Playing LEGO with JSON: Probabilistic Joins over Attribute-Value Fragments

Manuel Hoffmann
University of Kaiserslautern
Kaiserslautern, Germany
mhoffmann@cs.uni-kl.de

Evica Milchevski
University of Kaiserslautern
Kaiserslautern, Germany
milchevski@cs.uni-kl.de

Sebastian Michel
University of Kaiserslautern
Kaiserslautern, Germany
smichel@cs.uni-kl.de

***Abstract*—Information about an entity can hardly be assumed to be given in one single document, created in a single instance of time. Rather, it is reasonable to assume that information is spread over multiple documents and created/enriched over time—for instance through crowdsourcing facts or mined from social network streams, one after the other. In this work, we consider the problem of assembling entity-centric information out of input comprising small pieces of information; provided in form of JSON document snippets. The final goal is to create a document that (possibly fully) describes an entity by putting related fragments together. What makes this task challenging is the lack of evidence telling which fragments belong together and, hence, can be safely combined. We focus on deciding this question using statistics of the already seen fragments, to justify if a join is reasonable or not. We evaluate our approach using real-world datasets and show that we can achieve high precision and recall.**

I. Introduction

Information about an entity is often collected over time and not created in one "big bang". Consider Web portals that collect critics, pictures, and other properties of restaurants. While initially there might be only few information like address and name of the restaurant, with evolving time more and more facts will be added. Take for instance Wikipedia, that serves as input to various knowledge bases [23], [2], where facts about entities resembling pages in Wikipedia are almost never complete but get constantly enriched over time. In the big-data era, this observation is even more drastic; information is often explicitly crowdsourced through tools like Amazon Mechanical Turk, posted to websites in form of reviews or comments, or disseminated by users through real-time feeds like Twitter. The two JSON documents in Figure 1 give details on a business in Phoenix, AZ. For instance, the left document might be created by a mobile device which automatically has attached the geolocation whereas the right document could be the result of a user entering data in a web form. If both documents relate to the same entity, the documents should not be presented separately to a visitor of a web portal but rather merged in order to provide complete information about this restaurant.

In recent work, we have outlined the main idea and research challenges of a system coined ligDB [21] that does not store data per se but collects and operates on small data fragments on demand. A core component of such a system needs to be able to assemble (join) information snippets into more complete knowledge—which resembles the motivation for the approach we describe in this work.

This work has been supported by the German Research Foundation (DFG) under grant MI 1794/1-1.

```
1 { "open": true,
2   "name": "Food City",
3   "longitude": -112.09,
4   "latitude": 33.39 }
```

```
1 { "open": true,
2   "name": "Food City",
3   "city": "Phoenix",
4   "state": "AZ" }
```

Fig. 1: Two documents extracted from the Yelp dataset, possibly describing the same entity.

Integrating data spread over multiple sources has been a widely researched field [12], [13]. The research in data integration mainly focuses on addressing the challenges of providing a unified querying access to data residing in multiple heterogeneous sources. However, the characteristics of the constantly changing data render traditional data integration approaches not suitable, and ask for devising new techniques. Recent work on uncertain data integration [1], [7], [22] has mentioned the need of extending the scope of data integration systems.

The approach proposed in this paper is tailored to handle dynamic data, it processes small pieces of data on-the-fly, and constantly builds new data on top of old one. We focus on the task of assembling the information about an entity dispersed over multiple (streaming) documents. Specifically, we consider the case of a stream of (small) documents where each document d_i is a set of attribute-value pairs. Attributes are simple strings, while values can be in form of integers, floats, boolean values, arrays of values, or are (nested) documents, too. It is very likely that some documents d_i may report facts about the same physical entity, hence, need to be gathered together.

A. Problem Statement

The core task can be described as follows. Given a stream of document fragments $\mathcal{F} = \{d_1, \ldots, d_i, \ldots, \}$, each d_i represented as a list of attribute-value pairs $a : v$, $a \in \mathcal{A}$ and $v \in dom(a)$, where $\mathcal{A}$ is a global set of attributes, we want to identify and merge all document fragments $d_j \in \mathcal{F}$ that describe facts about the same entity e. We need to assess whether a join (aka. merge) of two fragments (or iteratively among multiple fragments) is likely to be valid and is not putting together fragments that belong to different entities. By merging we refer to an operation that creates a new fragment as the union of all attribute-value pairs from either input fragment.

Consider the following gedankenexperiment (thought experiment) for the sake of emphasizing the main objective. We

take a set of documents D, where each $d \in D$ contains *all* available information about one specific entity. Each document we split into small fragments and give this resulting batch of fragments to our algorithm. The task of the algorithm is to reconstruct all documents $d \in D$ by assembling the individual pieces, while making sure not to create combinations of fragments that did not exist in D before.

Getting back to reality, we see that there is an unknown number of entities for which facts need to be merged together into documents describing those entities. For each evolving document that stems from data fragments collected over time, we need to compute a likelihood measure that this document is in fact a consistent representation of **only one** physical entity. If each fragment reports on a globally unique identifier, the problem becomes trivial, as this key can be used to group document fragments. However, the presence of such an id cannot be taken granted, as there is no globally accepted authority/organization that assigns unique ids to every possible entity. Thus, we need to asses, using reliable statistical models, whether or not an arriving document is introducing facts of a newly emergent entity, or is adding to facts of already existing entities.

