significantly higher than cutting two tuple matches with lower individual (0.6 each) but higher total probabilities (1.2 in total). The latter case would only lead to a objective value drop by 2.25 times. Based on this observation, we prioritize cutting tuple matches with lower probabilities and avoid cutting tuple matches with high probabilities. We achieve this by adjusting the edge weight assignments as below:

$$w(t_i, t_j) = \begin{cases} p \cdot R, & if\ p \geq \theta_h; \\ p/R, & if\ p \leq \theta_l; \\ p, & otherwise. \end{cases}$$

Where $R \in (1, \infty)$ is a constant for rewarding (or penalizing) high probability (or low probability) tuple matches and $0 \leq \theta_l < \theta_h \leq 1$ are two thresholds specifying low and high probability tuple matches. In this paper, we set $\theta_l = 0.1, \theta_h = 0.9, R = 100$.

Existing graph partitioners, e.g., METIS [41] and hMETIS [31], can be used directly to derive the sub-problems, but they are not efficient when R is large. To further optimize partitioning efficiency, we employ a *pre-partitioning step* that combines tuples connected by high probability tuple matches. This pre-partitioning step can also be considered as an extra coarsening level on top of the *multi-level graph partitioning algorithms* [32, 33]. Empirically, this step achieves $200\times$ partition time speedup over graphs with $10K$ tuples without compromising optimality.

Algorithm 2 presents the pseudocode of the pre-partitioning step. The algorithm iterates over tuples in the bipartite graph in arbitrary order and attempts to merge tuples that are connected by high probability tuple matches as much as possible (Lines 2-7). It then iterates over the remaining tuple matches and updates the edge weights of the merged tuples accordingly (Lines 8-10). This algorithm has linear time complexity: $O(|T_1| + |T_2| + |\mathcal{M}_{tuple}|)$.

Finally, Algorithm 3 presents our smart-partitioning method. This algorithm first leverages the pre-partitioning algorithm (Algorithm 2) to generate a much smaller graph (Line 1); it then partitions the smaller graph (Line 2) with a standard graph partitioner; it finally produces the final partitioning Π according to the tuples' assigned partitions (Lines 3-6).

[8]This is based on the assumption that the probabilities of other tuples and tuples matches are not impacted.

Algorithm 3: The smart-partitioning algorithm

Input : A bipartite graph $G = (T_1, T_2, \mathcal{M}_{tuple})$, thresholds θ_l, θ_h, R, the number of partitions k, and the maximum partition size L_{max}
Output: A partition Π

1 $G_c \leftarrow$ PrePartition$(G, \theta_l, \theta_h, R)$;
2 $\Pi_c \leftarrow$ GraphPartitioner(G_c, k, L_{max});
3 $\Pi \leftarrow$ InitializeKEmptyPartitions(k) ;
4 **foreach** (C'_1, C'_2) *in* G_c **do**
5     $idx \leftarrow \Pi_c(C'_1, C'_2)$;
6     $\Pi[idx] \leftarrow$ AddTuples(C'_1, C'_2);
7 **return** Π;

5. EXPERIMENTAL EVALUATION

In this section, we evaluate the effectiveness and efficiency of explain3D using both real-world and synthetic data. In particular, we first compare explain3D with several alternative algorithms over two categories of real-world data (Section 5.2); then, we evaluate the performance and benefit of the smart-partitioning optimization over a series of synthetic datasets with diverse properties (Section 5.3).

5.1 Experimental setup

All experiments were performed on 4×2.77 GHz machines with 32GB RAM running IBM CPLEX [29] as the MILP solver on MacOS version 10.11.6.

5.1.1 Datasets, queries, and gold standards

We first describe the real-world data used in our evaluation; we describe our synthetic data experiments in Section 5.3.

Academic datasets. We collect three publicly available academic datasets, the UMass-Amherst dataset on undergraduate programs and the National Center for Education Statistics (NCES) dataset, described in Example 1, and the the OSU dataset on undergraduate programs[9]. We create two pairs of datasets for comparisons: (1) *UMass-Amherst vs. NCES*, described in Example 1, and (2) *OSU vs. NCES*, described in the table below. We evaluate all alternative algorithms with queries that compute *the number of undergraduate programs at UMass Amherst (or OSU, respectively)* on each pair of data.

OSU data (D_{OSU})	***NCES data*** (D_{NCES})
Major(Major, Degree, Campus, School)	*School(<u>ID</u>, Univ_name, City, Url)* *Stats(<u>ID</u>, Program, bach_degr)*

Q_1 :
`SELECT COUNT(Major)`
`FROM Major;`
	Q_2 :
	`SELECT SUM(bach_degr)`
	`FROM School, Stats`
	`WHERE Name = ‘OSU’`
	`AND School.ID=Stats.ID;`

Gold Standard: We manually create the gold standard for the explanations and the evidence mapping on both pairs of data. The datasets, queries, and gold standards are publicly available[10]. Figure 4 shows the detailed statistics of the academic datasets.

