Explaining Differences Between Unaligned Table Snapshots

Manuel Fink
Data and Web Science Group
University of Mannheim
manuel@informatik
.uni-mannheim.de

Christian Meilicke
Data and Web Science Group
University of Mannheim
christian@informatik
.uni-mannheim.de

Heiner Stuckenschmidt
Data and Web Science Group
University of Mannheim
heiner@informatik
.uni-mannheim.de

ABSTRACT

We study the problem of explaining differences between two snapshots of the same database table including record insertions, deletions and in particular record updates. Unlike existing alternatives, our solution induces transformation functions and does not require knowledge of the correct alignment between the record sets. This allows profiling snapshots of tables with unspecified or modified primary keys. In such a problem setting, there are always multiple explanations for the differences. Our goal is to find the simplest explanation. We propose to measure the complexity of explanations on the basis of minimum description length in order to formulate the task as an optimization problem. We show that the problem is NP-hard and propose a heuristic search algorithm to solve practical problem instances. We implement a prototype called Affidavit to assess the explanatory qualities of our approach in experiments based on different real-world data sets. We show that it can scale to both a large number of records and attributes and is able to reliably provide correct explanations under practical levels of modifications.

1 INTRODUCTION

When the content of a database table is frequently changing, it is difficult to find and understand the differences manually. For this reason, a large number of tools has been developed with the goal of supporting database administrators in situations like these [1, 7–9, 24]. Most of them cannot only identify deleted and inserted records but also highlight changes of individual attribute values of records that exist in both snapshots. However, the existing solutions share a big limitation. They require knowledge of the correct record alignment, usually derived from primary key attributes. In certain use cases though, immutability of primary keys is not a valid assumption. Our research is motivated by a use case of an industry project that aims to understand database updates caused by proprietary software updates. We found existing solutions not well suited because keys of the same records sometimes get reassigned during the update.

Figure 1 serves as a running example for such a problem instance. It shows two table snapshots $\mathcal{S}_1$ and $\mathcal{T}_1$ whose uncolored records have been deleted and inserted respectively. Equally colored records resemble a correctly aligned pair of records in which the record from $\mathcal{T}_1$ was derived from the record in $\mathcal{S}_1$ with the transformations specified below the table.

Snapshots $\mathcal{S}_1$ and $\mathcal{T}_1$ could belong to a company's ERP system whose database was transformed as part of an update to a newer software revision. While the attribute value changes were likely done to meet a new data format specification, the deletions and insertions constitute changes of the table content or noise from continued use of both databases between transformation and snapshotting. The company might be interested in an explanation for the changes because the conversion script is unavailable, proprietary or legacy code that is difficult to understand. A direct benefit of reverse-engineering the transformation is that, additional full system conversions can be avoided if more data needs to be transformed later on, reducing both costs and downtime.

Other application domains include data integration, e.g. duplicate detection when integrating multiple sources with redundant records in the target schema, as well as analysis of changes of third-party data sources without access to the transaction log.

What makes the running example interesting, are the changes to the composite primary key {*ID1, ID2, Date*} that make it necessary to identify other suitable attributes for record linking. *ID2* looks very promising because it is part of the primary key and has perfect discriminability and coverage [3]: The provenance of every single target record is reduced to exactly one source record. However, the correct alignment shows that these characteristics can be highly misleading. *ID2* in $\mathcal{T}_1$ was most likely filled using a skolem function [2] as part of an auto-increment policy. Linking with *Date* is another promising option, yet it would fail to explain the provenance of the three records *T13, T14, T15* in which *'99991231'* in *Date* was replaced by *'20180701'*. On the other hand, once the correct transformation function for Val has been learned, it would be very helpful for aligning the records without missing out on these three pairs. Learning this function without the alignment is difficult though.

Intuitively, we can expect at least *some* attributes to be unchanged in practice and use them to partially resolve the alignment problem. For example, *Type* and *Org* suggest an alignment of records *S11* and *T13*. The division function of *Val* implied by the input-output example *'65'* $\mapsto$ *'0.065'* generalizes to other alignment clusters, too, often resolving them.

Extending snapshot comparison with record linking and function synthesis creates a challenging duality. Scalable unsupervised record linking methods need domain knowledge on how to use the attributes to cluster records into blocks that are small enough for detailed similarity comparisons. In the case of attributes whose values have been systematically changed, algorithms that induce string transformations from examples are needed to learn how to use the attribute for blocking. However, the records need to be aligned already to produce the required input-output examples. Hence, these two sub-problems affect each other and cannot be solved independently.

The core of our contribution is an unsupervised search algorithm that iteratively learns which attributes have likely been changed and induces the corresponding value transformation functions. The resulting solution can deal with transformed or unspecified primary keys and produces more than a report of the differences. It yields an explanation that can be used to transform additional, unseen records of the source table because it generalizes the value changes instead of only listing them.

ID1	ID2	Date	Type	Val	Unit	Org
S01	**0000**	**20130416**	A	80000	USD	IBM
S02	**0001**	**20120128**	A	180000	USD	IBM
S03	**0002**	**20130315**	A	220000	USD	IBM
S04	**0003**	**20120128**	B	3780000	USD	IBM
S05	**0004**	**20120731**	B	425000	USD	IBM
S06	**0005**	**20120731**	C	21000	USD	IBM
S07	**0006**	**20140503**	C	422400	USD	IBM
S08	**0007**	**20140503**	C	6540	USD	SAP
S09	**0008**	**20131021**	C	9800	USD	SAP
S10	**0009**	**20121125**	C	0	USD	SAP
S11	**0010**	**99991231**	D	65	USD	SAP
S12	**0011**	**99991231**	D	180000	USD	BASF
S13	**0012**	**99991231**	D	220000	USD	BASF
S14	**0013**	**20150203**	D	21000	USD	BASF
S15	**0014**	**20150213**	D	65	USD	BASF
S16	**0015**	**20160807**	E	80000	USD	BASF
S17	**0016**	**20161231**	E	80000	USD	BASF

Source table $\mathcal{S}_1$

ID1	ID2	Date	Type	Val	Unit	Org
T01	**0000**	**99991231**	A	80	k \$	IBM
T02	**0001**	**20120128**	A	180	k \$	IBM
T03	**0002**	**20120731**	C	21	k \$	IBM
T04	**0003**	**20120731**	B	425	k \$	IBM
T05	**0004**	**20121125**	B	0.022	k \$	DAB
T06	**0005**	**20130315**	A	220	k \$	IBM
T07	**0006**	**20130416**	A	80	k \$	IBM
T08	**0007**	**20131021**	C	9.8	k \$	SAP
T09	**0008**	**20140503**	C	422.4	k \$	IBM
T10	**0009**	**20140503**	C	6.54	k \$	SAP
T11	**0010**	**20150213**	D	0.065	k \$	BASF
T12	**0011**	**20161231**	E	80	k \$	BASF
T13	**0012**	**20180701**	D	0.065	k \$	SAP
T14	**0013**	**20180701**	D	180	k \$	BASF
T15	**0014**	**20180701**	D	220	k \$	BASF
T16	**0015**	**99991231**	F	0.45	k \$	SAP

Target table $\mathcal{T}_1$

$$\mathcal{F}^{\mathcal{E}_1} = \left(\begin{array}{ll} f^{\mathcal{E}_1}_{ID1}: & \{S01 \mapsto T07,\quad S02 \mapsto T02,\quad S03 \mapsto T06,\quad S05 \mapsto T04,\quad S06 \mapsto T03,\quad S07 \mapsto T09,\quad S08 \mapsto T10, \\ & S09 \mapsto T08,\quad S11 \mapsto T13,\quad S12 \mapsto T14,\quad S13 \mapsto T15,\quad S15 \mapsto T11\quad S17 \mapsto T12\}, \\ & \{0000 \mapsto 0006,\quad 0001 \mapsto 0001,\quad 0002 \mapsto 0005,\quad 0004 \mapsto 0003,\quad 0005 \mapsto 0002,\quad 0006 \mapsto 0008, \\ f^{\mathcal{E}_1}_{ID2}: & 0007 \mapsto 0009,\quad 0008 \mapsto 0007,\quad 0010 \mapsto 0012,\quad 0011 \mapsto 0013,\quad 0012 \mapsto 0014,\quad 0014 \mapsto 0010, \\ & 0016 \mapsto 0011\},\qquad f^{\mathcal{E}_1}_{Date}: \quad \text{‘9999123’}x \mapsto \text{‘2018070’}x, \quad otherwise \quad x \mapsto x, \\ f^{\mathcal{E}_1}_{Type}: & x \mapsto x,\qquad f^{\mathcal{E}_1}_{Val}: \quad x \mapsto x\ /\ 1000,\qquad f^{\mathcal{E}_1}_{Unit}: \quad x \mapsto \text{’k \$’},\qquad f^{\mathcal{E}_1}_{Org}: \quad x \mapsto x \end{array} \right)$$

Figure 1: Problem Instance $\mathcal{I}_1 = (\mathcal{S}_1, \mathcal{T}_1, \mathcal{A}_1, \mathcal{F}_1)$ that shows the content of two snapshots of the same table. Primary key attributes are bolded. The attribute functions specified in $\mathcal{F}^{\mathcal{E}_1}$ are part of a possible explanation $\mathcal{E}_1$ for the changes that uses the colored records in $\mathcal{S}_1$ to create the records of the same color in $\mathcal{T}_1$. Note that the alignment of the colored records is also given by $f^{\mathcal{E}_1}_{ID1}$. Uncolored records are records which $\mathcal{E}_1$ explains as deleted and inserted respectively.

2 RELATED WORK

The problem presented in this work extends the classic task of reporting differences of table snapshots in two dimensions: record linking and function synthesis. Handling both efficiently at once is not as trivial as combining the best solutions of both fields. It is not even straight-forward how to apply the respective state-of-the-art techniques at all in this context due to their requirements. Our contribution is an exploration of the intersection of the two problems. It is not our goal to improve the state-of-the-art in either of these fields. Instead, we aim to solve them in the context of a comparison tool that requires minimal user effort to make it practical to profile database snapshots with hundreds of tables.

Table Comparison Tools In the industry, there is a high demand for database analysis software which resulted in a large number of both free and commercial solutions for the comparison of relational database tables. Exploring this market, we found, among others, options such as ApexSQL Data Diff [1], Replicator [8], Redgate SQL Data Compare [24], Devart Data Compare [9] or SQL Delta [7]. However, to the best of our knowledge, they can only be used on tables for which a primary key is specified and none of the available products is able to cope with changes of the primary key attributes as it is common standard to use them to link the records for comparison. If they do include functionality to link records in a different way, it requires manual effort by the user, for example by explicitly defining linkage rules. Furthermore, while most of the products are able to export executable SQL scripts that implement the transformation of the data, they do not generalize well to unknown records because the value changes are explicitly stated per record and there is no learning of systematic transformation functions on the level of attributes. As a consequence, the generated reports lack an explanation of the changes in case a systematic pattern exists.

