Flexible and Efficient Distributed Resolution of Large Entities*

András J. Molnár, András A. Benczúr, and Csaba István Sidló

Data Mining and Web Search Group, Informatics Laboratory
Institute for Computer Science and Control, Hungarian Academy of Sciences
{modras, benczur, sidlo}@ilab.sztaki.hu

Abstract. Entity resolution (ER) is a computationally hard problem of data integration scenarios, where database records have to be grouped according to the real-world entities they belong to. In practice these entities may consist of only a few records from different data sources with typos or historical data. In other cases they may contain significantly more records, especially when we search for entities on a higher level of a concept hierarchy than records.

In this paper we give theoretical foundation of a variety of practically important match functions. We show that under these formulations, ER with large entities can be solved efficiently with algorithms based on MapReduce, a distributed computing paradigm. Our algorithm can efficiently incorporate probabilistic and similarity-based record match, enabling flexible match function definition. We demonstrate the usability of our model and algorithm in a real-world insurance ER scenario, where we identify household groups of client records.

1 Introduction

Entity Resolution (ER) is the process of identifying groups of records that refer to the same real-world entity. The process was described in several contexts under many different names: duplicate detection, instance identification, heterogeneous join, merge/purge, reference reconciliation, or object matching. Closely related topics include clustering, similarity join, string similarity, data cleaning, data warehousing, data integration and information integration.

In most cases, records are heterogeneous and erroneous and hence the mapping to hidden real-world entities is not straightforward. Structural and syntactic heterogeneity originates mostly from the heterogeneity of source systems, difference in data handling policies, standards, and finally from low data quality due to typos, missing values and other problems. ER can be therefore handled as a data cleansing task, occurring in data integration scenarios often.

Entity resolution is an actively researched area, and the problem can be formulated in many different ways. Input and output can be a set of records with attributes, a set of XML documents or a graph. The algorithms can either produce exact results or probabilistic mappings. Match functions can be defined by

* This work was supported by the EU FP7 SEC project SCIIMS (Ref. 218223).

T. Lukasiewicz and A. Sali (Eds.): FoIKS 2012, LNCS 7153, pp. 245–264, 2012.

exact rules, by similarities or by links between records. Results can be represented by record sets, by representative merged elements, or both. Training data or entity activity log can be present. The architecture used to solve the problem can be distributed, can be a single database server, a standard standalone computer or, for another example, a data mining framework.

Our main contributions are the following. We present an extended data model and problem formulation based on indexable features that facilitate the formulation of the given business logic, concentrating on match functions of independent entity properties, and providing a framework for defining efficient indexes. Based on the model we describe a scalable distributed algorithm for MapReduce. The algorithm is able to scale up to hundreds of millions of records and copes with large entities as well.

The rest of this paper is organized as follows. After giving an overview of the related work, we describe a motivating insurance ER scenario and enumerate issues of creating a client database. We formally define the ER problem in Section 3 based on the concept of indexable features. Our distributed algorithm is described in Section 5. Techniques to define indexable features for efficient resolution are described in Section 6. Finally, evaluation of the proposed methods is given in Section 7.

1.1 Related Work

One of the first descriptions of record linkage appears in the influential paper of Fellegi and Sunter [18] in 1969, describing a probabilistic model. Since then, entity resolution problems have been studied in many different disciplines and names. For overview, in [17] a survey is given on duplicate record detection, describing supervised, unsupervised and active learning, and summarizing statistical and machine learning solutions based on various text similarity and match measures. Recently the book [38] introduces key models, methods and new trends from a more practical point of view.

Traditional deduplication approach uses similarity measures for attributes, and learns when two records can be resolved to the same entity. A survey of string similarity functions can be found in [17], along with a survey of basic duplicate detection algorithms. In [20] a nice solution is presented for implementing string-similarity joins using q-grams in an RDBMS environment.

ER can be handled as a supervised learning problem, if training data is present. We can apply data mining classification methods, for example Bayes methods [23,18], decision trees [31] or SVM [7,11]. Unsupervised learning methods such as latent Dirichlet allocation [3] or clustering methods can also be used, if there is no training data. An interesting approach lying between the previous two is called active learning: when a small set of training data is given, the algorithm decides the new elements it could use the best to extend the training set ([34]). An automated training data selection method is described in [28]. Recently, [10] shows that cost sensitive alternating decision trees are practical for industrial applications weighting type I and II errors, while producing easily interpretable models.