IMDb Datasets. We retrieve the IMDb data[11], and use it to create a pair of disjoint datasets, as two views with different schemas over the original data. To simulate the real-world disagreements over disjoint data, we choose a schema design for the first view such that a certain portion of data is lost during the data migration process.[12] We further introduce ∼5% random errors to both

[9]`http://undergrad.osu.edu/majors-and-academics/majors`
[10]`https://bitbucket.org/xlwang/explain3d`
[11]`https://datasets.imdbws.com/`
[12]In D_{IMDb1}, a movie is associated with a single country and genre.

views with the BART system [1]. We create 10 query templates (listed below), mapped over each view, covering a wide range of query types, including joins, subqueries, non-aggregates, and 5 different aggregate functions. We create 10 instantiations of each template, by selecting a random value for *year*$\in [1970, 2003]$ for templates Q_1–Q_9, and a random value for *genre* in Q_{10}, resulting in a total of 100 different queries.

IMDb View 1 (D_{IMDb1})
Movie (<u>movie_id</u>, title, release_year, genre, country, runtimes, gross, budget)
Actor (<u>actor_id</u>, firstname, lastname, gender, dob)
Director (<u>director_id</u>, firstname, lastname, gender, dob)
MovieDirector (movie_id, director_id) MovieActor (movie_id, actor_id)
IMDb View 2 (D_{IMDb2})
Movie (<u>m_id</u>, title, release_year) MovieInfo (m_id, info_type, info)
Person (p_id, name, gender, dob) MoviePerson (m_id, p_id)
Query templates
Q_1 *Return actors who were cast in short movies released in ⟨year⟩.*
Q_2 *Return movies directed by someone born in ⟨year⟩.*
Q_3 *Return the number of comedy movies released in ⟨year⟩.*
Q_4 *Return the number of movies released in the US in ⟨year⟩.*
Q_5 *Return the total gross value for movies released in ⟨year⟩.*
Q_6 *Return the maximum gross value for movies released in ⟨year⟩.*
Q_7 *Return the longest movie released in ⟨year⟩.*
Q_8 *Return the average gross value for movies released in ⟨year⟩.*
Q_9 *Return the average runtime for movies released in ⟨year⟩.*
Q_{10} *Return actresses who have not starred in any ⟨genre⟩ movies.*

Gold Standard: While creating the two disjoint views, we keep track of the data lost in the first view and record the random errors introduced by BART; these are the optimal explanations of the query disagreements. The optimal evidence mapping can also be easily acquired through the mapping between the views and the original dataset. The detailed statistics of the IMDb datasets are shown in Figure 4.

5.1.2 Attribute matches and tuple mapping

Attribute Matches. The attribute matches ($\mathcal{M}_{attr}$) for the two real-world datasets are shown in Figure 5.

Tuple Mapping. In evaluation, we use a similarity-to-probability method [24, 55] to collect the initial tuple mapping ($\mathcal{M}_{tuple}$). This similarity-to-probability method is a two-step process that generates the tuple matches probabilities from their similarity values: (1) it first divides the tuple matches into k continuous buckets over the similarity values; (2) in each bucket, it calculates the probability of tuple matches by the ratio of true matches within the current bucket. The true matches can be acquired by labeling a subset of data, or by a known gold standard.

To generate the similarity values, we use token-wise Jaccard similarity for *String attributes*:

$$sim(t_i.A, t_j.A) = \frac{|t_i.A \cap t_j.A|}{|t_i.A \cup t_j.A|}$$

We use normalized Euclidean distance on *numeric attributes*:

$$sim(t_i.A, t_j.A) = \frac{1}{1 + |t_i.A - t_j.A|^2}$$

We finally combine the similarity values over multiple attributes by taking their mean value:

$$sim(t_i, t_j) = \frac{\sum_{A \in \mathcal{M}_{attr}} sim(t_i.A, t_j.A)}{|\mathcal{M}_{attr}|}$$

After computing the pair-wise similarity for tuples in the canonical relations, we generate the initial tuple matches and their probabilities with the above similarity-to-probability method. In particular, we divide the tuple matches into 50 buckets and we use the evidence mapping in the gold standard to label a sample of matches and produce the probabilities of the buckets. The sizes of the initial tuple matches for each of the datasets are shown in Figure 4.

Academic datasets				
	# of undergrad majors		# of undergrad majors	
	UMass	NCES	OSU	NCES
$N/\lvert P\rvert/\lvert T\rvert$	113/113/95	239K/81/81	282/282/206	239K/153/153
$\lvert\mathcal{M}_{tuple}\rvert$	169		607	
$\lvert\mathcal{M}^*_{tuple}\rvert$	71		140	
$\lvert E\rvert \to \lvert E_S\rvert$	$64 \to 11$		$127 \to 16$	