B. Contributions and Outline

In this paper, we make the following contributions: We first analyze the problem of merging small sets of attribute-value pairs into larger documents. Next, we present ways of reasoning in order to avoid generating unnecessary documents and we assess the confidence that two or more fragments belong together. Finally, we report on the results from a performance evaluation using real-word data.

The remainder of this paper is organized as follows. Section II discusses related work, specifically work done in the area of data integration and entity matching. Section III describes the principles behind our approach and, in particular, ways to limit the joining of fragments as the result would likely be invalid, together with an algorithm for joining a given set of fragments. Section IV shows the implementation details of our approach. Section V reports on the setup and results of our experimental evaluation. Section VI concludes the paper.

II. Related Work

Data integration and data fusion is the task of integrating data from multiple sources and providing unified access to the data. It is a widely studied research field [12], [19], [13] which recently has again attracted the attention of the research community.

Fagin et al. [11] provide insights on the algorithmic approach for schema-mapping creation and query generation which they gained in the Clio project. A schema mapping describes how heterogeneously stored data, possibly in different sources, is related. In this project, they merge tuples representing the same entity based on properties of the schema at query time.

Dong et al. [7] examine data integration with uncertainty. They discuss the need for probabilistic schema mappings where either a probability is assigned to a mapping on a per-tuple basis, or a per-schema basis.

In our approach, we do not rely on the presence of a predefined schema, but, by observing attribute sets of documents, we rather use an implicitly given schema.

In [6], Dong et al. address the problem of finding the true value of an object or attribute when there are multiple sources for this information. They devise probabilistic models based on Bayesian analysis to reason about the accuracy of sources, the confidence of a value being true or false, and weather a source is a copier or not, thereby gaining a more accurate number of observed original values. They further propose an iterative algorithm, that given initial probability values, converges to the true probabilities.

Ioannou et al. [14] propose algorithms for query-driven entity merging in probabilistic linkage database, i.e., a probabilistic database with probabilistic linkage association between entities. In our entity merging approach, however, we do not assume any pre-knowledge on existing links between entities. Further, they assign probabilistic values to each entry of a record, while we assign only one value per total record.

Entity matching, or record matching is the task of identifying records from different structured sources, that refer to the same entity. This problem has been intensively studied in the past decades in different research communities, and different approaches have been proposed. Elmagarmid et al. [9] summarize the thus far research and discuss open questions that still need to be addressed.

Köpcke et al. [18] discuss automatic and semi-automatic entity matching techniques. They evaluate these techniques on the Fever platform focusing on attribute-based matching using different string similarity measures.

Kenig and Gal [17] focus on the subproblem of blocking entities for achieving better performance in the entity matching problem. Blocking is the process of grouping tuples together in such a way that tuples in different groups (blocks) must not refer to the same entity and tuples in the same group may refer to the same entity, serving as a preprocessing step for more complex comparisons inside the blocks. They propose an efficient blocking method based on the maximal frequent itemsets principle. Their algorithm does not need a user generated key for the blocking and ensures that the created blocks satisfy the compact set (CS) and sparse neighborhood (SN) [5] properties.

Du Bois [8] provides a probabilistic model for entity matching/linkage in the presence on missing (null) values. Entity matching seems similar to the problem we address in this paper, however, in entity matching one major concern is field matching using similarity functions. For instance, Wang et al. [24] propose an algorithm for finding the best similarity function and threshold for a given set of record-matching rules.

In contrast, we consider the case that the same value is represented the same way in all records, e.g., by a normalization step applied beforehand. Further, we do not want to classify our records in only two classes, identical or not, but we want to cluster all records that belong to the same entity, from possibly many entities.

Kannan et al. [16] address the problem of matching unstructured product offers to structured, attribute-value, product

descriptions. They apply supervised learning techniques to learn a probabilistic matching function, which is then applied at run-time for matching offers to products.

Ioannou et al. [15] devise a probabilistic method for inferring linkages between data that refer to the same entity. They use Bayesian inference to reason about the possibilities that two resources refer to the same entity, whereas we rely on frequency-based statistics. To do this, they construct a Bayesian network where they model both the entity matches, the evidence of entity match, and the relationship between entities. While we keep all effects introduced by an update minimal, in their approach the network is updated and the probabilities of linkage between resources are recomputed.

III. Approach

As input our system gets a stream of JSON document fragments, describing facts of various entities. We aim at correctly identifying those document fragments that **describe the same entity**. However, this is not a straightforward task. Consider an existing document $d_{old} = \{a : 1, b : 2\}$ and an arriving document $d_{new} = \{c : 3\}$. Although no information is shared between these documents, they could refer to the same entity and, thus, should be merged to a new document $d_{joined} = d_{old} \cup d_{new} = \{a : 1, b : 2, c : 3\}$. On the other hand, d_{old} and d_{new} might refer to different entities and thus should not be merged.

A naive algorithm would join all fragments in all possible ways, and would also definitively reconstruct all documents—reaching a recall of 100%. The drawback is, however, that also many documents are created by joining fragments that do not belong to the same entity—leading to a low precision.

What we propose in this work is a viable approach and a generic framework that applies simple statistics to reason about the likelihood that a join of two fragments is valid or should be avoided.