ER is formalized many times as generating clusters of linked records. In the citation database scenario, with the goal of identifying authors, we do not really have author attributes other than their names. We can however link these records by joint papers. This way ER can be seen as a special problem of link-mining; a survey containing link based entity resolution can be found in [19]. The approach is called relational ER, based on the relations between records, or collective ER, because we would like to resolve records based on the link graph as a whole.

Entity resolution as a hypergraph clustering problem can be found in [4], under the name of relational clustering. Input data is handled as a reference graph, with nodes as entity records and edges as links between these nodes. The goal of the resolution process is to produce a resolved entity graph, where nodes are entity instances that hold entity records together. Clustering is also suggested [24]; however, general clustering methods are usually designed for larger and less number of clusters than records of entities in ER.

A seminal paper, [2] (published first in 2005) introduces generic entity resolution with black-box match and merge functions, where resolution means the closure of the original entity set according to these functions. Simple feature indexes are also used. The model and the algorithms are extended in [32] for handling approximate results as records with confidences. [1] adapts the algorithms to a distributed environment. Our generic ER algorithms for relational databases were published in [35].

Other interesting approaches to ER include utilizing aggregate constraints [9], or giving methods for query time ER [5]. In [6] a unified model is suggested for entity identification and document categorization. [43] widens the coreference problem with schema matching and canonization, and provides a unified model. The role of cross-field dependencies is described in detail in [22].

Recently, several new ER results were published. A new approach can be found in [44]: entity behavior is recorded as a transactional log. Common patterns of these transactions are used to identify similar or identical entities. Measuring the quality of entity resolution results is a crucial problem [33], dealing with possible quality metrics. [41] enhances core ER algorithms by combining the results of different blocking strategies. [21] exploits the role of constraints when finding duplicates. [40] deals with the effect of match/merge rule evolution, and gives methods to preserve results when rules change. [14] builds special inverted indexes to speed up ER with blocking. A survey of indexing techniques available for deduplication is provided in [12], including blocking, sorted neighborhood, Q-grams and canopies.

Entity resolution frameworks including SERF, MTB, DDUpe and MARLIN (see [29] for a survey) integrate several variations of the problem formulation in effective systems. A practical comparison of ER approaches can be found in [30] using the FEVER framework. The Febrl framework also provides parallelization [13]. Other parallel algorithms are presented in [26], tested on a few thousands of records. More recently [27] introduces parallel match and a distributed infrastructure, using similarity-based matchers.

In [36] we studied how efficient indexing methods can be used to speed up the ER process. Our first results on scalable parallel ER were published in [37].

2 Motivating Example

Companies typically face the entity resolution problem when building a client database, or manage client master data. Clients may appear multiple times in multiple source systems, e.g. a record for a contract, another for a purchase. As another example, the same person may appear in several marketing databases obtained by different means. ER is the key step in producing sound and clean client master data.

Our motivating application is the client data integration of several insurance source systems at AEGON Hungary Insurance Ltd.[1] The ER problem comes into sight during the construction of a client data mart over legacy systems that remained independent of each other for operational reasons during merges and ownership changes.

Client records may consist of attributes, both of persons (birth data, tax and social security numbers, postal address, etc.) and of organizations (client ID, contract number). Attribute values are often missing or erroneous, and some attributes change in time (name, postal address).

In addition to finding records of clients, another useful ER task is to find household entities. The use of households enable efficient marketing and a better knowledge of clients. The task requires however more complex match strategies, using for example postal addresses or phone numbers to deduce relationships between clients. Furthermore, household entities contain more records in average than usual ER subjects: usually more than one client belongs to a household.

Match between records might be based on pairwise equality of one or more attributes. There are cases, however, when the definition of match is more complex. Some of the attributes might refer to the same concept with possible cross-matches (e.g. home address and postal address, general phone number and mobile phone, or a name and a maiden name). There might be frequent, dummy phone numbers or incomplete addresses that produce false merges unless we can use some filtering, for instance, based on frequencies in a probabilistic model. Some records might be ambiguous and belong to more than one entity at the same time. Some of the overlapping entities might be merged, some others might be not, based on the desired match logic of the actual entity resolution task.

Figure 1 depicts such an overlapping situation. Nodes represent data records, containing customer names and addresses with edges showing the match relationship. Sets correspond to discovered households as entities. The two examples demonstrate that the match logic in such a case must be more explicitly defined and is not implied by the pairwise matches alone: the entities on the left can be merged while on the right not.