IMDb datasets					
	Q_1	Q_2	Q_3	Q_4	Q_5
$\lvert P\rvert$ (IMDb1/IMDb2)	1.3K/4.6K	2.8K/3.8K	1.6K/3.1K	2.7K/6.2K	8.9K/9.0K
$\lvert\mathcal{M}_{tuple}\rvert$	0.6M	0.8M	51K	0.3M	1.1M
$\lvert\mathcal{M}^*_{tuple}\rvert$	1271	2768	1601	2756	4231
$\lvert E\rvert \to \lvert E_S\rvert$	3.4K→ 33	1.1K→ 23	1.5K→ 28	3.4K→ 38	5.5K→ 43
	Q_6	Q_7	Q_8	Q_9	Q_{10}
$\lvert P\rvert$ (IMDb1/IMDb2)	5.8K/5.9K	10.9K/10.9K	3.4K/3.5K	4.8K/4.9K	11.5K/14.4K
$\lvert\mathcal{M}_{tuple}\rvert$	0.5M	2.2M	0.2M	0.4M	1.3M
$\lvert\mathcal{M}^*_{tuple}\rvert$	5353	6259	2365	3147	5959
$\lvert E\rvert \to \lvert E_S\rvert$	1.3K→ 19	21.1K→ 86	2.5K→ 33	3.9K→ 40	13.4K→ 75

Figure 4: Dataset statistics. N, $|P|$, $|T|$ are the original data size, the provenance relation size, and the canonical relation size, respectively; the size of the initial tuple mapping is $|\mathcal{M}_{tuple}|$; the sizes of the optimal evidence mapping and the optimal explanations are $|\mathcal{M}^*_{tuple}|$ and $|E|$, respectively. $|E_S|$ is the size of the explanations after summarizing them with Data X-Ray [50,51]. N for D_{IMDb1} and D_{IMDb2} are 3.7M and 6.8M tuples, respectively, for all IMDb queries. In the IMDb datasets, we show the average numbers over 10 instantiations of each query; $|P|$, $|T|$ in these datasets are the same, so we report only one.

UMass vs. NCES	**OSU vs. NCES**
(Major.Major) $\sqsubseteq$ (Stats.Program)	(Major.Major) $\sqsubseteq$ (Stats.Program)
IMDb View 1 vs. IMDb View 2	
(Movie.title, Movie.release_year) $\equiv$	(Movie.title, Movie.release_year)
(Actor.firstname, Actor.lastname, Actor.gender, Actor.dob) $\equiv$	(Person.name, Person.gender, Person.dob)
(Director.firstname, Director.lastname, Director.gender, Director.dob) $\equiv$	(Person.name, Person.gender, Person.dob)

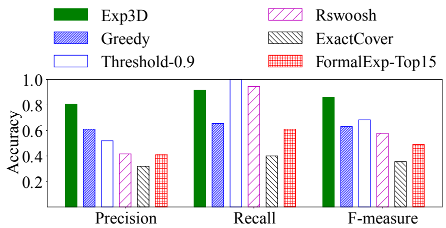
Figure 5: Attribute matches for the real-world datasets.

5.1.3 Algorithms

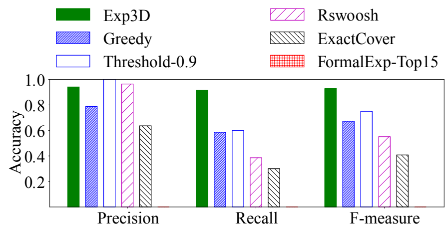
We compare our framework, explain3D, against FormalExp, an approach that focuses on explanations in the single dataset setting, RSwoosh, a state-of-the-art record linkage system, and three additional baseline methods. We describe all the algorithms below.

FormalExp: FormalExp explains surprising outcomes of aggregate queries in a single database [45]. To apply FormalExp in disjoint datasets, we first compare the results of the queries and then ask FormalExp to explain why the query result is high (or low) on each individual dataset. Tuples that are included by the derived explanations are considered provenance-based explanations. FormalExp returns the Top-k explanations, and requires k as an input. In our experiments, we set $k = 15$, denoted by FormalExp-Top15, as it achieves the highest overall accuracy.

RSwoosh: RSwoosh [5] is an entity resolution technique that produces deterministic tuple matches. For RSwoosh, we treat all derived tuple matches as the evidence mapping since their probabilities are all equal to 1.0. We include tuples that do not have a match in this evidence mapping as provenance-based explanations, and tuples with unequal impacts as value-based explanations. Here



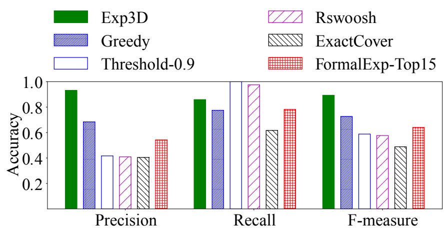
(a) NCES vs. UMass Explanation Accuracy.



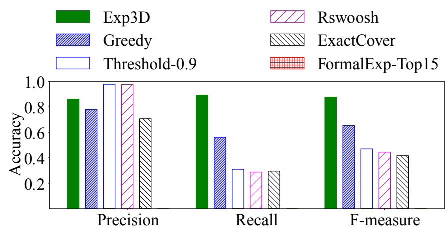
(b) NCES vs. UMass Evidence Accuracy.