Let $\mathcal{D}$, $\mathcal{D} \subset \mathcal{F}$, be the set of previously observed document fragments together with already joined fragments. Algorithm 1 captures the basic idea of our approach—an arriving fragment can be merged with any already seen document fragments, if the two are joinable.

```
/* generation with replacement */
input: document fragment d
       and set of known docs D
1  foreach d_old ∈ D do
2      if joinable(d, d_old)
3          D ← D ∪ {d ∪ d_old} /* d_old remains in D */
4  D ← D ∪ {d}
```

Alg. 1: Algorithm capturing the basic idea of our approach.

One important decision is the way new and old fragments are handled. In the above pseudocode we see in line 3 that the fragment created by the merge is placed in $\mathcal{D}$ while the old fragment d_{old} is also kept in $\mathcal{D}$. Based on the terminology of urn models, we refer to this as generation with replacement. In contrast, a generation without replacement would not keep the original fragment d_{old}. The question of which method works better raises naturally. For instance, consider that a specific set of full (not fragmented) documents is cut into fragments and sent to our system, one fragment after the other. Now, imagine, that all joins are valid ones, i.e., there is no uncertainty in deciding whether or not a join is valid or not. In that (unrealistic) case, generation without replacement would not do any harm; in contrast, it would be very efficient to remove old fragments from the pool of fragments $\mathcal{D}$. In any other case, most importantly due to mistakes in deciding if a join is valid, we need to keep original fragments in $\mathcal{D}$.

The second crucial decision is when do we consider two documents to be *joinable*. We discuss this in the following section.

A. Joinability of Document Fragments

We start by defining conflicting document fragments, which gives us a criterion for exclusion of fragments from being joined.

Definition 1: Given two document fragments d_i and d_j. d_i and d_j are **conflicting** if they disagree on the value of any attribute c, i.e., $d_i[c] \neq d_j[c]$.

Consider for instance the example in Table I (left). There are two fragments that have a common attribute *open* but with a different value. Hence, these fragments are conflicting and should not be merged as they cannot stem from the same full-fledged document.

Next we define an overlap between two fragments:

Definition 2: Given two document fragments d_i and d_j, the set of attributes c that are present in both fragments and for which $d_i[c] = d_j[c]$ holds is called **overlapping attributes**, and denoted $\mathcal{A}(d_i \cap d_j)$. The size of this set is called an **overlap**. If this set is non-empty, d_i and d_j are **overlapping**.

Consider the example in Table I (right). The two fragments are overlapping because they have the overlapping attribute "full_address".

We combine these two aspects and get a syntactical requirement which every pair of document fragments must satisfy in order to be joined together:

Definition 3: Two document fragments d_i and d_j are **joinable** if they are not conflicting and they are overlapping. [1]

We use the words join and merge interchangeably and denote this operation between two document fragments d_1 and d_2 with $d_1 \cup d_2$. However, not every joinable pair of document fragments should be joined. For instance, a join over a "primary" key is certainly valid, compared to joining two fragments based on a boolean attribute. We say that a join is **valid** if the joined fragments refer to the same entity, otherwise the join is **erroneous**.

Intuitively, our goal is to perform only valid joins, however, unless a unique document id is given in each of the fragments, this is not possible. Thus, we introduce a confidence value for every document d, denoted $\kappa(d)$, that measures the probability

[1] Note that to decide if two fragments should be joined or not we consider only exact matching attributes and values, however our work can easily be extended to accept also similar values. Matching entities with similar values (attributes) has already been extensively studied before [9], and therefore, is out of the focus of this work.

Conflicting		Non-Overlapping		Overlapping	
{ “open”: **true**, “cuisine”: “French” }	{ “open”: **false**, “city”: “Mobile” }	{ “open”: true, “cuisine”: “Cantonese” }	{ “full_address”: “845 W SouthernAve Phoenix, AZ 85041” }	{ “open”: true, “cuisine”: “Cantonese”, **“full_address”: “845 W SouthernAve Phoenix, AZ 85041”** }	{ **“full_address”: “845 W SouthernAve Phoenix, AZ 85041”**, “name”: “Food City” }

TABLE I: Illustration of conflicting (left), non-overlapping (middle), and overlapping (right) pairs of fragments.

that d is a valid document. If d is a document from the input stream, we set $\kappa(d) = 1$ since we assume that there is no false information in the input. In order to prune joins that lead to very uncertain documents, we further introduce the parameter **minimum required confidence** θ and only allow joins between document fragments with $\kappa(d_i \cup d_j) \geq \theta$ to be made.

In order to compute the confidence of a merge $\kappa(d_i \cup d_j)$, i.e., the probability that such a document exists, we rely on attribute statistics. Consider for instance the following two fragments:

```
1 { “business_id”: “usAsSV36QmUej8–yvN-dg”,
2   “full_address”: “845 W SouthernAve Phoenix,
                                    AZ 85041”,
3   “open”: true }
```

and

```
1 { “full_address”: “845 W SouthernAve Phoenix,
                                    AZ 85041”,
2   “name”: “Food City”}
```

Unfortunately, the unique “business_id” is not given in *both* of the document fragments, but they do share the same “full_address” and it appears that these two fragments indeed talk about the same physical entity. If this shared part was the country name or the boolean state “open”, it would be difficult to confirm the validity of the join. Following this reasoning, we say that an $a : v$ pair is a good join candidate if there is a low probability that some document d has exactly this attribute-value combination (i.e., $P[d.a = v]$). This means, ideally there are plenty of distinct values for this attribute and if two fragments really agree on the value for a they “must” belong together.