[1] AEGON Hungary has been a member of the AEGON Group since 1992, one of the world's largest life insurance and pension groups, and a strong provider of investment products.

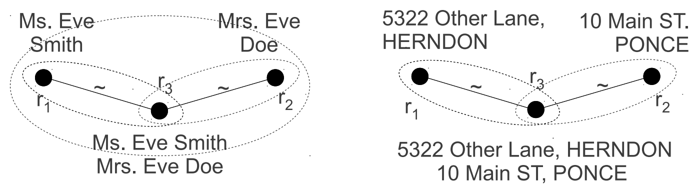

Fig. 1. An example of overlapping entities

In the first case the married name is different from the maiden name. Record r_1 contains maiden name only, while r_2 the current name and r_3 both names. Other attributes might also be equal, giving enough confidence for match except between r_1 and r_2 due to the lack of a common name. In this case, we can conclude that these three records belong to the same person and hence, the same household, so the records can be unified as a single entity.

In the second case we see the addresses of the same person with the address of the parents in r_1 and of the husband in r_2. We have two overlapping households: one with the parents and another, distinct one with the husband. In such a case, the three records can not be unified.

This simple match pattern demonstrates the motivation behind considering the entities as record sets instead of single representative records and the match relation to be defined over the sets instead of record pairs only. More complex match logic can be treated in this way, as well. In the following, we give a more general model, which can serve as a basis for different entity resolution approaches, one of them being the reachable subset model we introduce later.

3 Problem Formulation

3.1 The General Model

Let a set of **records** be $R = \{r_1, r_2, ...r_m\}$, where each r_j is described by its k **attribute values** $a_{r_j1}, \ldots, a_{r_jk}$ such as ID, name, address etc. Each record has the same k number attributes. Some attribute values may be missing, e.g. we may not know the e-mail address of customer j, that we denote by $a_{r_j\ell} = \emptyset$ (NULL value).

More formally, for a fixed k and for each $i \in [1..k]$ the ith **attribute** is a function $a_{\cdot i} : R \to DOM_i^* = DOM_i \cup \{\emptyset\}$, where DOM_i can be any set as the attribute domain. $a_{\cdot i}(r)$ is denoted as a_{ri} for simplicity.

For a general formulation of our problem we introduce a partial algebra with a binary merge operation $\langle\cdot,\cdot\rangle$ over R and a binary match relation $\sim$:

$$\mathcal{E} = (E, \langle\cdot,\cdot\rangle, \sim)$$

The base set E of this partial algebra is called the set of **entities**. The algebra is generated by a subset $R \subseteq E$ called the record set R.

The match relation $\sim$ is a general construct on entities, and might be defined arbitrarily. In practice it is defined according to the actual entity resolution task, and is based on or related to the attributes of records. We introduce features as a practical way to define entity matching in Section 4.

The base set E is generated by R, i.e. we may define

$$E_0 = R, \quad E_i = E_{i-1} \cup \{\langle x, y\rangle | x, y \in E_{i-1} \wedge x \sim y\}, \quad E = \bigcup_{i=0}^{\infty} E_i.$$

Let the **entity resolution** $ER(\mathcal{E})$ be the null elements of E, i.e. the set of all "maximal" elements of E that cannot be extended by merging matching elements:

$$ER(\mathcal{E}) = \{x \in E \mid \forall y \in E : (y \sim x \Rightarrow \langle y, x\rangle = x) \wedge (x \sim y \Rightarrow \langle x, y\rangle = x)\}$$

The base set E is not necessarily finite and maximal elements may also be generated by applying the $\langle , \rangle$ operator infinitely many times. This may happen if we concatenate values of the attributes without removing duplicates, growing attribute values longer and longer.

As another example, E may not contain null elements at all, i.e. there may be no maximal, non-extendible entities in the data. A (cyclic) group is clearly an algebraic example; the corresponding ER task keeps the most recently added attribute value a_1 of x_1 and a_2 of x_2 when defining $\langle , \rangle$. In this case

$$\langle \ldots \langle\langle\langle x_1, x_2\rangle, x_1\rangle, x_2\rangle \ldots x_i\rangle \tag{1}$$

is equal to the last element $x_i = x_1$ or x_2 of the expression.