Method	NCES/UMass (sec)
FORMALEXP-Top15	0.052
RSWOOSH	0.273
THRESHOLD-0.9	0.276
GREEDY	0.280
EXACTCOVER	0.272
EXPLAIN3D	0.322

(c) NCES vs. UMass Execution Time.



(d) NCES vs. OSU Explanation Accuracy.

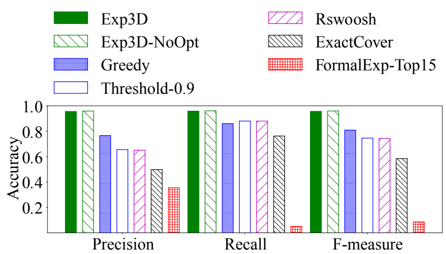


(e) NCES vs. OSU Evidence Accuracy.

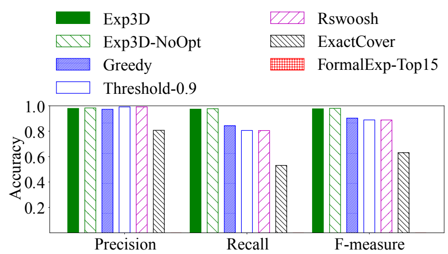
Method	NCES/OSU (sec)
FORMALEXP-Top15	0.064
RSWOOSH	0.541
THRESHOLD-0.9	0.581
GREEDY	0.573
EXACTCOVER	0.562
EXPLAIN3D	0.729

(f) NCES vs. OSU Execution Time.

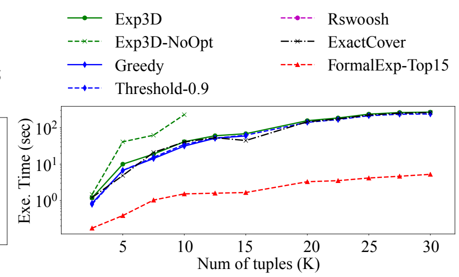
Figure 6: Accuracy and efficiency comparison over Academic datasets. EXPLAIN3D achieves much higher accuracy than the other methods. THRESHOLD obtains high precision but low recall in the derived evidence. FORMALEXP does not provide any tuple matches in the evidence.



(a) Average Explanation Accuracy.



(b) Average Evidence Accuracy.



(c) Average Execution Time.

Figure 7: Accuracy and efficiency comparison over IMDb datasets. EXPLAIN3D achieves near perfect accuracy. RSWOOSH and EXPLAIN3D without the smart-partitioning optimization fails to produce any results for queries with more than $10K$ tuples in 1hr.

we use the Jaccard similarity metric to compare string attributes, using 0.75 as the default threshold value.[13]

THRESHOLD: THRESHOLD is a simple baseline that refines the initial probabilistic tuple matches by a fixed threshold. It uses the derived evidence mapping to derive explanations, in the same manner as RSWOOSH. In our experiment, we set a threshold of 0.9 and denote it as THRESHOLD-0.9.

GREEDY: GREEDY is a baseline that implements explain3D's objective function (Definition 1), but builds the evidence mapping in a greedy fashion (whereas explain3D derives it by solving constrained optimization problems). Initialized with an empty evidence mapping, GREEDY prioritizes tuple matches with higher probabilities and includes into the evidence the match with highest probability that does not violate the valid mapping restriction (Definition 3.2) and improves the objective value. After examining all initial tuple matches, GREEDY finalizes the evidence mapping and creates the explanations in the same way as RSWOOSH and THRESHOLD.

EXACTCOVER: We create a final baseline by adapting the integer programming solution of the Exact Cover problem to solve the EXP-3D problem as follows: we map tuples in one provenance relation as elements, and tuples in the other provenance relation as sets; an element is covered by a set if there exists an initial tuple mapping between their corresponding tuples. We further adapt the objective function of the Exact Cover problem from a decision problem to a optimization problem, where we want to find a collection of sets such that the total number of covered sets and elements is maximized.

EXPLAIN3D: Our proposed system, explain3D, expresses and optimizes the problem as linear constraints and solves the constructed MILP problem(s) through a MILP solver (Section 3, Section 4).

5.1.4 Metrics

Explanation accuracy: We evaluate the explanation accuracy of the algorithms using precision, recall, and F-measure. We calculate precision as the fraction of true explanations over derived explanations, and recall as the fraction of true explanations over the gold standard; F-measure is the harmonic mean of precision and recall ($\frac{2*precision*recall}{precision+recall}$).

Evidence accuracy: We also evaluate the evidence mapping accuracy with the same metrics. Similarly, we calculate the precision as the fraction of true tuple matches over the refined tuple matches, and recall as the fraction of true matches over the gold standard; F-measure as the harmonic mean of precision and recall.

Execution time: We evaluate the efficiency of all alternative algorithms through their total execution times, including the time for generating initial tuple matches.