We call this probability the **success probability for attribute** a, denoted p_a. Assuming the document fragments agree on the value of attribute a being v and there are $|a_v|$ entities having v associated with a, then $p_a = 1/|a_v|$. We rely on uniformly distributed values, but note that if the exact distribution for attribute a is known, then a more precise answer can be given.

If two fragments d_i and d_j overlap in exactly one attribute a, then $\kappa(d_i \cup d_j) = p_a$. But in general, it might happen that d_i and d_j have more than just one attribute in common. Let $\mathcal{A}(d_i \cap d_j) = \{a_1, a_2, \dots a_k\}$ be the attributes where the document fragments overlap, and p_i denote the success probability for a_i. Then, using the converse probabilities to combine the single p_i, we get this value p as generalization for arbitrary overlaps:

$$p = 1 - \prod_{i=1}^{k} (1 - p_i)$$

This formula implies that a join between documents having attributes $\mathcal{A}$ in common is more likely to be valid than a join between documents having attributes $\mathcal{A}' \subset \mathcal{A}$ in common. Intuitively, this means that with every additional attribute, more evidence is seen that two fragments should be merged.

Consider two document fragments having overlapping attributes $a =$ “full_address” and $b =$ “open” with success probabilities $p_a = 0.95$ and $p_b = 0.01$. Then the confidence we have in this join is 0.9505, which is significantly higher than with only b as overlapping attribute and a little higher than only with a.

Since not only input fragments can serve as join partners, but also results from previous join operations, we have to take into consideration that previous joins could have been erroneous. For instance, consider a document fragment d_i that was created by a join over rather insignificant attributes and which is rated just above θ. If now a document fragment d_j can be joined with d_i over a very significant attribute, then the result should not have a higher κ-value than d. We achieve this by reconsidering the κ-values of the source documents of a join and thereby get the formula we use for assessing the confidence:

$$\kappa(d_i \cup d_j) := \kappa(d_i)\ \kappa(d_j) \left(1 - \prod_{i=1}^{k} (1 - p_i)\right)$$

As before, we assume an overlap of k between d_i and d_j.

B. Harnessing Attribute Co-Occurrences

One aspect we did not discuss so far is preventing a join between two different types of documents. While looking at the overlapping attributes helps, to some extent, in avoiding such joins, it does not eliminate this possibility completely. For instance, consider a document fragment describing the sports club ”Liverpool”, and another one containing information about the city. They might have the overlapping information `"city": "Liverpool"` but by joining them together we would create the document `{"club": "FC", "city": "Liverpool", "inhabitants": 10.000}`. It clearly makes no sense to display the inhabitants of a sports club.

To prevent this, take in consideration the set of (as input) observed document fragments, $\mathcal{D}, \mathcal{D} \subset \mathcal{F}$: If the join of two

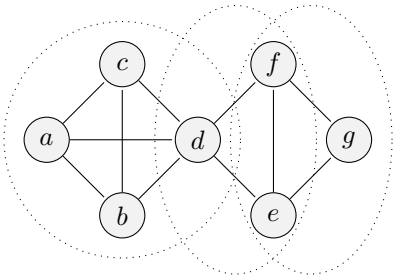


Fig. 2: Co-occurrence graph built from three fragments.

documents creates a new document consisting of attributes that have never been observed together in a valid input, it is likely, that these attributes should not be merged.

Definition 4: Given two document fragments d_i and d_j with attribute sets A_i and A_j. d_i and d_j have **compatible attributes** if all attribute combinations $(a_i, a_j), a_i \in \mathcal{A}_i$ and $a_j \in \mathcal{A}_j$ are seen in some document d in $\mathcal{D}$, i.e., $a_i, a_j \in \mathcal{A}(d)$.

We model this using a co-occurrence graph $\mathcal{G}_\mathcal{D} = (V, E)$, where every node $v \in V$ represents an attribute observed in a document fragment $d \in \mathcal{D}$. An edge $(v_i, v_j) \in E$ indicates that there is a fragment $d \in \mathcal{D}$ with $v_i, v_j \in \mathcal{A}(d)$. For instance, the co-occurrence graph shown in Figure 2 is the result of observing three document fragments with attribute sets $\{a, b, c, d\}$, $\{d, e, f\}$ and $\{e, f, g\}$. To check whether two fragments d_1, d_2 with attributes $\{a, b\}$ and $\{b, d\}$ can be merged, it is sufficient to check if all edges between the attributes of $d_1 \cup d_2$ are realized, which they are indeed. In contrast, d_1 and d_3 with $\mathcal{A}(d_3) = \{b, e\}$ do not have compatible attributes, since the edge between a and e is not realized. Note, that checking compatibility is the same as checking whether the attributes of $d_1 \cup d_2$ are a subset of any maximal clique in the co-occurrence graph.