Record Linking Record linking has been frequently studied in academic research and is also known as record, entity or instance matching, identity resolution or deduplication [5]. There are several solutions available that implement both supervised and unsupervised algorithms for linking structured data, for example Magellan [17] inlcuding DeepMatcher [20], JedAI [21], dedupe [4], SILK [16], or WInte.R [18].

In a supervised setting, a set of annotated examples is given. Each example corresponds to a record described in two different representations that usually share some attributes but not necessarily the whole schema. While powerful, we do not consider techniques centered around supervision as users typically use comparison tools as a first attempt to understand changes in large amounts of data without investing manual annotation effort.

JedAI is a suite of unsupervised algorithms that do not require an annotated training dataset to link records. It contains methods for data cleaning, blocking and matching that can be configured by a user to guide the matching process. JedAI offers a rather generic approach where it is not required to define attribute-specific similarity functions. However, the lack of annotated examples means, the user needs to choose a suitable configuration of the different algorithms manually. This depends on the individual data and requires domain knowledge. Our solution is unsupervised without user guidance by using a universally applicable cost function to search for good linking criteria.

A major difference to both supervised and unsupervised approaches, is that we aim to learn transformation functions that transform records from one representation to the other. At the same time, this yields a strong criterion for the task of separating deleted or inserted records from transformed data. Most record linking algorithms however, link records purely based on a fuzzy notion of similarity. They are designed to support use cases in which no transformation function might exist, e.g. linking data from different sources.

Induction of Transformation Functions Learning string manipulations from input-output examples is a research field that has real-world applications in widespread tasks such as data preparation and spreadsheet manipulation. This is exemplified by several works of Gulwani et al. [12–14, 23] that laid the foundations for different add-ins of Microsoft Excel and Azure, marketed under the names QuickCode and FlashFill. FlashMeta [23] on the other hand, is a framework into which the authors were able to cast several different instances of the problem by separating the induction algorithm from the domain-specific language of the transformations.

The language of transformation functions that can be induced by these algorithms spans a subset of regular expressions that includes loops and conditionals. These kinds of transformations are more expressive but have many more parameters than the function families we consider in this work. As it is usually impossible to learn all function parameters from a single input-output example, a set of examples is used to unambiguously induce a specific function. Typically, a user is supposed to give a handful of examples.

Unfortunately, the authors in [26] found that current methods do not scale well to a large number of examples which led them to develop an iterative algorithm that re-uses intermediate search results from previous examples. The third-party scala re-implementation of FlashFill [19] that we were able to try was indeed too slow to be used on large tables as its induction time was in the range of seconds for a single example. On the other hand, there is no mention in [26] of how it deals with mislabeled or noisy examples, leading us to belief that it is not well suited for such scenarios. While in our setting one can reasonably expect to be able to provide a set of input-output examples that includes some correct ones, the examples can be extremely noisy due to record deletions and insertions as well as the unknown alignment of records. This makes it difficult to use state-of-the-art techniques that support complex transformations if the goal is to scale to large tables.

An alternative to induction for learning transformation functions, is the retrieval of a fitting transformation from a corpus. TDE [15] follows such an approach with $50K$ functions crawled from Github and Stackoverflow and recently showed substantially better performance than induction-based systems.

3 PROBLEM STATEMENT

Given two table snapshots with unaligned records, our goal is to explain the differences with a set of operations that transform the source snapshot to the target snapshot. Allowed operations are record insertions, deletions and transformation functions on an attribute level, for example to express a primary key mapping.

Definition 3.1. **(Problem Instance)** A *problem instance* $\mathcal{I} = (\mathcal{S}, \mathcal{T}, \mathcal{A}, \mathcal{F})$ is a set of source ($\mathcal{S}$) and target ($\mathcal{T}$) records given as value tuples under the same schema $\mathcal{A}$ which is a tuple of d attributes. $\mathcal{F} \supset \{id\}$ contains candidate transformation functions.

We will describe $\mathcal{F}$ implicitly with meta functions (see Table 1) used to solve a problem instance, such as prefixing or integer addition. $\mathcal{F}$ contains all their instantiations (i.e. parameter choices) that transform at least one source value to a target value of the same attribute, e.g. $x \mapsto x + 5$. We do not define a problem instance directly by meta functions because the space of possible explanations depends on the concrete instantiations (see Def. 4.1).

The definition of all other symbols is to be understood in relation to one specific, fixed problem instance. It should be clear from the context which problem instance they refer to.

Definition 3.2. **(Explanation)** An *explanation* is a tuple $\mathcal{E} = (\mathcal{S}^{\mathcal{E}-}, \mathcal{T}^{\mathcal{E}+}, \mathcal{F}^{\mathcal{E}})$ of source records labeled as deleted ($\mathcal{S}^{\mathcal{E}-} \in \mathcal{S}$), target records labeled as inserted ($\mathcal{T}^{\mathcal{E}+} \in \mathcal{T}$) and a tuple $\mathcal{F}^{\mathcal{E}} = (f^{\mathcal{E}}_{a_1}, \dots, f^{\mathcal{E}}_{a_d})$ of attribute functions from $\mathcal{F}$.

Definition 3.3. **(Core)** $\mathcal{S}^{\mathcal{E}} := \mathcal{S} \setminus \mathcal{S}^{\mathcal{E}-}$ is called the *core* of an explanation $\mathcal{E}$.

The core contains all source records which are not labeled as deleted. It is used to produce the target records which are not labeled as inserted.

Definition 3.4. **(Core Image)** The result $\mathcal{T}^{\mathcal{E}}$ of applying the attribute functions of $\mathcal{F}^{\mathcal{E}}$ to the core $\mathcal{S}^{\mathcal{E}}$ is called *core image*:

$$\begin{aligned} \mathcal{T}^{\mathcal{E}} := \mathcal{F}^{\mathcal{E}}(\mathcal{S}^{\mathcal{E}}) \quad &:= \Pi_{f^{\mathcal{E}}_{a_1}, \dots, f^{\mathcal{E}}_{a_d}}(\mathcal{S}^{\mathcal{E}}), \text{ and so for } s \in \mathcal{S}^{\mathcal{E}}: \\ \mathcal{F}^{\mathcal{E}}(s) \quad &= \left(f^{\mathcal{E}}_{a_1}(\Pi_{a_1}(s)), \dots, f^{\mathcal{E}}_{a_d}(\Pi_{a_d}(s))\right). \end{aligned}$$

We are only interested in explanations that state the origin of every single target record, either as the result of transforming a source record or as an insertion. Moreover, we demand that each target record provenance is unambiguous by enforcing $\mathcal{F}^{\mathcal{E}}$ to be a bijection between $\mathcal{S}^{\mathcal{E}}$ and $\mathcal{T}^{\mathcal{E}}$.

Definition 3.5. **(Validity)** An explanation $\mathcal{E}$ is called *valid* if $\mathcal{T}^{\mathcal{E}+} = \mathcal{T} \setminus \mathcal{T}^{\mathcal{E}}$ and $|\mathcal{S}^{\mathcal{E}}| = |\mathcal{T}^{\mathcal{E}}|$. The set of valid explanations for a problem instance $\mathcal{I}$ is denoted by $\mathcal{E}^{\mathcal{I}}$.

From now on, when we talk about explanations, we implicitly mean valid explanations.

PROPOSITION 3.6. *Given a problem instance $\mathcal{I}$ and attribute functions $\mathcal{F}^{\mathcal{E}}$, a valid explanation $\mathcal{E}$ can be constructed from $\mathcal{F}^{\mathcal{E}}$ by appropriately choosing $\mathcal{S}^{\mathcal{E}-}$ and $\mathcal{T}^{\mathcal{E}+}$.*

PROOF. Let $\mathcal{S}^{\mathcal{E}} = \{s \mid s \in \mathcal{S} \text{ and } \mathcal{F}^{\mathcal{E}}(s) \in \mathcal{T}\}$. If multiple core records are transformed into the same target record, remove all but one such record from $\mathcal{S}^{\mathcal{E}}$. This yields $\mathcal{T}^{\mathcal{E}} = \mathcal{F}^{\mathcal{E}}(\mathcal{S}^{\mathcal{E}})$ (Definition 3.4) with $|\mathcal{S}^{\mathcal{E}}| = |\mathcal{T}^{\mathcal{E}}|$. Finally, construct $\mathcal{S}^{\mathcal{E}-} = \mathcal{S} \setminus \mathcal{S}^{\mathcal{E}}$ (Definition 3.3) and $\mathcal{T}^{\mathcal{E}+} = \mathcal{T} \setminus \mathcal{T}^{\mathcal{E}}$ (Definition 3.5). □

We use Figure 1 to elaborate on these definitions. It visualizes some problem instance $\mathcal{I}_1$ with $\mathcal{S}_1$, $\mathcal{T}_1$, $\mathcal{A}_1$ as depicted by the two tables. $\mathcal{F}_1$ could be defined implicitly by the following meta functions which have a varying number of parameters: identity, constant value, division, prefix replacement and value mappings (see Table 1).

The record coloring visualizes a possible explanation $\mathcal{E}_1$. The colored target records are producible from the source records of the same color using the specified attribute functions. $\mathcal{E}_1$ has the following formal components:

$$\begin{aligned} \mathcal{S}^{\mathcal{E}_1-} &= \{s \in \mathcal{S}_1 \mid \Pi_{\mathrm{ID1}}(s) \in \{\mathrm{S10, S04, S14, S16}\}\} \\ \mathcal{T}^{\mathcal{E}_1+} &= \{t \in \mathcal{T}_1 \mid \Pi_{\mathrm{ID1}}(t) \in \{\mathrm{T01, T05, T16}\}\} \\ \mathcal{F}^{\mathcal{E}_1} &= \textit{as shown below the tables in Figure 1} \end{aligned}$$

For instance, applying its attribute functions to the first source record (S01) produces the seventh target record (T07):

$$\begin{aligned} &\mathcal{F}^{\mathcal{E}_1}((\mathrm{S01, 0000, 20130416, A, 80000, USD, IBM})) \\ =\ &(f_{ID1}^{\mathcal{E}_1}(\mathrm{S01}), f_{ID2}^{\mathcal{E}_1}(0000), f_{Date}^{\mathcal{E}_1}(20130416), f_{Type}^{\mathcal{E}_1}(\mathrm{A}), \\ &f_{Val}^{\mathcal{E}_1}(80000), f_{Unit}^{\mathcal{E}_1}(\mathrm{USD}), f_{Org}^{\mathcal{E}_1}(\mathrm{IBM})) \\ =\ &(\mathrm{T07, 0006, 20130416, A, 80, k\ \$, IBM}) \end{aligned}$$

$\mathcal{E}_1$ is a valid explanation because every target record is either producible from exactly one core record or is included in $\mathcal{T}^{\mathcal{E}_1+}$.