Different algebras may result in different models of entity resolution. For instance, the model used by [2] corresponds to the setting $E \subseteq \times_{i=1}^{k} DOM_i^*$, so that the merge operation maps to single, unified representative records. It also introduces the Swoosh algorithm and the so-called ICAR conditions for the merge operation and match relation to make sure the algorithm works correctly.

The definition for ER above, as shown by the examples, is too general for practice. Next we introduce further constraints to reduce the complexity of the general ER problem.

3.2 The ICAR Conditions and Their Limitation

The ICAR conditions of [2] are considered a standard definition of the entity resolution problem. After giving the definition we will show that the fourth representativity condition is too strong and introduces inflexibility for practical applications, especially if entity boundaries are not sharp and may overlap.

Idempotence: For all r, $r \sim r$ and $\langle r, r\rangle = r$. A record always matches itself, and merging it with itself still yields the same record.

Commutativity: For all r_1, r_2, $r_1 \sim r_2$ iff $r_2 \sim r_1$, and if $r_1 \sim r_2$, then $\langle r_1, r_2\rangle = \langle r2, r1\rangle$.

(Weak) Associativity: For all $r_1, r_3 \in R$ and $X \in E$ such that $\langle\langle r_1, X\rangle, r_3\rangle$ and $\langle r_1, \langle X, r_3\rangle\rangle$ exist, $\langle\langle r_1, X\rangle, r_3\rangle = \langle r_1, \langle X, r_3\rangle\rangle$. This constraint is required for records X only in [2].

Representativity: If $r_3 = \langle r1, r2\rangle$, then for any r_4 such that $r_1 \sim r4$, we also have $r_3 \sim r_4$.

Representativity is too strong in that it does not allow overlapping entities. In the example of households, we may however have a household consisting of a person with parent and another distinct one with spouse, a practically important case of overlapping entities.

Although the original paper [2] defines the third property as associativity, it is in fact weaker than the usual definition that also requires the existence of $\langle r_1, \langle r_2, r_3\rangle\rangle$, whenever $\langle\langle r_1, r_2\rangle, r_3\rangle$ exist. The stronger version of associativity does not hold in practice either, since r_1 may share an attribute with r_3 but may be distinct from r_2. In the example of Fig. 1, r_1 may not be similar to r_2 while $\langle\langle r_1, r_2\rangle, r_3\rangle$ may exist and correspond to the person as an entity.

Also note that Idempotence and Commutativity implies the reflexivity and symmetry of the $\sim$ relation. Note that if associativity is further strengthened to imply transitivity, the match relation becomes an equivalence of the records.

Instead of associativity and as a weaker version of representativity, a better motivated definition that can be easily checked over the example of Fig. 1 is

$$\langle X, r_2\rangle \sim r_3 \text{ implies either } \begin{array}{l} X \sim r_2 \text{ and } \langle\langle X, r_2\rangle, r_3\rangle = \langle X, \langle r_2, r_3\rangle\rangle, \text{ or} \\ X \sim r_3 \text{ and } \langle\langle X, r_2\rangle, r_3\rangle = \langle r_2, \langle X, r_3\rangle\rangle. \end{array} \tag{2}$$

Under the additional constraints in the next section, the definition in (3) is equivalent to weak associativity. For this reason, we give no motivation for weak associativity, simply use as a notion from [2].

3.3 The Accessible Subset Model

For our algorithms, we introduce the accessible subset model as an alternative approach, which allows us to relax some of the ICAR conditions. Our model will have a similar but weaker implication to form representative merged elements: as in ICAR, the merge operation is simply the union of record sets. The concept below is hence is a weaker version of the representativity from the ICAR properties in [2].

For $X \in E$, let rank (X) be the minimum number of generator elements R, with multiplicities, needed to generate X. We define the *accessibility* property similar to [8] as follows:

$$\forall X \in E \exists r \in R, Y \in E \text{ with } \mathrm{rank}(Y) < \mathrm{rank}(Y) \text{ such that } X = \langle y, r\rangle. \tag{3}$$

We extend the absorption property

$$\langle\langle r_1, r_2\rangle, r_1\rangle = \langle r_1, r_2\rangle \tag{4}$$

and define **strong absorptivity**: if two elements of E are generated by the same set of records R, then they are equal.