Furthermore, using this graph we can find the expected size of documents to build. For instance, consider the scenario of data describing restaurants. A document fragment containing only the state, the review count and the type of the restaurant is not complete enough to be presented to the user even if the system cannot attach any more data to this fragment. If we know the number of mandatory attributes that a document about a certain entity type must have, then we can avoid outputting document fragments containing less attributes than this. The information about which and how many attributes belong to which type of documents can be learned by looking at its (maximum) cliques.

However, relying on the attribute co-occurrence graph could be ill-conditioned if it takes too long to learn the structures of different document types. For instance, if hundreds of thousands of fragments would have to be observed before the first join can be made, then this would heavily delay, or even prevent the formation of valid documents.
To compute how long it takes to see all attribute combinations of an attribute set by observing random subsets of attributes, we reformulate the problem to an instance of the coupon collector's problem [28]. The solution to this problem describes how many coupons a collector needs to buy, in expectation, in order to collect all distinct coupons available. We specifically consider the variant of the coupon collector's problem where the collector acquires packages of coupons, such that within each package, there are no duplicate coupons. Applied to our setting, the collector needs to collect a total of $m = |E|$ coupons, where the size of each package is the same as the number of attribute-value pairs in a fragment, denoted k. Then the number of packages that need to be acquired to get the i^{th} distinct coupon can be estimated as:

$$h(m, i, k) = \frac{m - (i \bmod k)}{m - i}$$

And overall the expected number of packages to be observed is then:

$$E(X_{all}) = \frac{1}{k} \sum_{i=0}^{m-1} h(m, i, k)$$

For instance, if a class of entities is fully described by 20 attributes and we observe fragments of size 10, then we need only to observe less than one thousand fragments about any entity of this type, in order to observe every co-occurrence in expectation.

IV. Implementation

A. Joining Algorithm

A verbatim implementation of Algorithm 1 would have a very poor performance, as the new document is tested against each document $d_{old} \in \mathcal{D}$. Since only those d_{old} that are joinable with the arriving document are of interest, we keep the number of to be investigated documents small. We determine a subset $\mathcal{D}_C \subseteq \mathcal{D}$, containing all candidate documents that can be joined with the arriving document fragment d, i.e., $\mathcal{D}_C = \{d_i | d_i \in \mathcal{D} \wedge joinable(d_i, d)\}$. Hence, it suffices to test joinability with documents in $\mathcal{D}_C$, where it is reasonable to assume that $|\mathcal{D}_C| \ll |\mathcal{D}|$.

We over-approximate $\mathcal{D}_C$ by using indices over the attribute-values of the document. The system maintains an index per $\mathcal{A}$ where each entry relates a value to the list of documents which contain the same attribute-value pair. When a document $d = \{a_1 : v_1, \ldots, a_k : v_k\}$ enters the system, each index for a_i is queried for the document list for value v_i. Then $\mathcal{D}_C$ consists of the union of those lists. Note, that this is a strict over-approximation. Consider an arriving document $d = \{a : 1, b : 1\}$, the index lookup can find the document $\{a : 1, b : 2\}$, which is conflicting with d.

As a first step towards solving the proposed problem, we have implemented the presented idea in an offline algorithm, depicted in Algorithm 2. Upfront, the input fragments are used to build the statistics, the document structures are detected, and all fragments are indexed. The algorithm maintains two working sets of documents, D and D', the former contains document fragments which are inspected in the current iteration, the latter contains the results of joins of the current iteration which are to be inspected in the next iteration. Besides, all document fragments, input and join-results, are collected in $\mathcal{D}$. The candidates determined in line 4 are a subset of $\mathcal{D}$.

This process stops, when no further join is conducted, i.e., a fixed point is reached. Then, all documents which are deemed complete are output (lines 12–14).

The joinable predicate consists of several checks. Since the index lookup provides an overestimation, it is necessary to

```
input: document fragments D
1  build_stats(D); build_shapes(D); index(D)
2  D' ← {}; 𝒟 ← D
3  foreach probe ∈ D
4      candidates ← index_lookup(probe)
5      foreach c ∈ candidates
6          if joinable(probe,c)
7              d ← join(probe,c)
8              D' ← D' ∪ {d}; 𝒟 ← 𝒟 ∪ {d}
9  if D' ≠ {}
10     D ← D'
11     goto 2
12 foreach d ∈ 𝒟
13     if complete(d)
14         output(d)
```

Alg. 2: Basic modus operandi of the offline join-algorithm.

adjacency list:		**maximal cliques:**
$a \mapsto$ {c,d}	$d \mapsto$ {e,f}	$\{\{a, c, d\},$
$b \mapsto$ {c,d}	$e \mapsto$ {f}	$\{b, c, d\},$
$c \mapsto$ {d}	$f \mapsto \{\}$	$\{d, e, f\}\}$

Fig. 3: Co-occurrence graph and internal representation before fragment f' with $\mathcal{A}(f') = \{a, b\}$ is inserted.

check if the probe and the candidate are not conflicting. Then the confidence value of the join is compared with the threshold and the compatibility is examined.

B. Dynamically Listing Maximal Cliques

Enumerating maximal cliques is in general an NP-complete problem and there are algorithms which tackle this problem [4], but they are oftentimes designed for large, sparse graphs [20], [10]. However, in our case, the co-occurrence graph stays rather small and dense. Furthermore, in an online setting, the co-occurrence graph is potentially updated with every arriving document. Also, the updates are not random, but every time a document is observed, a whole complete subgraph is inserted, i.e., a clique over the attribute set of the fragment.