3.1 Explanation Quality

The example above can be used to demonstrate that, even if systematic operations are used to change a table, it is in general impossible to be sure about the correct explanation given two snapshots. Besides $\mathcal{E}_1$, there are many more valid explanations using the same meta functions. For instance, the twelfth target record (T12) could also be created from the sixteenth (S16) source record instead of the seventeenth (S17). A second valid explanation $\mathcal{E}_2$ could therefore be constructed by adjusting the set of core records and replacing two value mappings in $f_{ID1}^{\mathcal{E}_1}$ and $f_{ID2}^{\mathcal{E}_1}$. This would not affect the brevity of the explanation. However, we would also have to replace $f_{Date}^{\mathcal{E}_1}$ with a function that additionally maps *20160807* to *20161231*. For this, we could not instantiate a simple meta function anymore and we would need a value mapping. As it would have more parameters than a prefix replacement, explanation $\mathcal{E}_2$ is less intuitive than $\mathcal{E}_1$.

Our formal definition of an explanation's quality is motivated by [11] in which the cost of a schema mapping induced from data instances is measured by the number of variables and constants in its minimum repair. Schema mapping repairs are defined as first-order formulas that state exceptions to the schema mapping to make it fit the given data example. The corresponding concept in our problem are records outside of the core whose origin can not be explained with the induced functions. The central task becomes the balancing of the size of the core with the complexity of the functions which does not have a straight-forward solution. We decide to loosely follow the concept of minimum description length [25] and prefer explanations that maximally compress the problem instance. Specifically, we evaluate the description length of our input data $\mathcal{S}$ and $\mathcal{T}$ under an explanation and ignore the contribution of inputs $\mathcal{A}$ and $\mathcal{F}$ as their description length is independent of the choice of $\mathcal{E}$.

Proposition 3.7. *$\mathcal{S}$ and $\mathcal{T}$ are implicitly described by a valid explanation $\mathcal{E} = (\mathcal{S}^{\mathcal{E}-}, \mathcal{T}^{\mathcal{E}+}, \mathcal{F}^{\mathcal{E}})$ and its core $\mathcal{S}^{\mathcal{E}}$.*

Proof. $\mathcal{S} = (\mathcal{S} \setminus \mathcal{S}^{\mathcal{E}-}) \cup \mathcal{S}^{\mathcal{E}-} \overset{(Def.\ 3.3)}{=} \mathcal{S}^{\mathcal{E}} \cup \mathcal{S}^{\mathcal{E}-}$,

$$\mathcal{T} \overset{(Def.\ 3.5)}{=} \mathcal{T}^{\mathcal{E}+} \cup \mathcal{T}^{\mathcal{E}} \overset{(Def.\ 3.4)}{=} \mathcal{T}^{\mathcal{E}+} \cup \mathcal{F}^{\mathcal{E}}(\mathcal{S}^{\mathcal{E}}) \quad \square$$

Note that every source record is described exactly once (in the two disjoint sets $\mathcal{S}^{\mathcal{E}}$ and $\mathcal{S}^{\mathcal{E}-}$) and the choice of $\mathcal{E}$ only affects the distribution of the source records to these two sets. However, from the records in $\mathcal{T}$, only those contained in $\mathcal{T}^{\mathcal{E}+}$ need to be described. Therefore, an explanation compresses inputs $\mathcal{S}$ and $\mathcal{T}$ if its attribute functions $\mathcal{F}^{\mathcal{E}}$ can be described shorter than the core image $\mathcal{T}^{\mathcal{E}}$ which are the records from $\mathcal{S}$ and $\mathcal{T}$ that can be reconstructed from $\mathcal{E}$ and $\mathcal{S}^{\mathcal{E}}$. For this reason, we optimize the description lengths of $\mathcal{T}^{\mathcal{E}+}$ and $\mathcal{F}^{\mathcal{E}}$ but not $\mathcal{S}^{\mathcal{E}-}$ or $\mathcal{S}^{\mathcal{E}}$.

To measure the description length of the records in $\mathcal{T}^{\mathcal{E}+}$ by strictly following the definition of minimum description length, we would need to determine the minimum number of bits needed to represent $\mathcal{T}^{\mathcal{E}+}$. We decide to loosen this requirement both for semantic and practical reasons. In the context of this work, it is sufficient to count the number of data values that appear in the formal description of an explanation.

Definition 3.8. The description length of the record set $\mathcal{T}^{\mathcal{E}+}$ is defined by $\mathcal{L}(\mathcal{T}^{\mathcal{E}+}) := |\mathcal{A}| \cdot |\mathcal{T}^{\mathcal{E}+}|$.

Concerning the description length of an explanation's attribute functions, it is difficult to find a general definition that captures the brevity of a function's signature. We decide to use the smallest number of parameters that are needed to instantiate the function from a meta function, which again is a count of data values. It shall be denoted by $\psi(f)$.

Definition 3.9. The description length of an explanation's attribute functions $\mathcal{F}^{\mathcal{E}}$ is defined by $\mathcal{L}(\mathcal{F}^{\mathcal{E}}) := \sum_{a \in \mathcal{A}} \psi(f_a^{\mathcal{E}})$.

Definition 3.10. **(Costs of Explanations)** The costs $c(\mathcal{E})$ of an explanation are defined by $c(\mathcal{E}) := 2\alpha\mathcal{L}(\mathcal{T}^{\mathcal{E}+}) + 2(1-\alpha)\mathcal{L}(\mathcal{F}^{\mathcal{E}})$.

Parameter $\alpha \in [0, 1]$ can be used to prioritize one of the components. For example, in the standard setting $\alpha = 0.5$, explanation $\mathcal{E}_1$ of Figure 1 has costs $c(\mathcal{E}_1) = \mathcal{L}(\mathcal{T}^{\mathcal{E}_1+}) + \mathcal{L}(\mathcal{F}^{\mathcal{E}_1}) = 7|\mathcal{T}^{\mathcal{E}_1+}| + \sum_{a \in \mathcal{A}} \psi(f_a^{\mathcal{E}}) = (7 \cdot 3) + (13 \cdot 2 + 13 \cdot 2 + 2 + 0 + 1 + 1 + 0) = 21 + 56 = 77$.

Our cost definition captures two desirable qualities of an explanation. The first term rewards explanations that use a large core to produce a big subset of the records in $\mathcal{T}$. Therefore, explanations that successfully align many records are preferred. The second term promotes simple explanations, as complicated attribute functions with many parameters, such as value mappings, are penalized for their lengthy description.

We can now formally express the requirements of an optimal solution for a problem instance.

Definition 3.11. **(Optimal Solution)** Given a problem instance $\mathcal{I}$, the set of optimal solutions is defined by $\mathcal{E}^* := \underset{\mathcal{E} \in \mathcal{E}^{\mathcal{I}}}{argmin}\ c(\mathcal{E})$. We call the problem of finding such an optimal solution Explain-Table-Delta.

In practice, $\mathcal{E}^*$ is unlikely to contain more than one optimal explanation. In some problem instances though, multiple source records can be used to produce the same target record and more than one provenance leads to an explanation with optimal costs. Note that $\mathcal{E}^{\mathcal{I}} \neq \emptyset$ as $\mathcal{E}_{\emptyset} = (\mathcal{S}, \mathcal{T}, \{id\}^d)$ is a trivial explanation for every problem instance $\mathcal{I}$ that can always be given. It lists all source records as deleted and all target records as inserted. For example, on $\mathcal{I}_1$ and $\alpha = 0.5$, this explanation has costs $|\mathcal{A}_1| \cdot |\mathcal{T}_1| = 7 \cdot 16 = 112$.

3.2 Problem Complexity

Theorem 3.12. ***(NP-Hardness)*** *The problem Explain-Table-Delta is NP-hard for $\alpha > 0$.*

c_i	#	v_1	v_2	v_3	v_4
$v_1 \vee v_2 \vee \overline{v_3}$	c1	1	1	0	-
$\overline{v_1} \vee v_4$	c2	0	-	-	1
v_3	c3	-	-	1	-

Source records $\mathcal{S}$

#	v_1	v_2	v_3	v_4	c_i
c1	0	0	0	-	$v_1 \vee v_2 \vee \overline{v_3}$
c1	0	1	0	-	
c1	0	1	1	-	
c1	1	0	0	-	
c1	1	1	0	-	
c1	1	0	1	-	
c1	1	1	1	-	
c2	0	-	-	0	$\overline{v_1} \vee v_4$
c2	0	-	-	1	
c2	1	-	-	1	
c3	-	-	1	-	v_3

Target records $\mathcal{T}$

Figure 2: Reduction of an example 3-SAT instance $c = (v_1 \vee v_2 \vee \overline{v_3}) \wedge (\overline{v_1} \vee v_4) \wedge v_3$ to a problem instance of Explain-Table-Delta with 3 source and 11 target records.

Proof. *Via polynomial-time reduction from 3-SAT.*
Figure 2 shows an example reduction. Let $c = \bigwedge_{i=1}^{n} c_i$ be an instance of 3-SAT with clauses c_i over variables from a set $\mathcal{V} = \{v_1, ..., v_d\}$. Then, we can create a problem instance $\mathcal{I} = (\mathcal{S}, \mathcal{T}, \mathcal{A}, \mathcal{F})$ for which the optimal solution specifies a model of c if c is satisfiable. For this, we set $\mathcal{A} := (\#, v_1, ..., v_d)$. We let $\mathcal{F}$ contain only two possible attribute functions, *id* ($x \mapsto x$) and *boolean negation* ($x \mapsto \overline{x}$). The latter shall swap the truth values {'0', '1'} and otherwise behave like *id*.

We construct $\mathcal{S}$ to have n records. For each clause c_i, $\mathcal{S}$ contains a record s_i such that $\Pi_\#(s_i) = \text{'c'} \circ i$ and

$$\Pi_{v_j}(s_i) = \begin{cases} \text{'1'}, & v_j \text{ positive in } c_i \\ \text{'0'}, & v_j \text{ negative in } c_i \\ \text{'-'}, & v_j \text{ not in } c_i. \end{cases}$$

$\mathcal{T}$ is constructed to contain a maximum of $7n$ records. For a clause c_i with k literals, the $2^k - 1$ different models over the variables in c_i are used to define one target record each. Let m_k be such a model for clause c_i. Then, the corresponding target record $t_{i,k}$ has $\Pi_\#(t_{i,k}) = \text{'c'} \circ i$. and

$$\Pi_{v_j}(t_{i,k}) = \begin{cases} \text{'1'}, & v_j \text{ positive in } c_i \text{ and } v_j \text{ true in } m_k \text{ or} \\ & v_j \text{ negative in } c_i \text{ and } v_j \text{ false in } m_k \\ \text{'0'}, & v_j \text{ positive in } c_i \text{ and } v_j \text{ false in } m_k \text{ or} \\ & v_j \text{ negative in } c_i \text{ and } v_j \text{ true in } m_k \\ \text{'-'}, & v_j \text{ not in } c_i. \end{cases}$$

The description length $\mathcal{L}(\mathcal{F}^{\mathcal{E}})$ is 0 for every explanation as the two possible functions *id* and *boolean negation* have no parameters that need to be instantiated. Consequently, for $\alpha > 0$, the costs of explanations for $\mathcal{I}$ are solely determined by $|\mathcal{T}^{\mathcal{E}+}|$.