5.2 Real-world datasets

We evaluate all the algorithms (Section 5.1.3) on both the Academic and IMDb datasets. Figures 6a, 6d, and 7a demonstrate the

[13]We have also conducted experiments using Jaro similarity, but its performance is strictly inferior to Jaccard similarity in all experiments, so we don't report it in the graphs.

precision, recall, and F-measure of the derived explanations; Figures 6b, 6e, and 7b demonstrate the precision, recall, and F-measure of derived evidence mapping; Figures 6c, 6f, and 7c demonstrate the total execution time.

Single-dataset explanations. Our evaluation with FORMALEXP examines whether single-dataset explanation solutions could address explanations across different datasets. This method does not generate an evidence mapping, and the derived explanations focus on why a query result is high or low, rather than why it is higher or lower than the other corresponding query. The why-high/why-low explanation question is a best-effort adaptation of this solution to our problem setting, but it is not a good enough proxy of the correspondence information encoded in the queries. As a result, the f-measure of FORMALEXP-Top15 is low, indicating that it is ill-suited for this problem setting.

Record-linkage approaches. Record-linkage methods do not generate explanations as a goal, but the tuple mappings they produce can be used as an evidence mapping and then mapped to explanations. RSWOOSH and THRESHOLD-0.9 produce evidence mappings with very high precision because they employ thresholds in refining the mappings (thus maintaining the most likely ones). However, their recall is low because they eliminate correct mappings that happen to have low probabilities. As these techniques miss many correct mappings, they include a large number of tuples in the explanations, thus resulting in low explanation precision.

Since RSWOOSH and THRESHOLD-0.9 employ thresholds in refining the mappings, they perform better when the initial mappings are of better quality, as is the case for the IMDb datasets. Their performance drops significantly in the Academic datasets. Through manual analysis, we noted that the initial tuple mappings in the academic data misses or has low probabilities for a significant portion of true matches. For example, the true tuple mapping, ("Foodservice Systems Administration", "Food Business Management") is absent from the initial mapping. Such cases are common in the academic datasets, but uncommon in the IMDb data because movie titles, persons' names, and other attributes are less ambiguous. Further, our view generation and error injection only contributed relatively small perturbations, making matches easier to identify with higher accuracy.

GREEDY is also a record linkage approach, but uses our objective function instead of a strict threshold; thus, it is able to identify a larger portion of true mappings and has a higher recall. However, it may easily reach a local maximum, which results in lower precision and recall on the evidence mapping and further hurts the explanation accuracy. GREEDY is also impacted by the initial mapping quality, but is a bit more robust to it compared to RSWOOSH.

Ultimately, record linkage methods also are oblivious to the correspondence implied by the input queries. Failing to leverage this information, their effectiveness remains relatively low (below 0.8 f-measure), even in the most favorable data settings.

EXPLAIN3D. Our experiments demonstrate that our framework is highly accurate, with respect to both explanations and evidence mappings. Its superior performance compared to the other two categories of approaches is due to two main reasons. First, its objective function is cognizant of the query associations, in that it does not only focus on maximizing the quality of the matched tuples, but also seeks to minimize the unmatched tuples. As a result, it produces smaller explanations and identifies more correct mappings. As an example of the distinction from record linkage, consider two datasets of two tuples: A, B and A', B'. Suppose that the initial probabilistic tuple mapping is $\{(A, A', 0.8)$, $(B, B', 0.8)$, $(A, B', 0.9)$, and $(B, A', 0.5)\}$. Typical record linkage methods would select (A, B') as the single match, because it maximizes the probability of the matched tuples. In contrast, EXPLAIN3D will derive the correct true mappings, (A, A') and (B, B'), because it considers explanation optimality by avoiding un-matched tuples. Second, record linkage methods often consider unmatched values as a very negative signal for matching a pair of tuples. In contrast, EXPLAIN3D does not weigh these mismatches as negatively, as it considers them as possible value-based explanations. As a result, EXPLAIN3D is more robust to variations in the quality of the initial tuple mapping. Nevertheless, the quality of the initial mapping does play a role, thus EXPLAIN3D performs better on the IMDb data than the academic datasets. However, in all cases, its accuracy is superior to the other methods.

Finally, while Exact Cover relates to EXP-3D through the NP-completeness reduction, it performs badly in all settings. This is expected since the Exact Cover problem does not consider tuple impacts, and does not refine the quality of the initial tuple mappings.

Efficiency. We show the total execution time of all methods in Figures 6c, 6f, and 7c. All methods are very efficient, with under a second runtimes. THRESHOLD, GREEDY, RSWOOSH, EXACTCOVER, and EXPLAIN3D rely on the same procedure to derive the input tuple matches, which takes more than 98% of their total execution time. EXACTCOVER scales better than the unoptimized version of EXPLAIN3D, because it has simpler problem settings. Figure 7c also demonstrates the effect of partitioning on the IMDb data. Partitioning allows EXPLAIN3D to scale effectively, without impact on its accuracy (Figures 7a and 7b).