We exploit this special case, to write a custom procedure that reuses previous results. To detect when the document fragments are complete, our system needs to register the structure from a fragment, evaluate an attribute combination and find and report the complete structures. Thus, to provide more efficient access, we materialize all maximal cliques of the co-occurrence graph.

Consider the co-occurrence graph and list of maximal cliques in Figure 3 and further assume the system observes a fragment with $\mathcal{A}(f) = \{c, d\}$. The update function then checks whether there is a maximal clique stored that is a superset of $\{c, d\}$, and it finds $\{a, c, d\}$, hence nothing changes. Now, the system observes a fragment with $\mathcal{A}(f') = \{a, b\}$. In this case, there is no superset in the maximal cliques set, thus a new edge is inserted. The algorithm determines that cliques which can be affected by the insertion are $\{a, c, d\}$ and $\{b, c, d\}$ because they both share a vertex with $\mathcal{A}(f')$. The system augments the existing cliques with the edges of $\mathcal{A}(f')$, and searches for new, bigger cliques, in this case $\{a, b, c, d\}$. Lastly, all subsets of the newly discovered cliques are removed, as they are no longer maximal, and we end up with the maximal cliques $[\{a, b, c, d\}, \{d, e, f\}]$.

In order to determine whether two fragments f_1, f_2 are compatible, it is then enough to check if $\mathcal{A}(f_1 \cup f_2)$ is a subset of any of the maximal cliques.

C. Indexing and Statistic Generation

For indexes as well as statistics we maintain nested hash-maps. The outer map relates attributes to inner maps which themselves relate values to lists of references to fragments, and counters respectively. Additionally the sum of all counters for the values is materialized. This way, all lookups for fragments and statistics can be performed in amortized constant time.

V. Experiments

We have implemented the proposed algorithm in the programming language Rust. The experiments are conducted on an Intel i5 CPU machine with 8GB RAM, running Archlinux Kernel 3.19.2. All experiments were executed in main memory.

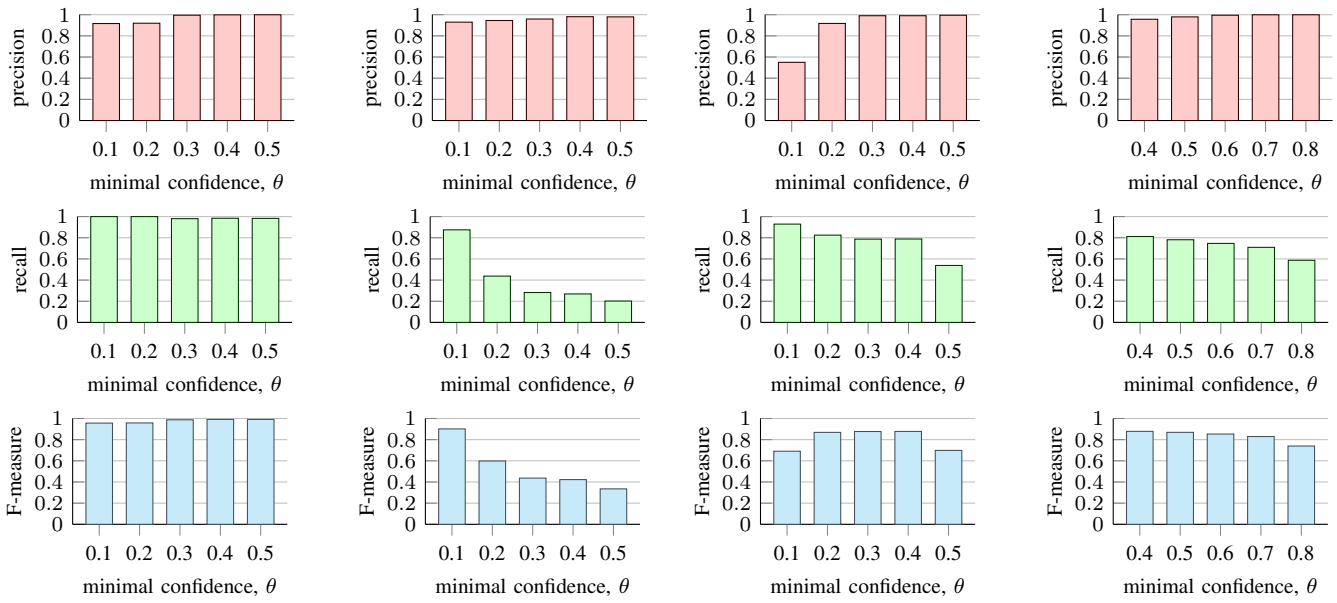
To evaluate the proposed algorithm we did the following: given a set of entities represented as JSON documents, each of them later randomly torn apart in overlapping fragments, we evaluate how good we can merge together fragments to resemble the initial set of entities/documents using the proposed algorithm.