Note that any explanation $\mathcal{E}$ can only produce exactly one target record from the source record of clause c_i because of the functionality of $\mathcal{F}^{\mathcal{E}}$. Because of attribute #, this target record needs to belong to a model of c_i, independent of the choice of $f_\#^{\mathcal{E}}$. For the same reason, a target record describing a model of clause c_i can only be produced by the source record corresponding to clause c_i. This means that for a fixed clause c_i, it is impossible to produce more than one target record that describes a model of c_i. Hence, each clause c_i for which $\mathcal{E}$ produces a target record from the corresponding source record reduces $|\mathcal{T}^{\mathcal{E}+}|$ by 1. An optimal solution is one that fulfills this criterion on the most clauses.

Lastly, note that $\mathcal{F}^{\mathcal{E}}$ describes an interpretation over the variables in $\mathcal{V}$. A variable $v_i \in \mathcal{V}$ in this interpretation is *true* if $f_{v_i}^{\mathcal{E}} = id$ and *false* if $f_{v_i}^{\mathcal{E}} = boolean\ negation$. Applying $\mathcal{F}^{\mathcal{E}}$ to the source record of clause c_i produces a record that contains the truth values of all variables occuring in c_i under this interpretation. If the resulting record is a target record, the interpretation satisfies the clause as target records of c_i describe models. We can conclude that if the 3-SAT instance c is satisfiable, an optimal solution $\mathcal{E}_0^*$ for the corresponding Explain-Table-Delta problem is able to produce a target record for every single clause by letting $\mathcal{F}^{\mathcal{E}_0^*}$ describe a model of c. Therefore, given the optimal solution $\mathcal{E}_0^*$, $|\mathcal{S}^{\mathcal{E}_0^*-}| = 0$ can be used to check if c is satisfiable and if it is, a model can be extracted from $\mathcal{F}^{\mathcal{E}_0^*}$. ◻

4 AFFIDAVIT

In this section, we describe the components of the search algorithm presented as Algorithm 1 to solve practical instances of Explain-Table-Delta. We implement it in a prototype that is called Affidavit[1] (Algorithm For Function-Inducing Delta Analysis Via Integration of Tables). For a given problem instance, it produces an explanation that serves as an affidavit to declare that, to the best of its knowledge, the specified modifications were used to generate the target from the source table.

Thanks to Proposition 3.6, the task of finding explanations can be reduced to a search for attribute functions. Consequently, Explain-Table-Delta can be understood as a constraint satisfaction problem. If the set of possible attribute functions is finite, a brute-force solution could enumerate and assess all possible attribute function tuples by treating each attribute as a variable with domain $\mathcal{F}$. Clearly, this approach does not scale and works poorly in practice because meta functions like value mappings cause $\mathcal{F}$ to grow exponentially with the size of the data.

We propose a best-first search instead, which starts from a set of empty or partial function assignments and efficiently navigates towards a full assignment with good quality. A transition in the search space corresponds to deciding on the function of an attribute. Each of the assignments of a search state acts as a constraint on the possible alignment of source and target records. The more attribute functions have been assigned, the more it becomes clear which records belong together under these assumptions, making it easier to decide on functions for the remaining attributes. A bad function choice eventually leads to high costs because it results in a small core or complicated functions for the remaining attributes. The search is guided by estimations of the final explanation costs of partial function assignments.

4.1 Preliminaries

Definition 4.1. **(Search Space)** Given $\mathcal{I}$ with $|\mathcal{A}| = d$, the *search space* is defined by $\mathcal{H}^{\mathcal{I}} := \{(h_1, ..., h_d) \mid h_i \in \{*, \diamond\} \cup \mathcal{F}\}$.

This means, a search state $\mathcal{H} \in \mathcal{H}^{\mathcal{I}}$ is a d-tuple whose component h_i assigns either $*$, $\diamond$ or some function f from $\mathcal{F}$ to attribute a_i. In the case of $*$, the function of a_i is still undecided. A $\diamond$ means that Affidavit has identified the attribute as one for which a value mapping is best suited. It will be resolved at the very end of the search when the alignment is maximally determined.

Definition 4.2. **(End State)** $\mathcal{H}$ is called *end state* if the function of each attribute is determined, i.e. if $\{a_i \in \mathcal{A} \mid h_i \in \{*, \diamond\}\} = \emptyset$.

[1] https://github.com/Finkman7/affidavit

Algorithm 1 Affidavit

```
function Affidavit($\mathcal{I}$)
    $Q \leftarrow$ Init-Start-States($\mathcal{I}$)                ▷ Init Priority Queue $Q$
    while $Q \neq \emptyset$ do
        $\mathcal{H} \leftarrow$ Poll($Q$)                          ▷ Remove Best State
        if Is-End-State($\mathcal{H}$) then break
        else
            $Q \leftarrow Q \cup$ Extensions($\mathcal{H}$)
    return Convert-To-Explanation($\mathcal{H}$)   ▷ Proposition 3.6

function Extensions($\mathcal{H}$)
    $\mathcal{A}^* \leftarrow$ Order-By-Indeterminacy($\{a_i \in \mathcal{A} \mid h_i = *\}$)
    $\mathcal{H}^{ext} \leftarrow \emptyset, \mathcal{A}^\diamond \leftarrow \emptyset$            ▷ Extensions and $\diamond$-attributes
    $\mathcal{A}' \leftarrow$ Poll($\mathcal{A}^*, \beta$)                       ▷ Poll $\beta$ attributes
    $\mathcal{R} \leftarrow$ Sample-Random-Alignment($\Phi^{\mathcal{H}}$)
    while $\mathcal{H}^{ext} = \emptyset$ and $\mathcal{A}' \neq \emptyset$ do
        for $a$ in $\mathcal{A}'$ do
            $\mathcal{H}^a \leftarrow \emptyset$            ▷ Promising attribute extensions
            $g \leftarrow$ Induce-Greedy-Map($\mathcal{R}$, $a$)
            $\mathcal{H}_g \leftarrow$ Extend($\mathcal{H}$, $a$, $g$)
            for $f$ in Induce-Functions($\Phi^{\mathcal{H}}$, $a$) do
                $\mathcal{H}_f \leftarrow$ Extend($\mathcal{H}$, $a$, $f$)
                if $c(\mathcal{H}_f) < c(\mathcal{H}_g)$ then
                    $\mathcal{H}^a \leftarrow \mathcal{H}^a \cup \{\mathcal{H}_f\}$
            if $\mathcal{H}^a \neq \emptyset$ then
                $\mathcal{H}^{ext} \leftarrow \mathcal{H}^{ext} \cup \mathcal{H}^a$
            else                  ▷ a map function is best suited for $a$
                $\mathcal{A}^\diamond \leftarrow \mathcal{A}^\diamond \cup \{a\}$
        $\mathcal{A}' \leftarrow$ Poll($\mathcal{A}^*$)                          ▷ Poll next attribute
    if $\mathcal{H}^{ext} = \emptyset$ then
        for $a$ in $\mathcal{A}^\diamond$ do                                  ▷ $\mathcal{A}^\diamond = \mathcal{A}^*$
            $\mathcal{H} \leftarrow$ Extend($\mathcal{H}$, $a$, $\diamond$)
        $\mathcal{H}^{ext} \leftarrow$ Finalize($\mathcal{H}$)                         ▷ Resolve $\diamond$s
    return $\mathcal{H}^{ext}$
```

Given a search state $\mathcal{H}$, its function assignments can be used as criteria for standard blocking [10] to group source and target records together.

Definition 4.3. **(Blocking Index)** The *blocking index* of a source or target record r under a search state $\mathcal{H}$ is determined by the projection $\xi^{\mathcal{H}}$ to those attributes whose functions are already determined. In the case of source records, the attribute functions are applied during projection:

$$\xi^{\mathcal{H}} := r \mapsto \begin{cases} \Pi_{\{h_i(a_i) \mid h_i \notin \{*,\diamond\}\}} & (r) \quad \text{if } r \in \mathcal{S} \\ \Pi_{\{a_i \mid h_i \notin \{*,\diamond\}\}} & (r) \quad \text{if } r \in \mathcal{T}. \end{cases}$$

$\Xi^{\mathcal{H}}$ denotes the set of all blocking indices:

$$\Xi^{\mathcal{H}} := \{\xi^{\mathcal{H}}(s) \mid s \in \mathcal{S}\} \cup \{\xi^{\mathcal{H}}(t) \mid t \in \mathcal{T}\}.$$

To address source records, target records and the block belonging to an index κ, we define:

$$\phi^{\mathcal{H}}_{\mathcal{S}} := \kappa \mapsto \{s \in \mathcal{S} \mid \xi^{\mathcal{H}}(s) = \kappa\}$$
$$\phi^{\mathcal{H}}_{\mathcal{T}} := \kappa \mapsto \{t \in \mathcal{T} \mid \xi^{\mathcal{H}}(t) = \kappa\}$$
$$\phi^{\mathcal{H}} := \kappa \mapsto \left(\phi^{\mathcal{H}}_{\mathcal{S}}(\kappa), \phi^{\mathcal{H}}_{\mathcal{T}}(\kappa)\right).$$

Definition 4.4. **(Blocking Result)** The *blocking result* under search state $\mathcal{H}$ is the set of all blocks $\Phi^{\mathcal{H}} := \{\phi^{\mathcal{H}}(\kappa) \mid \kappa \in \Xi^{\mathcal{H}}\}$.

Figure 3 visualizes parts of an example blocking result.

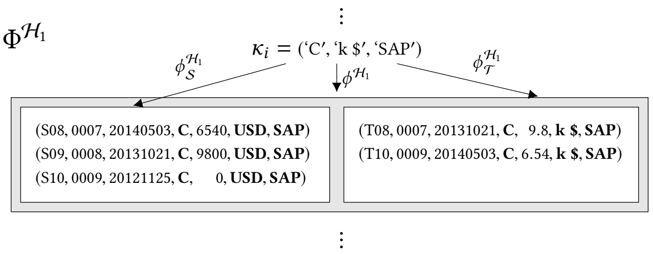
$\Phi^{\mathcal{H}_1}$
$\kappa_i = (\text{‘C'}, \text{‘k \$'}, \text{‘SAP'})$
$\phi^{\mathcal{H}_1}_{\mathcal{S}}$
$\phi^{\mathcal{H}_1}$
$\phi^{\mathcal{H}_1}_{\mathcal{T}}$
(S08, 0007, 20140503, **C**, 6540, **USD**, **SAP**)
(S09, 0008, 20131021, **C**, 9800, **USD**, **SAP**)
(S10, 0009, 20121125, **C**, 0, **USD**, **SAP**)
(T08, 0007, 20131021, **C**, 9.8, **k $**, **SAP**)
(T10, 0009, 20140503, **C**, 6.54, **k $**, **SAP**)

Figure 3: A block with index κ_i within blocking result $\Phi^{\mathcal{H}_1}$ of search state $\mathcal{H}_1 := (*, *, *, id, *, x \to \text{‘}k\ \$\text{'}, id)$ on $\mathcal{I}_1$.

4.2 Initialization Strategy

A natural choice for the set of start states $\mathcal{H}^0$ of the search is $\mathcal{H}^\emptyset = \{(*, \ldots, *)\}$ in which all assignments are empty. While it begins the search without any wrong assumptions and can in theory lead to any explanation, it comes with a drawback. Without any assumption on how to link the records, producing input-output examples to learn the first function is very difficult.