5.3 Synthetic datasets

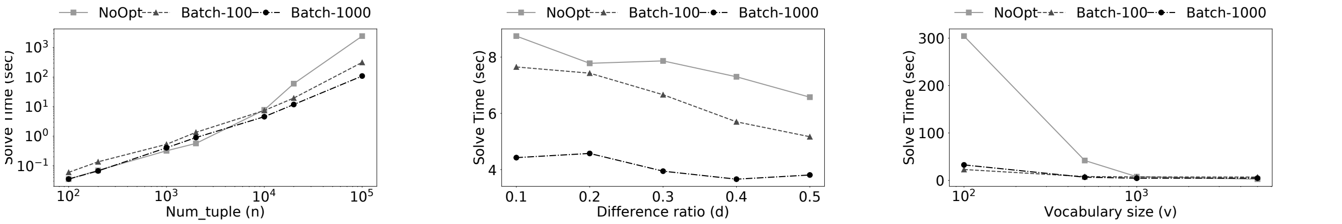
To stress-test EXPLAIN3D and evaluate its smart-partitioning optimization, we create a synthetic data generator to produce datasets and queries with diverse properties. In the synthetic data generator, we use the same schema and queries for every pair of datasets:

Dataset 1	***Dataset 2***
Table(<u>id</u>, match_attr, val)	*Table(<u>id</u>, match_attr, val)*
(match_attr) $\equiv$	*(match_attr)*
Q_1 : `SELECT SUM(val) FROM Table;`	Q_2 : `SELECT SUM(val) FROM Table;`

Based on the above schema, we follow three steps to produce a pair of datasets with the specified properties: (1) We first create n tuples with random attribute values and add them to both datasets. (2) We then randomly drop d percent of tuples, with uniform probability across tuples. (3) We randomly select d percent of tuples, again with uniform probability, and corrupt the tuples' "val" attribute. To generate random values in the "match_attr" attribute, we first create a vocabulary containing $v > 5$ random words and then generate phrases, each of which consists of 5 random words from the vocabulary, as the attribute values; To generate random values in the "val" attribute, we randomly select an integer in the range of $[1, 10]$. The optimal explanations include tuples we dropped or corrupted in the steps (2) and (3); the optimal evidence can be easily derived from step (1). In this experiment, we study the performance of smart-partitioning by dynamically changing the number of partitions ($k \in \mathbb{N}_{>1}$, Definition 2) using a fixed batch size: $k = \lceil \frac{|T_1|+|T_2|}{batch\_size} \rceil$.

We evaluate EXPLAIN3D on three different settings: (1) the basic algorithm without the smart-partitioning optimization (NOOPT), (2) the optimized algorithm with batch size 100 (BATCH-100), and (3) the optimized algorithm with batch size 1000 (BATCH-1000). Figure 8 demonstrates the performance of NOOPT, BATCH-100, and BATCH-1000 over diverse parameter settings.

Adjusting number of tuples (n): We first adjust the number of tuples (n) in the synthetic datasets from 100 to $100K$ with fixed



(a) Solve time vs. # of tuples (n). **(b) Solve time vs. difference ratio (p).** **(c) Solve time vs. vocabulary size (k).**

Figure 8: Efficiency performance of NoOpt, Batch-100, and Batch-1000 over synthetic datasets with diverse properties. Note that we only evaluate the solve time instead of the total execution time since the all methods share the same initial tuple matches generation time.

difference ratio $d = 0.2$ and vocabulary size $v = 1K$. As shown in Figure 8a, NoOpt performs well for problems with fewer tuples as the problem can be efficiently solved by a single MILP problem. However, its execution time grows quadratically, if not exponentially, with increasing data size. Batch-100 and Batch-1000 solve multiple MILP problems with bounded sizes, thus their solve time grows linearly with increasing number of tuples. Meanwhile, Batch-1000 is significantly more efficient than Batch-100 as Batch-100 requires longer time to initialize and solve each individual sub-problems. With the smart-partitioning optimization, Batch-1000 is more than $20\times$ faster than NoOpt on problems with $100K$ tuples.

Adjusting difference ratio (d): We next adjust the difference ratio (d) from 0.1 to 0.5 while keeping the other parameters fixed: $n = 1K, v = 1K$. As expected, all three methods require longer time for problems with lower difference ratio. This is because with higher difference ratio, there will be fewer tuples remaining in the datasets. Again, Batch-1000 is much more efficient than Batch-100 and NoOpt.

Adjusting vocabulary size (v): Finally, we adjust the vocabulary size (v) from 100 to $10K$ and keep $n = 1K, d = 0.2$. In the synthetic data generator, we generate the value of the "match_attr" attribute by randomly selecting 5 words from the vocabulary. Thus the probability that two tuples share at least one common word increases with lower vocabulary sizes. In other words, there will be many more initial tuple matches when we set $v = 100$ than $v = 10K$. As shown in Figure 8c, Batch-100 is $15\times$ faster than NoOpt and even outperforms Batch-1000 when $v = 100$. This is because the number of tuple matches in each sub-problem also affects the problem's overall complexity. Thus, we need to divide the problem into smaller partitions when there is a larger number of initial tuple matches. With increasing vocabulary size (and decreasing number of tuple matches), Batch-1000 starts to outperform the other two methods. When we increase the vocabulary size to a large enough number, e.g., $v = 10K$, NoOpt, Batch-1000, Batch-100 start to perform similarly.