Datasets: We evaluate our approach using real-world datasets. From the Yelp Academic Dataset [25] we use the business (YB), review (YR), and user (YU) data and from the British Governments' open data portal we use road safety (RS) data [26] and land-registry price-paid (PP) data [27]. The documents in each set are homogeneous, i.e., they have all the same structure. Among each other, the datasets vary with respect to number of documents, number of attributes per document, and in the attribute cardinalities, i.e., the different values for each attribute, as shown in the analysis in Table II. Interestingly, all data sets except RS have an uniquely identifying attribute.

We partition each document of those data sets uniformly at random into τ fragments overlapping by η attributes. We use these fragments as input to our algorithm and compare the output with the original entries of the datasets measuring **precision**, **recall**, and the combined **F-measure**. Precision is the fraction of correctly identified original documents over all generated documents and recall is the fraction of all correctly identified documents over all input documents. In other words, precision measures the erroneous joins—the lower the precision the more erroneous joins are made—and recall measures the completeness of the results. The F-measure is the harmonic mean of precision and recall and gives a unified impression on the quality of the results.

Reassembling fragments stemming from a single type of documents: We start by reassembling fragments that originate from one type of documents. Thereby we initially ignore the problem of deciding to which type one fragment belongs.

Figure 4a shows the results of the joining procedure, applied to a part of the YR dataset, that is fragmented with



(a) Yelp review dataset, $\eta = 3$, $\tau = 2$

(b) Road safety dataset, $\eta = 3$, $\tau = 2$

(c) Yelp review dataset, $\eta = 2$, $\tau = 3$

(d) Mixed dataset, $\eta = 3$, $\tau = 2$

Fig. 4: Precision, recall and F-measure for different datasets and varying parameters of overlap η and number of partitions τ.

	Size	#attr	Min card	Max card
Yelp Business (YB)	11537	13	1	11537
Yelp Review (YR)	229907	8	1	229907
Yelp User (YU)	43873	6	1	43873
Price Paid (PP)	70097	15	2	70097
Road Safety (RS)	252913	21	3	138660

TABLE II: Statistics on the original datasets used in the experiments. "Size" shows the number of documents per data set, "#attr" how many attributes each document is composed of. "Min (Max) card" shows the minimum (maximum) value of how many different values exist per attribute.

parameters $\tau = 2$ and $\eta = 3$, for different values of θ, the minimum required confidence. Each document is split up into two fragments which share three attributes and each fragment consists of eight attributes. We see that for all values of θ we achieve high recall while also having a very high precision (over 90%). For a confidence θ of 0.3 and higher, a precision of 1 is achieved meaning no erroneous joins were made. On the other hand, for $\theta \leq 0.2$, we were able to recreate all the original documents, reaching a recall of 1. Although for all values of θ we were not able to recreate all the original fragments without making any mistakes, our algorithm did not make any mistake, while recreating 98% of the original documents for values of θ higher than 0.3.

We performed the same experiment for the RS dataset. The results are shown in Figure 4b. We see that in general the system was able to recreate considerably less of the original fragments with a lower precision, compared to the YR. While still having a precision of over 90% for all values of θ, the recall drops significantly when increasing θ. For values of θ larger than 0.2 we were able to recreate less than 50% of the original documents. This can be explained by the absence of high cardinality attributes in RS where for valid joins the three overlapping attributes together less often induce a high enough confidence.

Further, these two plots reveal the monotonicity of the precision and recall, while varying the value of θ: by lowering θ more joins are made by the algorithm. Hence, for a small values of θ, the likelihood for recreating all fragments increases, however, together with the number of mistakes. Vice versa, for higher values of θ, the join decisions are more conservative, and thus the chance for making a mistake is lower, but at the same time, this leads to missing some true results.

To see how the fragmentation of documents affects the efficiency of our algorithm, we fragmented the Yelp Review dataset with $\eta = 2$, $\tau = 3$, i.e., a document d is split up into $d_1 \cup d_2 \cup d_3$ such that d_1 and d_2 as well as d_2 d_3 have two overlapping attributes. The results as shown in Figure 4c. We see that the precision remains high, for $\theta \geq 0.2$, while the recall slightly drops. However, the system was still able to recreate more than 79% of the fragments for $\theta \leq 0.4$. This setting leads to in general lower efficiency of our system, since a higher recall can only be reached by sacrificing the precision as seen at $\theta = 0.1$, where the precision drops to a half and only 0.93 percent recall is reached. This can be explained by the fact that, in this case, our system is fed not only with more fragments, but also fragments with a lower overlap, leading to more uncertainty in the joining process.

Reassembling fragments stemming from multiple types of documents: Another important aspect of our system is how good it can handle fragments stemming from different types of documents. To test this, we fragmented all the datasets, using $\eta = 3$, $\tau = 2$, and fed all the fragments to our system. Figure 4d shows the results of this experiment. We

see that even when the fragments stem from different types of documents, for a threshold of $\theta = 0.7$ the system achieved 100% precision and about 70% recall. Also, for a smaller value of the minimum confidence, $\theta = 0.4$, the system made almost no mistake while restoring 81% of the original data. This shows that our system can make the "right" merge decisions even when the fragments are stemming from different type of documents, by leveraging the attribute co-occurrence statistics.