A second natural way of beginning the search comes to mind which dampens this issue. Given the assumption that there is at least one attribute which has not been changed, the search can be started from the set

$$\mathcal{H}^{id} := \{(id, \ldots, *), (*, id, *, \ldots, *), \ldots, (*, \ldots, id)\}.$$

We prefer it to $\mathcal{H}^\emptyset$ as this assumption should be valid for nearly all practical use cases.

Furthermore, we find overlap scores another reasonable choice to determine a start state. It can drastically improve the runtime by beginning the search from a state in which most of the attributes have already been assigned. The idea is to independently assume for each attribute that it has not been changed and use it to link source and target records that have the same value on this attribute. Giving a score of 1 per attribute on which two records are identical, the score of each pair denotes their similarity in terms of an attribute overlap. Assuming that k unchanged attributes exist, the score of correctly aligned record pairs will be at least k and it is very likely that among the pairs with the highest overlap score, their large overlap will stem mainly from these attributes. We take advantage of this by using the target record with the highest overlap for each source record to build the most likely a-priory alignment over all source records. Sorting the attributes by how often their values overlap on these pairs, we use the k' most frequently overlapping ones to build a set $\mathcal{A}^{id}$. Our choice of k' is determined by the most frequent overlap score among these pairs. This leads to the set of start states

$$\mathcal{H}^s := \{(h_1, \ldots, h_d)\} \text{ with } h_i = \begin{cases} id & \text{if } a_i \in \mathcal{A}^{id} \\ * & \text{otherwise.} \end{cases}$$

To compute record overlaps without a quadratic comparison of all records, we calculate it only for record pairs that share at least one value. Very frequent attribute values that are shared by nearly every record generate an enormous amount of alignment pairs. Therefore, we ignore value overlaps in which the number of resulting pairs would be above a configurable threshold. This limits the a-priori matching to pairs that share at least one value that is not too frequent.

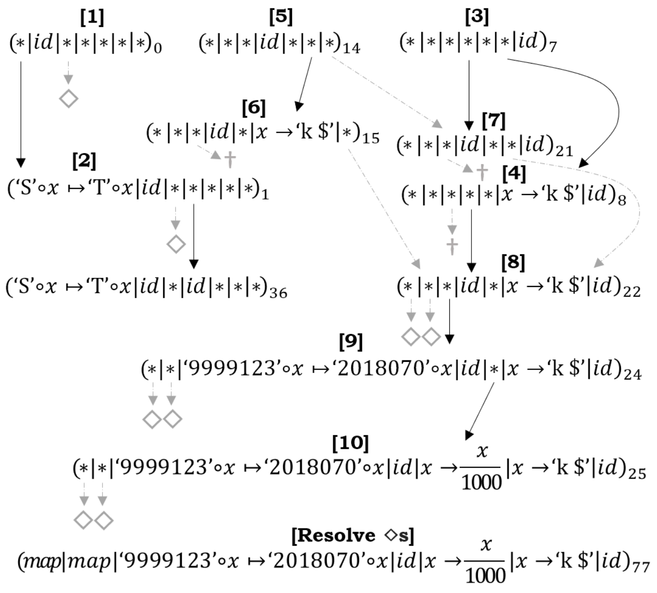
[1]
(∗|id|∗|∗|∗|∗|∗)0
[5]
(∗|∗|∗|id|∗|∗|∗)14
[3]
(∗|∗|∗|∗|∗|∗|id)7
[6]
(∗|∗|∗|id|∗|x →'k $'|∗)15
[7]
(∗|∗|∗|id|∗|∗|id)21
[2]
('S'∘x ↦'T'∘x|id|∗|∗|∗|∗|∗)1
[4]
(∗|∗|∗|∗|∗|x →'k $'|id)8
('S'∘x ↦'T'∘x|id|∗|id|∗|∗|∗)36
[8]
(∗|∗|∗|id|∗|x →'k $'|id)22
[9]
(∗|∗|'9999123'∘x ↦'2018070'∘x|id|∗|x →'k $'|id)24
[10]
(∗|∗|'9999123'∘x ↦'2018070'∘x|id|x → x/1000|x →'k $'|id)25
[Resolve ◇s]
(map|map|'9999123'∘x ↦'2018070'∘x|id|x → x/1000|x →'k $'|id)77

Figure 4: Search Tree on $\mathcal{I}_1$ with $\alpha = 0.5, \beta = 2, \varrho = 3$.

4.3 Extending Search States

Affidavit discovers potential attribute functions from the blocking result $\Phi^{\mathcal{H}}$ of a search state $\mathcal{H}$ as described in Section 4.4. The resulting functions are used to extend $\mathcal{H}$ by assigning an induced function to the corresponding attribute.

To extend a state $\mathcal{H}$, we first decide on a set of attributes $\mathcal{A}'$ for which functions are induced. The decision is based on an estimation of the indeterminacy of the undecided attributes of $\mathcal{H}$. We estimate it for an attribute by computing its maximum number of distinct source values over all blocks that have both target and source records. This corresponds to an upper bound for the number of source values that need to be considered as the origin of a target value. $\mathcal{A}'$ consists of the β most determined attributes from this estimation. In the next step, we create extensions of $\mathcal{H}$ with the β most promising function candidates of each attribute in $\mathcal{A}'$. The branching factor β is configurable to limit the number of extensions that are produced.

For each $a \in \mathcal{A}'$, we compare its resulting extensions to $\mathcal{H}_g$ which is an extension of $\mathcal{H}$ on a with a map function constructed from a random alignment of all records that respects $\Phi^{\mathcal{H}}$. We build it by mapping each source value to the target value with the highest co-occurrence in the random record alignment. If $\mathcal{H}_g$ has the lowest costs, we store a in the set $\mathcal{A}^\diamond$ and reject the attribute's extensions, otherwise we keep all extensions with costs lower than those of $\mathcal{H}_g$. If the function of a in the optimal solution is a value mapping, there typically is no other function type that can be instantiated to a good function candidate for that attribute. This is why, once their indeterminacy is low, this check allows the detection of attributes for which map functions are likely needed.

If we did not keep any extension from the β most determined attributes, we try the next most determined one until we have found at least one extension or we have come to the conclusion that all undecided attributes should receive a map function. In the latter case, we mark all attributes from $\mathcal{A}^\diamond$ with $\diamond$ and finalize the state by replacing one $\diamond$ after another. The replacement is a greedy value mapping from the random alignment, just like $\mathcal{H}_g$. We re-sample a new random alignment after each $\diamond$ is replaced in order to have the next map respect the previous assignment.

Figure 4 demonstrates how Affidavit finds the optimal solution on $\mathcal{I}_1$ starting from $\mathcal{H}^0 = \mathcal{H}^{id}$. The number in square brackets indicates the order in which the states are expanded. $\mathcal{A}'$ is implicitly given by the origin of the arrows. Greyed out arrows lead to extensions that are not added to the queue because no induced function was better than a greedy map ($\diamond$), the queue was full with better states (†, see Section 4.6) or because of duplicate detection.

4.4 Inducing Functions

4.4.1 Supported Meta Functions. Our framework supports any meta function whose parameters are learnable from one input-output example. An example for such a meta function is the conversion of a date attribute. An input-output example such as *'Sep 31 2019'* ↦ *'20190931'* contains enough information to learn to split the source value by white spaces to extract month, day and year (in that order) and express the date in '*yyyymmdd*' format. Note that there can still be input-output examples of that function such as *'Oct 10 2019'* ↦ *'20191010'* for which the function instantiation is not unambiguous. It would not be clear from this example if the target format is '*yyyymmdd*' or '*yyyyddmm*'. However, one could simply generate both candidate functions or learn the function from a different input-output example. On the other hand, any linear function of the form $x \mapsto mx + c$ needs at least two input-output examples to learn its parameters m and c. After one example, the number of possible meta function instantiations is still infinitely large. There is no single example that would be enough to induce the function and it is impossible too, to generate all possible candidate functions from one example.

To transform values of an attribute that was the target of a function type that is not supported, Affidavit tries to learn the full value mapping. This way it can still produce explanations with a correct record alignment, even if a more concise function can not be learned. As a mapping with more than one entry needs an input-output example for every value it transforms, value mappings are not learned during the search like other functions. Instead, they are learned at the very end when the record alignment is maximally defined by regular functions.

4.4.2 Inducing Function Candidates. To induce functions for an attribute, Affidavit uses noisy input-output examples that it samples from blocks that have both source and target records inside them. It does this by randomly selecting up to k distinct target records from these blocks and trying for each one to produce its attribute value from the value of any source record in the same block. We do so by instantiating functions from the meta functions. For example, if target record *T08* from Figure 3 was sampled to learn functions for Val, the following functions could be induced: $x \mapsto x - 6530.2$ (from *'6540'*), $x \mapsto \frac{x}{1000}$ (from *'9800'*), $x \mapsto x + 9.8$ (from *'0'*), $x \mapsto$ *'9.8'* (from any source value).

The set of induced functions over the sampled target records is filtered to include only functions that have been generated a statistically significant amount of time. The idea behind this, is that the function of the optimal solution would be generated each time we sample a target record from the core image of the optimal solution. However, it is only generated from examples in which the effect of the optimal function is actually visible. How often it gets generated, depends on the fraction θ of records with this property in relation to the number of target records. For example, the optimal function might be the one that removes trailing zeroes if there are any but it would not be generated from correct examples without trailing zeroes in the source value.

We regard the sampling of a target record as Bernoulli experiment with success chance θ. Assuming $|\mathcal{T}| >> k$, we treat each experiment as independent, such that the number of records X from which the best function would be generated is a random variable that follows a Binomial distribution with success chance θ and sample size k. We set k to the smallest integer for which $p(X \geq 5) \geq \rho$. Both the estimated fraction θ and the confidence level ρ are configurable parameters. Choosing a larger θ speeds up the algorithm but risks that functions of the optimal solution will not be sampled if θ underestimates the amount of noise in the target records or the rarity of the function's effect. A function that was generated n times, is filtered if $p(X = n) < \rho$. If θ is set lower than the actual fraction, the function of the optimal solution will be found with a probability of a least ρ.

4.4.3 Ranking Function Candidates. In the next step, we determine the best β candidate functions for each attribute. This time, the fact that some functions are not induced from all value pairs which they cover, prevents us from simply ranking the candidates by how often they were generated. While the function from the optimal solution is very likely to be contained in the candidate set after filtering, it is not necessarily the one that was generated most often.