In all experiments on the synthetic datasets, NoOpt, Batch-100, and Batch-1000 achieve near perfect accuracy in the derived explanations and evidence mapping.

6. RELATED WORK

In this paper, we study the problem of explaining the disagreements in the results of semantically similar queries over disjoint datasets. While there is a growing body of work in data management research on deriving explanations, existing work focuses on one dataset at a time, and cannot address disagreements across datasets with potentially different schemas. Explain3D is, to the best of our knowledge, the first framework of its kind, that handles disagreements across disjoint datasets.

Data management research on explanations has focused on the assumption that data resides in a single dataset. The Scorpion system [56] finds predicates on the input data as explanations for a labeled set of outlier points in an aggregate query over a single relation. Roy and Suciu [45] extended explanations with a formal framework that handles complex SQL queries and database schemas involving multiple relations and functional dependencies. This explanation tool does not require any preparation for the data and derives the explanations as a set of conjunctive predicates. Roy, Orr and Suciu [44] further extend their work to provide richer and more insightful explanations on datasets with prepared candidate explanations derived by domain experts.

Other explanation work investigates the absence of answers from a query result [14,47,49]; these systems provide why-not explanations and sometimes modification suggestions to the queries. Work on provenance and causality [12, 25, 39] focuses on identifying the tuples that contribute to a query, and quantify their contributions. Finally, application-specific explanations focus on a particular domain, such as performance of MapReduce jobs [35], item rating [15,48], and auditing and security [4,22].

To compare two semantically similar queries and the corresponding databases, explain3D leverages existing schema matching techniques [7,18,28,38] to derive the correspondence among attributes in two semantically correlated schemas. Existing schema matching solutions leverage a wide variety of techniques, from heuristics [18], to rules [38], to learning-based approaches [7,28].

Another essential input for explain3D is the initial tuple matches (or the tuple mapping). We may acquire such initial tuple matches by leveraging existing entity resolution (or record linkage) techniques [5,8,16,20,54]. More specifically, explain3D treats existing entity resolution approaches as blackboxes and uses them to derive the matches and include them as part of the input.

7. SUMMARY OF CONTRIBUTIONS

In this paper, we presented an effective and scalable framework, explain3D, that derives explanations for the disagreements between the results of two semantically similar queries over two disjoint datasets. Our work formalized several important concepts and essential properties that explanations should satisfy. Explain3D uses a novel formalization and models explanations as two generic types, provenance-based explanations and value-based explanations, and evaluates the quality of explanations through a probabilistic model. The core stage of explain3D is a translation of the explanation problem into a mixed integer linear program, allowing the use of modern constrained solvers to address it. Our work further introduced a smart-partitioning optimization that allows explain3D to scale to large data sizes. To the best of our knowledge, explain3D is the first explanation framework that can address disagreeing query results across disjoint datasets.

Acknowledgements: This material is based upon work supported by the NSF under grants CCF-1763423 and IIS-1453543.

8. REFERENCES

[1] P. C. Arocena, B. Glavic, G. Mecca, R. J. Miller, P. Papotti, and D. Santoro. Messing up with BART: error generation for evaluating data-cleaning algorithms. *PVLDB*, 9(2):36–47, 2015.

[2] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *SIGMOD*, pages 906–908. ACM, 2005.

[3] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[4] G. Bender, L. Kot, and J. Gehrke. Explainable security for relational databases. In *SIGMOD*, pages 1411–1422, 2014.

[5] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *PVLDB*, 18(1):255–276, 2009.

[6] J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *CAiSE*, pages 452–466. Springer-Verlag, 2002.

[7] J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *International Conference on Advanced Information Systems Engineering*, pages 452–466. Springer, 2002.

[8] I. Bhattacharya and L. Getoor. A latent dirichlet model for unsupervised entity resolution. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 47–58, 2006.

[9] J. Bleiholder and F. Naumann. Data fusion. *ACM Computing Surveys (CSUR)*, 41(1):1, 2009.

[10] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154. ACM, 2005.

[11] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent advances in graph partitioning. In *Algorithm Engineering*, pages 117–158. Springer, 2016.

[12] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[13] A. Chapman and H. Jagadish. Why not? In *SIGMOD*, pages 523–534. ACM, 2009.

[14] A. Chapman and H. V. Jagadish. Why not? In *SIGMOD*, pages 523–534, 2009.

[15] M. Das, S. Amer-Yahia, G. Das, and C. Yu. MRI: Meaningful interpretations of collaborative ratings. *PVLDB*, 4(11):1063–1074, 2011.

[16] J. Davis, I. Dutra, D. Page, and V. Santos Costa. Establishing identity equivalence in multi-relational domains. In *Proceedings of the International Conference on Intelligence Analysis*, 2005.

[17] D. Deutch and A. Gilad. Qplain: Query by explanation. In *ICDE*, pages 1358–1361. IEEE, 2016.