VI. Conclusion

In this work, we proposed an approach to build up larger JSON documents based on small document fragments that arrive at the system. We considered ways to decide whether two fragments can be joined together or not, using frequency statistics on the shared attributes and by leveraging attribute co-occurrence statistics. We evaluated the approach using two real-world datasets, where JSON documents are cut into (overlapping) fragments that are later-on being merged together—hopefully reassembling the original documents without making erroneous decisions. We showed that our system was able to recreate many of the original documents, while making no, or only few mistakes.

References

[1] P. Agrawal, A. D. Sarma, J. D. Ullman, and J. Widom, "Foundations of uncertain-data integration," *PVLDB*, vol. 3, no. 1, pp. 1080–1090, 2010.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, "Dbpedia: A nucleus for a web of open data," in *ISWC/ASWC*, ser. Lecture Notes in Computer Science, Springer, 2007, pp. 722–735.

[3] G. Blom, L. Holst, and D. Sandell, *Problems and Snapshots from the World of Probability*. Springer New York, 1994.

[4] C. Bron and J. Kerbosch, "Algorithm 457: Finding all cliques of an undirected graph," *Commun. ACM*, vol. 16, no. 9, pp. 575–577, Sep. 1973.

[5] S. Chaudhuri, V. Ganti, and R. Motwani, "Robust identification of fuzzy duplicates," in *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, IEEE Computer Society, 2005, pp. 865–876.

[6] X. L. Dong, L. Berti-Equille, and D. Srivastava, "Data fusion: Resolving conflicts from multiple sources," in *WAIM*, ser. Lecture Notes in Computer Science, Springer, 2013, pp. 64–76.

[7] X. L. Dong, A. Y. Halevy, and C. Yu, "Data integration with uncertainty," *VLDB J.*, vol. 18, no. 2, pp. 469–500, 2009.

[8] N. D. Du Bois Jr, "A solution to the problem of linking multivariate documents," *Journal of the American Statistical Association*, vol. 64, no. 325, pp. 163–174, 1969.

[9] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 1, pp. 1–16, 2007.

[10] D. Eppstein, M. Löffler, and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," *ACM Journal of Experimental Algorithmics*, vol. 18, 2013.

[11] R. Fagin, L. M. Haas, M. A. Hernández, R. J. Miller, L. Popa, and Y. Velegrakis, "Clio: Schema mapping creation and data exchange," in *Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos*, ser. Lecture Notes in Computer Science, Springer, 2009, pp. 198–236.

[12] L. M. Haas, "Beauty and the beast: The theory and practice of information integration," in *Database Theory - ICDT 2007, 11th International Conference, Barcelona, Spain, January 10-12, 2007, Proceedings*, ser. Lecture Notes in Computer Science, Springer, 2007, pp. 28–43.

[13] A. Y. Halevy, A. Rajaraman, and J. J. Ordille, "Data integration: The teenage years," in *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, ACM, 2006, pp. 9–16.

[14] E. Ioannou, W. Nejdl, C. Niederée, and Y. Velegrakis, "On-the-fly entity-aware query processing in the presence of linkage," *PVLDB*, vol. 3, no. 1, pp. 429–438, 2010.

[15] E. Ioannou, C. Niederée, and W. Nejdl, "Probabilistic entity linkage for heterogeneous information spaces," in *Advanced Information Systems Engineering, 20th International Conference, CAiSE 2008, Montpellier, France, June 16-20, 2008, Proceedings*, ser. Lecture Notes in Computer Science, Springer, 2008, pp. 556–570.

[16] A. Kannan, I. E. Givoni, R. Agrawal, and A. Fuxman, "Matching unstructured product offers to structured product specifications," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, ACM, 2011, pp. 404–412.

[17] B. Kenig and A. Gal, "Mfiblocks: An effective blocking algorithm for entity resolution," *Inf. Syst.*, vol. 38, no. 6, pp. 908–926, 2013.

[18] H. Köpcke, A. Thor, and E. Rahm, "Learning-based approaches for matching web data entities," *IEEE Internet Computing*, vol. 14, no. 4, pp. 23–31, 2010.

[19] M. Magnani and D. Montesi, "A survey on uncertainty management in data integration," *J. Data and Information Quality*, vol. 2, no. 1, 2010.

[20] G. Manoussakis, "Listing all maximal cliques in sparse graphs in optimal time," *CoRR*, vol. abs/1501.01819, 2015.

[21] E. Milchevski and S. Michel, "ligdb - online query processing without (almost) any storage," in *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, OpenProceedings.org, 2015, pp. 683–688.

[22] F. Sadri, "On the foundations of probabilistic information integration," in *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, ACM, 2012, pp. 882–891.

[23] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *WWW*, ACM, 2007, pp. 697–706.

[24] J. Wang, G. Li, J. X. Yu, and J. Feng, "Entity matching: How similar is similar," *PVLDB*, vol. 4, no. 10, pp. 622–633, 2011.

[25] "Yelp Academic Dataset," https://www.yelp.com/academic_dataset.

[26] "Road Safety Data," https://data.gov.uk/dataset/road-accidents-safety-data.

[27] "Price Paid Data," https://www.gov.uk/government/statistical-data-sets/price-paid-data-downloads.

[28] P. Flajolet, D. Gardy, and L. Thimonier, "Birthday Paradox, Coupon Collectors, Caching Algorithms and Self-Organizing Search," *Discrete Applied Mathematics*, vol. 39, no. 3, 1992.