A complete evaluation would consist in traversing all blocks and applying every function candidate to the block's source record values in order to compare the resulting histogram with the block's target values. As this can be very expensive, we use sampling to estimate the fraction of records that each function would align. This time, we sample k' distinct source records and to penalize functions that map too many source values to the same target value, we evaluate on the level of blocks that contain them instead of the individual records. In each block of a sampled source record, we apply all function candidates to every source record value of the block to create a value histogram each in which every resulting value has a frequency equal to the sum of the frequencies of all source values from which it was created. For example, $x \mapsto \frac{x}{1000}$ on block κ_i from Figure 3 results in the histogram $\{1 \times$ *'6.54'*$, 1 \times$ *'9.8'*$, 1 \times$ *'0'*$\}$, while $x \mapsto$ *'9.8'* produces $\{3 \times$ *'9.8'*$\}$. For each function candidate of an attribute, we compute the overlap of the resulting histogram and the target value histogram of the block ($\{1 \times$ *'9.8'*$, 1 \times$ *'6.54'*$\}$ for κ_i) by summing up the minimum of the frequency of each value present in both histograms. On block κ_i, $x \mapsto \frac{x}{1000}$ would have an overlap of 2, whereas $x \mapsto$ *'9.8'* has an overlap of 1. We rank the candidate functions in descending order by the size of their total overlap minus their description length to determine the best β candidates.

We choose the smallest integer k' for which it holds that if we use $p = \theta$ in Cochran's formula [6] for determining sample sizes:

$$k' \geq \frac{z^2 \cdot p \cdot (1-p)}{e^2}.$$

For this, we choose $z = 1.96$ and $e = 0.05$ which yields a confidence of 95% that the overlap on the sampled blocks is within ±5% of the overlap over all blocks. If $\Phi^{\mathcal{H}}$ is already very fine-grained with many blocks, this results in a huge speed-up over an evaluation over all blocks. If $\Phi^{\mathcal{H}}$ is still very vague with few blocks, we usually evaluate on many more source records than we sample because we fully evaluate the blocks in which they are contained. This makes the sampling actually less risky than the guarantees of the formula imply.

4.5 Evaluating Search States

The cost function from Definition 3.10 is defined for explanations which can only be constructed from end states. However, during the search, it is necessary to assess the quality of partial search states. In this section, we describe how we extend this definition to partial search states (and end states) in a coherent way to arrive at the cost function used by AFFIDAVIT.

The cost component $\mathcal{L}(\mathcal{F}^{\mathcal{E}})$ that measures the description length of an explanation's attribute functions has an obvious counterpart for search states:

$$c_f(\mathcal{H}) := \sum_{h_i, h_i \notin \{*, \diamond\}} \psi(h_i).$$

The value of $\mathcal{L}(\mathcal{T}^{\mathcal{E}+})$ however, depends on $|\mathcal{T}^{\mathcal{E}+}|$ which is not yet determined by a partial search state $\mathcal{H}$. A lower bound is given by the record set for which it is already clear from the partial function assignments of $\mathcal{H}$ that no source record will be aligned with it in any end state to which this search state can lead. Any record in a block without source records is such a record.

On the other hand, the blocks that do have source records can still be used to improve this lower bound. Because a valid explanation's attribute function tuple is a bijection, the number of those records can be estimated from the blocking result of $\mathcal{H}$ from the blocks which have more target than source records:

$$c_t(\mathcal{H}) := \sum_{\kappa \in \Xi^{\mathcal{H}} | \ |\phi_{\mathcal{T}}^{\mathcal{H}}(\kappa)| > |\phi_{\mathcal{S}}^{\mathcal{H}}(\kappa)|} |\phi_{\mathcal{T}}^{\mathcal{H}}(\kappa)| - |\phi_{\mathcal{S}}^{\mathcal{H}}(\kappa)|.$$

Moreover, there is an alternative way of computing $|\mathcal{T}^{\mathcal{E}+}|$ that can be useful to estimate costs during search.

COROLLARY 4.5. *Let* $\Delta = |\mathcal{S}| - |\mathcal{T}|$. *The validity properties can be leveraged to compute* $|\mathcal{T}^{\mathcal{E}+}|$ *in terms of* $|\mathcal{S}^{\mathcal{E}-}|$ *and* Δ.

PROOF. $|\mathcal{T}^{\mathcal{E}+}| \overset{(Def.\ 3.5)}{=} |\mathcal{T}| - |\mathcal{T}^{\mathcal{E}}| = (|\mathcal{T}| + \Delta) - \Delta - |\mathcal{T}^{\mathcal{E}}|$
$= |\mathcal{S}| - \Delta - |\mathcal{T}^{\mathcal{E}}| \overset{(Def.\ 3.5)}{=} |\mathcal{S}| - \Delta - |\mathcal{S}^{\mathcal{E}}|$
$= (|\mathcal{S}| - |\mathcal{S}^{\mathcal{E}}|) - \Delta \overset{(Def.\ 3.3)}{=} |\mathcal{S}^{\mathcal{E}-}| - \Delta$ □

Just like $|\mathcal{T}^{\mathcal{E}+}|$, $|\mathcal{S}^{\mathcal{E}-}|$ cannot be completely calculated for a partial search state $\mathcal{H}$. However, as before we can compute a lower bound by using the blocking result of $\mathcal{H}$:

$$c_s(\mathcal{H}) := \sum_{\kappa \in \Xi^{\mathcal{H}} | \ |\phi_{\mathcal{S}}^{\mathcal{H}}(\kappa)| > |\phi_{\mathcal{T}}^{\mathcal{H}}(\kappa)|} |\phi_{\mathcal{S}}^{\mathcal{H}}(\kappa)| - |\phi_{\mathcal{T}}^{\mathcal{H}}(\kappa)|$$

Definition 4.6. **Costs of Search States** The cost of a search state $\mathcal{H}$ is defined by

$$c(\mathcal{H}) := 2\alpha \cdot c_f(\mathcal{H}) + 2(\alpha - 1) \cdot \max(c_t(\mathcal{H}), c_s(\mathcal{H}) - \Delta).$$

It depends on both the problem instance and the search state which lower bound underestimates less.

4.6 Queue

The best-first search tends to spend most of its time visiting search states with few assignments which is most pronounced when starting the search from $\mathcal{H}_\emptyset$ or $\mathcal{H}_{id}$. This stems from the fact that costs monotonically increase when a function is added to an undecided attribute. What makes this behavior problematic, is the fact that there are exponentially many search states from which one can reach an end state. For instance, there are $2^{|\mathcal{A}|}$ states in the search lattice on a path from $\mathcal{H}_\emptyset = (*, \dots, *)$ to the end state $\mathcal{H}_{id} = (id, \dots, id)$. Even with duplicate elimination, unless $\mathcal{H}_{id}$ aligns all records, a complete search would possibly try exponentially many subsets of *id* assignments to check if the remaining attributes can have other functions assigned that result in a better explanation. This behavior cripples performance as the number of attributes grows.

Fortunately, apart from (direct as well as indirect) parents of the optimal solution, it is in practice very unlikely to find many different search states that are at least as good as the optimal solution and will therefore be extracted before it. The likelihood of finding a search state with this property strongly decreases with the number of assignments of a state. For instance, setting *ID1* to *id* in $\mathcal{I}_1$ is an assignment that makes the state $(*, *, id, *, *, *, *)$ look promising at first and even still after extending it to $(*, *, id, *, *, x \mapsto \text{`}k\ \$\text{'}, *)$. These states align a lot of records with relatively cheap functions. This is why they are extracted first in Figure 4. However, the costs of states that result in an incorrect record alignment eventually increases fast when assigning functions to the remaining attributes. In this case, costly value mappings are neededÃű on attributes that could be transformed with a simple operation under the correct alignment. In addition, if the same source value is aligned with multiple different target values (which is more likely under a wrong alignment), the number of aligned records will drop even when using value mappings. For an increasing number of assignments, this makes it more and more unlikely to find states with *ID1* set to *id* that have lower costs than the optimal solution.

Therefore, we decide to use a modified priority queue that is bounded to hold $max(1, \varrho - i + 1)$ search states at the same time on the i-th level of the lattice, i.e. the level on which states have i attributes assigned. If a level is full, it only accepts a new state if it is not worse than all states on the same level. If an inserted state is accepted, it drops the worst state on the same level to make space. Polling the queue still returns the state with the lowest costs independent of the level. In case of equal costs, it returns states with a higher number of assignments first. Heuristically, it is quite unlikely in practice to skip the optimal solution due to this limitation. The most important decisions of the search happen at the early levels. While the cost of search states with only one or a few assignments might still be underestimated, a handful of assignments is in practical problem instances usually enough to identify the best foundation for inducing the remaining attribute functions.

5 EVALUATION

To evaluate Affidavit, we have implemented the meta functions described in Table 1 which include basic string and number transformations. The experiments are meant to demonstrate the core functionality and scalability of the framework. In practice, one might encounter problem instances with functions not supported by our prototype. However, we decided to evaluate on self-created problem instances based on these meta functions because this gives us certainty about the correct transformations and alignment as well as control over the degree of change and noise. This way, we can judge the given explanations in-depth. Furthermore, we can evaluate on the same table multiple times with different transformations, giving a more trust-worthy result. We describe our synthetic transformation of real-world datasets in Section 5.1 and the evaluation protocol in Section 5.2. In Section 5.3, we report about the quality of the produced explanations. Finally, we evaluate in Section 5.4 how our algorithm scales with the number of records and attributes of a problem instance.

Name	Operation	Parameters
Identity	$x \mapsto x$	–
Uppercasing	$x \mapsto \text{Uppercase}(x)$	–
Constant Value	$x \mapsto c$	c
Addition (Numeric)	$x \mapsto x + y$	y
Division (Numeric)	$x \mapsto x/y$	y
Front Masking	$.\{\lvert m\rvert\} \circ x \mapsto m \circ x$	m
Front Char Trimming	$[c]^* \circ x \mapsto x$	c
Prefixing	$x \mapsto y \circ x$	y
Prefix Replacement	$y \circ x \mapsto z \circ x$	y, z
Value Mappings	$x \mapsto \begin{cases} y_1 & \text{if } x = x_1 \\ \dots & \\ y_n & \text{if } x = x_n \end{cases}$	$x_1, \dots, x_n,$ $y_1, \dots, y_n$

Table 1: Meta functions implemented in Affidavit. The inverse variants of these functions are also supported, e.g. suffixing in addition to prefixing. String concatenation is denoted by $\circ$.

5.1 Datasets

We perform our experiments on the datasets[2] described in more detail in [22] which have already been used to evaluate algorithms for detecting functional dependencies. They cover a wide range of topics (e.g., flight routes, chess game logs, web log data, etc.) and feature different structural properties, both in terms of the number of attributes (5 to 223) and records (100 to 250000).

For each dataset used in [22], we create ten problem instances in three settings of varying difficulty. Each problem instance is the result of choosing some records of the table as core, transforming it with randomly sampled functions and using the remaining records as noise for the source and target snapshots. A setting consists of two parameters τ and η. The transformation percentage τ denotes the likelihood to sample a function different from *id* for an attribute. This means, it can happen that every attribute gets transformed. In this case, we reject the sampling and generate another one. To sample a function for an attribute that is to be transformed, we randomly instantiate a function from the meta functions described in Table 1. We make sure to generate functions that fit the domain of the attribute, e.g. we do not use uppercasing on numerical attributes. In the case of value mappings, we instantiate it as a random permutation of the source values. These are potentially the hardest transformations to learn due to the high number of parameters and can easily lead to a wrong alignment when confused with *id*. The noise percentage η refers to the fraction of source and target records that are outside the core of the generated problem instance.