[18] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: discovering complex semantic matches between database schemas. In *SIGMOD*, pages 383–394. ACM, 2004.

[19] W. B. Dolan and C. Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[20] P. Domingos. Multi-relational record linkage. In *In Proceedings of the KDD-2004 Workshop on Multi-Relational Data Mining*. Citeseer, 2004.

[21] X. L. Dong and F. Naumann. Data fusion: resolving data conflicts for integration. *PVLDB*, 2(2):1654–1655, 2009.

[22] D. Fabbri and K. LeFevre. Explanation-based auditing. *PVLDB*, 5(1):1–12, 2011.

[23] A. Fariha, S. M. Sarwar, and A. Meliou. SQuID: Semantic similarity-aware query intent discovery. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1745–1748. ACM, 2018.

[24] D. Firmani, B. Saha, and D. Srivastava. Online entity resolution using an oracle. *PVLDB*, 9(5):384–395, 2016.

[25] C. Freire, W. Gatterbauer, N. Immerman, and A. Meliou. A characterization of the complexity of resilience and responsibility for self-join-free conjunctive queries. *PVLDB*, 9(3):180–191, 2015.

[26] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *European semantic web symposium*, pages 61–75. Springer, 2004.

[27] L. Golab, H. J. Karloff, F. Korn, and D. Srivastava. Data Auditor: Exploring data quality and semantics using pattern tableaux. *PVLDB*, 3(2):1641–1644, 2010.

[28] M. C. Hansen, R. S. DeFries, J. R. Townshend, and R. Sohlberg. Global land cover classification at 1 km spatial resolution using a classification tree approach. *International journal of remote sensing*, 21(6-7):1331–1364, 2000.

[29] IBM CPLEX: High-performance software for mathematical programming and optimization, 2005.

[30] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[31] G. Karypis, E.-H. Han, and V. Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[32] G. Karypis and V. Kumar. Multilevel graph partitioning schemes. In *ICPP (3)*, pages 113–122, 1995.

[33] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[34] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[35] N. Khoussainova, M. Balazinska, and D. Suciu. PerfXplain: debugging mapreduce job performance. *PVLDB*, 5(7):598–609, 2012.

[36] S. Kolahi and L. V. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory*, pages 53–62. ACM, 2009.

[37] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69(2):197–210, 2010.

[38] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *vldb*, volume 1, pages 49–58, 2001.

[39] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.

[40] R. J. Miller. Open data integration. *PVLDB*, 11(12):2130–2139, 2018.

[41] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten. Scalable molecular dynamics with NAMD.

Journal of computational chemistry, 26(16):1781–1802, 2005.

[42] N. Prokoshyna, J. Szlichta, F. Chiang, R. J. Miller, and D. Srivastava. Combining quantitative and logical data cleaning. *PVLDB*, 9(4):300–311, 2015.

[43] F. Psallidas, B. Ding, K. Chakrabarti, and S. Chaudhuri. S4: Top-k spreadsheet-style search for query discovery. In *SIGMOD*, pages 2001–2016. ACM, 2015.

[44] S. Roy, L. Orr, and D. Suciu. Explaining query answers with explanation-ready databases. *PVLDB*, 9(4):348–359, 2015.

[45] S. Roy and D. Suciu. A formal approach to finding explanations for database queries. In *SIGMOD*, pages 1579–1590. ACM, 2014.

[46] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. Discovering queries based on example tuples. In *SIGMOD*, pages 493–504. ACM, 2014.

[47] B. ten Cate, C. Civili, E. Sherkhonov, and W. C. Tan. High-Level Why-Not Explanations using Ontologies. In *PODS*, pages 31–43, 2015.

[48] S. Thirumuruganathan, M. Das, S. Desai, S. Amer-Yahia, G. Das, and C. Yu. MapRat: meaningful explanation, interactive exploration and geo-visualization of collaborative ratings. *PVLDB*, 5(12):1986–1989, 2012.

[49] Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *SIGMOD*, pages 15–26. ACM, 2010.

[50] X. Wang, X. L. Dong, and A. Meliou. Data X-Ray: A diagnostic tool for data errors. In *SIGMOD*, pages 1231–1245. ACM, 2015.

[51] X. Wang, M. Feng, Y. Wang, L. Dong, and A. Meliou. Error diagnosis and data profiling with Data X-Ray. *PVLDB*, 8(12):1984–1987, 2015.

[52] X. Wang, L. Haas, and A. Meliou. Explaining data integration. *Data Engineering Bulletin*, 41(2), 2018.

[53] X. Wang and A. Meliou. Explain3D: Explaining disagreements in disjoint datasets. *CoRR*, abs/1903.09246, 2019.

[54] S. E. Whang and H. Garcia-Molina. Entity resolution with evolving rules. *PVLDB*, 3(1-2):1326–1337, 2010.

[55] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.

[56] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, 2013.

[57] C. J. Zhang, L. Chen, H. V. Jagadish, and C. C. Cao. Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 6(9):757–768, 2013.