To create two table snaphots from a dataset, we first determine the source and target noise by randomly selecting two record subsets. We choose the size of the noise sets such that these records make up a fraction η each of the resulting snapshots. The number of records in the resulting snapshots decreases to a fraction $\frac{1}{\eta+1}$ of the dataset as the noise records are distributed over both snapshots. The rest of the dataset records resembles the core of our reference explanation. We create the core image by applying the sampled transformations to the corresponding attributes of these core records. We also apply the sampled transformations

[2]https://hpi.de/naumann/projects/repeatability/data-profiling/fds.html

to the target noise as its data format should be similar to the core image records. Finally, we add the the source and target noise to the core and core image, respectively.

In addition to the random attribute transformations, we augment each dataset with a new attribute that contains a set of running integers to simulate a simple primary key. We use the same integers in both snapshots in two different permutations. The resulting attribute results in a wrong record alignment if it is used for blocking and is supposed to challenge Affidavit's ability to deal with transformed primary key attributes. If the dataset already has attributes in which the fraction of distinct attribute values is larger than 0.7, we remove these attributes for our experiments as it might make the alignment too easy when that attribute is not transformed. In Table 2, $|\mathcal{A}|$ denotes the resulting number of attributes after these modifications. Dataset attributes that are completely empty prior to the transformations are ignored as well and do not count towards this number.

5.2 Evaluation Protocol

On each problem instance, we evaluate Affidavit with two different configurations on a unix system with 24 cores at 2.6 GHz and 200GB memory. The first configuration uses $\mathcal{H}^0 = \mathcal{H}^s$ (start states) determined with a maximum block size of 100000 for the overlap matching, $\beta = 1$ (branching factor) and $\varrho = 1$ (queue width). The second configuration uses $\mathcal{H}^0 = \mathcal{H}^{id}$, $\beta = 2$ and $\varrho = 5$. Both configurations were run with $\alpha = 0.5$, $\theta = 0.1$ (core size estimation) and $\rho = 0.95$ (confidence).

We have chosen two configurations of Affidavit that resemble different approaches of tackling the problem. Using overlap sampling to begin the search from a promising start state follows the spirit of unsupervised record linking and assumes that similarities in the data can be leveraged a-priori to align the records with sufficient accuracy for the induction of transformation functions. The intention of this approach is a reduction of runtime compared to a more exhaustive search. As such, we chose to push further into that direction by limiting both branching factor and queue width to see how well one can induce the remaining functions when starting from an a-priori alignment. The resulting configuration corresponds to a greedy search that induces one attribute after another without backtracking. We compare this configuration with a search that begins with the set of start states $\mathcal{H}^{id}$ in which each state corresponds to the assumption that one particular attribute was not changed. We use parameters that allow the search to traverse a larger part of the search lattice as this setting is supposed to be more robust in exchange for runtime.

Table 2 describes the macro average over the ten problem instances per dataset per setting with four numbers comprised of runtime t, relative core size Δ_{core}, relative costs Δ_{costs} and accuracy *acc*. The latter three numbers are computed by comparing the resulting explanation $\mathcal{E}_{res}$ to the reference explanation $\mathcal{E}_{ref}$ that correctly describes the attribute functions and separation of core and noise records that were used to create the problem instance. For example, $\Delta_{core} = 0.8$ means that $\mathcal{E}_{res}$ aligned 20% less records than $\mathcal{E}_{ref}$ and $\Delta_{costs} = 0.99$ says that the cost $c(\mathcal{E}_{res})$ was 1% lower (better) than $c(\mathcal{E}_{ref})$. Accuracy is calculated by applying the learned functions $\mathcal{F}^{\mathcal{E}_{res}}$ to each core record $r \in \mathcal{S}^{\mathcal{E}_{ref}}$ and comparing it with the correct transformation $\mathcal{F}^{\mathcal{E}_{ref}}(r)$. For computing accuracy, we ignore the artificial primary key attribute that we added and measure it as the fraction of cells in $\mathcal{S}^{\mathcal{E}_{ref}}$ that are correctly translated this way.

5.3 Result Quality

The results presented in Table 2 show that $\mathcal{H}^{id}$ performs very well in the setting ($\eta = 0.3, \tau = 0.3$) as it learned to correctly translate the core in every run, with minor deviations only in the *balance* and *nursery* datasets. This is a hint that Affidavit can be used with this setting out-of-the-box to produce high quality explanations for problem instances with a reasonable amount of noise and transformations. However, we see a definite decline in accuracy on some datasets in the setting ($\eta = 0.7, \tau = 0.7$). We attribute it mainly to the high noise, as we can see on *balance*, *chess* and *nursery* that Affidavit was able to produce explanations cheaper than the reference, aligning a larger number of records by including noise in the core. This shows that our search is effective at minimizing costs but that our cost definition does not reliably lead to the correct explanation when the majority of the records are noise. Consequently, this effect is more pronounced in tables with few attributes. Nevertheless, we see on some datasets that Affidavit can still correctly learn transformations when the majority of attributes has been changed. This holds especially for tables with a large number of attributes which supports our claim that at least a handful of unchanged attributes is needed to correctly bootstrap the alignment in the beginning of the search – but not necessarily more.

As expected, we find most of the runtimes of $\mathcal{H}^s$ to be significantly lower. The performance in terms of accuracy is mostly comparable to that of $\mathcal{H}^{id}$ which shows that our use of overlap sampling is a promising way to start the search. However, we can see obvious gaps in performance on datasets such as *chess*, *letter* or *nursery*. We manually investigated this and found out that $\mathcal{H}^s$ begins the search assuming that the artificial primary key attribute was unchanged. As highlighted by $\Delta_{core} = 0$, this leads to a trivial explanation because Affidavit was not able to find functions for the remaining attributes to support anything but an empty core. The reason for this behavior is the fact that these tables contain only attributes with very few distinct values in relation to the number of records. As such, the maximum block size is exceeded when producing records pairs based on overlapping attribute values. The exception is the maximally distinct attribute of running integers which leads to a wrong a-priori alignment. Increasing the maximum block size to the point where a correct start state is found, we found the initial matching already took longer than the total runtime of the more exhaustive configuration. This shows that there are problem instances for which the search from $\mathcal{H}^{id}$ is preferable because it is independent of record similarity and can correct wrong decisions by backtracking.

5.4 Scalability

Affidavit was designed with the goal of scaling to large problem instances. In the context of database tables, this means that the runtime should increase at most linearly with both a growing number of records and attributes.

5.4.1 *Row Scalability.* We begin by experimentally measuring the scalabilty of Affidavit to tables with a large number of records. For this, we run the $\mathcal{H}^{id}$ configuration on scaled versions of a ($\eta = 0.3, \tau = 0.3$) problem instance of *flight-500k* which comes from the same source as *flight-1k* but by default has 500000 records and 20 attributes. To scale the problem instance to $x\%$ of the original size, we use $x\%$ of the core records as well as $x\%$ of the source and target noise. We then create the corresponding core image from the scaled core. The sampled transformations

				$\eta = 0.3, \tau = 0.3$				$\eta = 0.5, \tau = 0.5$				$\eta = 0.7, \tau = 0.7$			
Dataset	$\|\mathcal{A}\|$	Records	$\mathcal{H}^0$	t	Δ_{core}	Δ_{costs}	acc	t	Δ_{core}	Δ_{costs}	acc	t	Δ_{core}	Δ_{costs}	acc
iris	6	150	$\mathcal{H}^s$	0.12s	1.01	1	1	0.09s	0.99	1.01	0.99	0.10s	1.04	0.99	0.99
			$\mathcal{H}^{id}$	0.69s	1.01	1	1	0.51s	1.02	0.99	1	0.38s	1.05	0.99	0.99
balance	6	625	$\mathcal{H}^s$	0.23s	1.01	0.99	0.99	0.21s	0.96	1.02	0.92	0.19s	1.42	0.9	0.84
			$\mathcal{H}^{id}$	0.82s	1.01	0.99	0.99	0.63s	0.93	1.03	0.9	0.79s	1.44	0.89	0.86
chess	8	28056	$\mathcal{H}^s$	2.83s	0	2.11	0.43	2.16s	0.24	1.46	0.56	2.00s	0.45	1.16	0.6
			$\mathcal{H}^{id}$	7.70s	1.03	0.96	1	6.37s	1.05	0.97	0.98	12.97s	1.24	0.93	0.86
abalone	9	4177	$\mathcal{H}^s$	1.49s	0.98	1.02	1	1.01s	0.98	1.01	1	0.88s	0.82	1.04	0.89
			$\mathcal{H}^{id}$	8.70s	1	1	1	3.44s	1	1	1	3.61s	0.97	1.01	1
nursery	10	12960	$\mathcal{H}^s$	1.58s	0	2.27	0.51	1.36s	0.16	1.56	0.56	1.41s	0	1.32	0.48
			$\mathcal{H}^{id}$	4.24s	1	1.01	0.98	5.26s	0.96	1.03	0.85	4.63s	1.55	0.83	0.87
bridges	10	108	$\mathcal{H}^s$	0.05s	0.99	1.02	1	0.08s	0.96	1.04	0.99	0.08s	1.05	1.11	0.9
			$\mathcal{H}^{id}$	0.43s	1	1	1	0.50s	1	1.01	0.99	0.69s	1.15	1.04	0.96
echo	10	132	$\mathcal{H}^s$	0.07s	0.99	1.02	1	0.13s	0.93	1.06	0.98	0.11s	0.89	1.13	0.93
			$\mathcal{H}^{id}$	0.79s	0.99	1.02	1	0.89s	0.93	1.04	0.99	0.95s	0.87	1.11	0.94
breast	11	699	$\mathcal{H}^s$	0.39s	1.07	0.91	1	0.42s	1.21	0.85	0.99	0.42s	1.49	0.83	0.98
			$\mathcal{H}^{id}$	1.02s	1.1	0.86	1	1.08s	1.26	0.81	1	1.37s	1.6	0.8	0.99
adult	15	48842	$\mathcal{H}^s$	6.42s	0.96	1.06	1	5.57s	0.97	1.05	0.99	4.17s	0.99	1.03	0.97
			$\mathcal{H}^{id}$	14.33s	1	1.01	1	19.91s	0.93	1.1	0.99	17.38s	1.1	0.99	0.98
ncvoter-1k	16	1000	$\mathcal{H}^s$	0.58s	0.95	1.08	1	0.57s	0.99	1.01	1	0.85s	0.88	1.06	0.97
			$\mathcal{H}^{id}$	1.81s	0.99	1.02	1	2.33s	0.98	1.01	1	3.50s	0.87	1.07	0.96
letter	18	20000	$\mathcal{H}^s$	4.41s	0	2.65	0.86	5.04s	0.31	1.55	0.82	5.59s	0.68	1.12	0.79
			$\mathcal{H}^{id}$	12.73s	1.02	0.97	1	10.78s	1.04	0.97	1	9.40s	1.14	0.95	1
hepatitis	19	155	$\mathcal{H}^s$	0.11s	0.95	1.09	1	0.14s	0.97	1.02	1	0.19s	0.83	1.09	0.98
			$\mathcal{H}^{id}$	0.79s	0.94	1.1	1	0.71s	0.96	1.03	1	0.76s	0.82	1.09	0.97
horse	28	368	$\mathcal{H}^s$	0.23s	0.99	1.01	1	0.38s	0.89	1.09	0.99	0.56s	0.99	1.01	1
			$\mathcal{H}^{id}$	1.19s	0.97	1.06	1	1.36s	0.94	1.05	0.99	1.82s	0.82	1.07	0.98
fd-red-30	31	250000	$\mathcal{H}^s$	261.18s	1.03	1.06	1	190.49s	0.96	1.04	1	132.03s	0.98	1.01	1
			$\mathcal{H}^{id}$	281.46s	1	1	1	342.02s	1	1	1	242.51s	1	1	1
plista	43	1000	$\mathcal{H}^s$	1.70s	0.9	1.2	1	2.35s	0.89	1.1	0.99	2.52s	1.06	0.98	1
			$\mathcal{H}^{id}$	4.34s	0.98	1.05	1	6.74s	1.01	0.99	1	8.28s	0.93	1.03	0.99
flight-1k	75	1000	$\mathcal{H}^s$	2.67s	0.81	1.41	0.99	3.85s	0.68	1.3	0.98	4.82s	0.69	1.13	0.98
			$\mathcal{H}^{id}$	14.98s	1	1.01	1	26.58s	0.95	1.05	1	35.89s	0.9	1.05	0.99
uniprot	182	1000	$\mathcal{H}^s$	2.95s	0.45	2.23	0.99	2.80s	0.33	1.65	0.99	3.96s	0.77	1.1	1
			$\mathcal{H}^{id}$	49.52s	1	1.01	1	40.55s	1	1.01	1	33.70s	0.85	1.08	1

Table 2: Experimental results of two AFFIDAVIT configurations $\mathcal{H}^s$ and $\mathcal{H}^{id}$ on problem instances of varying difficulty.

stay the same. However, we remove value mapping entries defined over attribute values that do not exist anymore in the scaled version. Otherwise the costs of the reference explanation would be unnecessarily high. The resulting run times in Figure 5 confirm that AFFIDAVIT scales linearly in the number of records. Moreover, it was able to produce the reference explanation in every run on these problem instances.

5.4.2 Attribute Scalability. Attribute scalability is difficult to assess experimentally because removing attributes from a problem instance can completely alter the difficulty. However, because of the modified priority queue, we can give a rough theoretical upper-bound in ϱ for the worst-case complexity that suggests linear scalability in the number of attributes. For a fixed ϱ, AFFIDAVIT begins the search with ϱ search states with one assignment each. Ignoring duplicate elimination, in the absolute worst-case, each of these search states and its (direct and indirect) children are visited in depth-first order. Assuming $\varrho < |\mathcal{A}|$, this results in visiting level ϱ with $O(\varrho!)$ states that are each followed by $|\mathcal{A}| - \varrho$ extensions to produce a full assignment which gives $|\mathcal{A}|O(\varrho!) - \varrho O(\varrho!)$ total extensions. In the case of $\mathcal{H}^0 = \mathcal{H}^{\emptyset}$, this number is at most one larger, for $\mathcal{H}^0 = \mathcal{H}^s$ it is smaller.

Technically, there are operations inside each extension that are not constant in the number of attributes, leading to a polynomial complexity. However, during the extension of a state, the runtime is dominated by the induction of functions for a fixed attribute. As the number of attributes for which this is performed is bounded by β, a linear runtime increase with a growing number of attributes should be the result in practice. The normalized runtimes in Figure 6 support this expectation. The resulting data is very noisy though for a low number of attributes which can be explained by the fact that individual differences in difficulty of the datasets have a proportionally bigger impact on the runtime than the difference in the already low number of attributes.

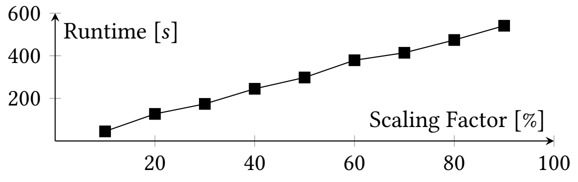

Figure 5: Runtimes on a ($\eta = 0.3, \tau = 0.3$) problem instance of *flight-500k* scaled to different numbers of records.

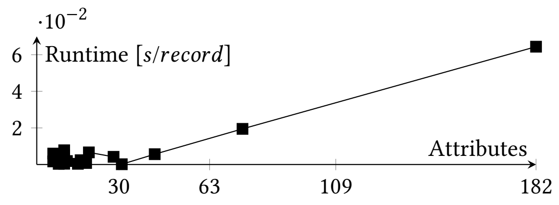

Figure 6: Runtimes of $\mathcal{H}^{id}$ taken from Table 2 for the setting ($\eta = 0.3, \tau = 0.3$) normalized by the number of records of each dataset in relation to the number of its attributes.

6 CONCLUSIONS AND FUTURE WORK

Motivated from an industrial use case in the domain of data exchange, we found a lack of solutions for reverse-engineering updates of relational tables without knowledge of the record alignment. In particular, this pertains snapshots of tables with unspecified or modified primary keys. The resulting task requires record linking and function induction at the same time. To the best of our knowledge, we presented the first theoretical framework that explores both problems at once. As there are no straight-forward criteria that define the best solution, we suggested to measure the quality of a solution on the basis of minimum description length. While we could prove that the resulting optimization problem is NP-hard, we proposed an algorithm based on a best-first search to solve practical instances of the problem. We implemented a prototype of our algorithm called AFFIDAVIT and evaluated it on several problem instances of varying difficulty based on real-world datasets. The results confirmed that AFFIDAVIT scales linearly with the number of records and attributes. Moreover, we have identified a parameter configuration that can be used out-of-the-box to reliably produce correct explanations under practical levels of noise and transformations of the data. As our algorithm is completely unsupervised, this setting can be used to compare snapshots of databases with many tables without prior linking or labeling of the data by hand.

In practical problem instances, the meta functions implemented so far, would likely not be versatile enough to explain all data transformations. In its current form, AFFIDAVIT is most usable by administrators with domain knowledge about which meta functions commonly occur in their domain. They are able to customize AFFIDAVIT by adding further meta functions via implementation of a small Java interface. In Future Work, the prototype could be updated to support a richer set of functions by default. For instance, we recently added support for date conversions. Furthermore, it would be interesting to integrate a function corpus like it was done in TDE [15] instead of manually extending the supported functions.

Future work could also investigate a problem variant without knowledge of the schema alignment. Consequently, table modifications like attribute renaming, merging or splitting could be supported. We think the insights and methods of this work would be valuable for such a task as well.

REFERENCES

[1] ApexSQL. 2019. Data Diff. Retrieved March 25, 2019 from https://www.apexsql.com/sql-tools-datadiff.aspx

[2] Patricia C. Arocena, Boris Glavic, and Renee J. Miller. 2013. Value Invention in Data Exchange. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. ACM, New York, NY, USA, 157–168. https://doi.org/10.1145/2463676.2465311

[3] Manuel Atencia, Jérôme David, and Jérôme Euzenat. 2014. Data interlinking through robust linkkey extraction.. In *ECAI*. 15–20.

[4] Mikhail Yuryevich Bilenko. 2006. *Learnable similarity functions and their application to record linkage and clustering.* Ph.D. Dissertation.

[5] Peter Christen. 2012. *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection.* Springer Science & Business Media.

[6] William Gemmell Cochran. 1977. Sampling Techniques (Third Edition). (1977).

[7] Australian Software Company. 2019. SQL Delta. Retrieved March 25, 2019 from http://www.sqldelta.com/

[8] Spectral Core. 2019. Replicator. Retrieved March 25, 2019 from https://www.spectralcore.com/replicator

[9] devart. 2019. Data Compare. Retrieved March 25, 2019 from http://www.devart.com/dbforge/sql/datacompare/

[10] Ivan P Fellegi and Alan B Sunter. 1969. A theory for record linkage. *J. Amer. Statist. Assoc.* 64, 328 (1969), 1183–1210.

[11] Georg Gottlob and Pierre Senellart. 2010. Schema mapping discovery from data instances. *Journal of the ACM (JACM)* 57, 2 (2010), 6.

[12] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets Using Input-output Examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '11)*. ACM, New York, NY, USA, 317–330. https://doi.org/10.1145/1926385.1926423

[13] Sumit Gulwani, William R Harris, and Rishabh Singh. 2012. Spreadsheet data manipulation using examples. *Commun. ACM* 55, 8 (2012), 97–105.

[14] Sumit Gulwani, Mikaël Mayer, Filip Niksic, and Ruzica Piskac. 2015. StriSynth: Synthesis for Live Programming. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 701–704. http://dl.acm.org/citation.cfm?id=2819009.2819142

[15] Yeye He, Xu Chu, Kris Ganjam, Yudian Zheng, Vivek Narasayya, and Surajit Chaudhuri. 2018. Transform-data-by-example (TDE): An Extensible Search Engine for Data Transformations. *Proc. VLDB Endow.* 11, 10 (June 2018), 1165–1177. https://doi.org/10.14778/3231751.3231766

[16] Robert Isele and Christian Bizer. 2012. Learning expressive linkage rules using genetic programming. *Proceedings of the VLDB Endowment* 5, 11 (2012), 1638–1649.

[17] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. 2016. Magellan: Toward building entity matching management systems. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1197–1208.

[18] Oliver Lehmberg, Alexander Brinkmann, and Christian Bizer. 2017. WInte.R - a web data integration framework. In *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC 2017)*, Vol. 1963. RWTH.

[19] Mikael Mayer. 2017. String-Solver. Retrieved April 18, 2019 from https://github.com/MikaelMayer/StringSolver

[20] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. ACM, New York, NY, USA, 19–34. https://doi.org/10.1145/3183713.3196926

[21] George Papadakis, Leonidas Tsekouras, Emmanouil Thanos, George Giannakopoulos, Themis Palpanas, and Manolis Koubarakis. 2018. The return of jedAI: end-to-end entity resolution for structured and semi-structured data. *Proceedings of the VLDB Endowment* 11, 12 (2018), 1950–1953.

[22] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. 2015. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment* 8, 10 (2015), 1082–1093.

[23] Oleksandr Polozov and Sumit Gulwani. 2015. FlashMeta: A Framework for Inductive Program Synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2015)*. ACM, New York, NY, USA, 107–126. https://doi.org/10.1145/2814270.2814310

[24] Redgate. 2019. SQL Data Compare. Retrieved March 25, 2019 from http://www.red-gate.com/products/SQL_Data_Compare/index.htm

[25] Jorma Rissanen. 1983. A Universal Prior for Integers and Estimation by Minimum Description Length. *The Annals of Statistics* 11, 2 (1983), 416–431. http://www.jstor.org/stable/2240558

[26] Bo Wu and Craig A. Knoblock. 2015. An Iterative Approach to Synthesize Data Transformation Programs. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press, 1726–1732. http://dl.acm.org/citation.cfm?id=2832415.2832489