By the above requirements, the elements E of the partial algebra becomes a partial lattice of records and henceforth we may simply consider subsets of R instead of general expressions generated by R:

Theorem 1. *If $(E, \langle\cdot,\cdot\rangle, \sim)$ is idempotent, commutative, weak associative, accessible and strong absorptive, then E is in bijection with 2^R (subsets of R) and if $\langle\cdot,\cdot\rangle$ exists, then it is equal to the set union.*

Proof. We only have to show that every $X \in E$ can be generated as

$$\langle \dots \langle\langle r_{i_1}, r_{i_2}\rangle, r_{i_3}\rangle \dots, r_{i_k}\rangle$$

using every generator at most once. Let us take the lowest rank counterexample X and use reachability to get $X = \langle Y, r\rangle$. If r appears in the description of Y using every generator at most once, then $X = Y$ by absorption and hence X is not a counterexample. Otherwise $\langle Y, r\rangle$ is the required form of X.

Absorptivity is a natural constraint that basically requires the attributes of a merged record be formed as the union of the original values. Note that it is easy to give examples with no absorption, for example the one in Section 3.1 that always keeps the attribute of the most recently merged record only. It remains an open question to find the weakest possible definition of absorptivity for Theorem 1 to hold. The condition in equation (4) is insufficient since if we merge four records in two different order, their equality will not be guaranteed.

Reachability is a frequent requirement for set systems. It is easy to construct counterexamples when the confidence of two entities belonging to the same household reaches the threshold to unify them only after several duplicates of each entity are merged by correcting and filling their attributes. However our algorithm is not capable of solving the general problem with no reachability. Here additional, weaker constraints may be necessary to extend our method.

3.4 Entity Closure: Towards an Efficient Algorithm

Towards solving the general entity resolution problem with the constraints as in Theorem 1, we introduce the concept of entity closure by basically imposing an additional weak transitivity constraint on the match relation.

The **entity closure** for an entity resolution problem formulated by $\mathcal{E} = (2^R, \cup, \sim)$ is defined by $\mathcal{E}^* = (2^R, \cup, \sim^*)$ where $\sim^*$ is the transitive closure of $\sim$. The entity resolution for the closure is an upper bound for the solution of the original entity resolution problem and can be generated by computing the maximal connected components of the graph defined by $(R, \sim|_{R\times R})$.

Given the entity closure, the solution of the original entity resolution problem can be reached by post-processing, i.e. by separating larger entities. We will first compute the transitive closure in an efficient scalable distributed way that provides an upper bound for non-transitive cases for post-processing. The rationale of the algorithm is that in real data, most matches have minor number of non-transitive instances, but some of the entities in the closure have to be split, e.g. when an ambiguous record belongs to more than one entities. Computing by closure followed by post-processing can be done efficiently, provided the entities in the closure are small.

4 Features

The entity match relation can be defined in many ways. In its simplest form, we may match entities with identical attribute values. For example, two client entities can be merged if we find two identical social security numbers or id card numbers. In such a case, match between record sets can be generated from match between record pairs: If two records can be connected through pairwise matches, then these two records belong to the same entity. Two entities match, if they have a matching record pair.

As shown in the example of Fig. 1, we may not necessary want to merge entities with just a few equal attributes but instead assign confidence thresholds that may result in overlapping entities. In this case the match relation generated by the record pairs forms an upper bound of the original match relation as in Section 3.4.

Next we give a formal but practical model to express a match relation using features that suits our distributed algorithm well. We distinguish three aspects of a feature. These three aspects can be independently formulated, and can be built around domain knowledge.

First, we isolate properties of entities based on attributes of records (eg. the birth date and maiden name pair of a client). Let a **feature** be an $E \to X$ function f, where X is an arbitrary set. Elements of X represent discriminating and independent properties of an entity, and can be for example numbers, strings or any data structures (eg. arrays of strings).

Note that the value of a feature can be a set or a single value, e.g. the 'name' feature may keep both current name and maiden name of a person record if they differ, resulting a 2-element set of feature values; or we can define it as a single value by concatenation of the two strings using a separator; or we can keep only one of the two. The situation is similar for non-singleton entities arising by merging records. The definition of feature f can be extended simply as the set union: $f(x \cup y) = f(x) \cup f(y)$. Other feature definitions may involve minimum or maximum, e.g. $f(x \cup y) = min\{f(x), f(y)\}$.

Second, let a **feature-based match function** be an $X \times X \to \{\text{true}, \text{false}\}$ partial function, where X is a set of feature values. It must be reflexive and symmetric but not necessarily transitive. We denote the match function of a given f feature as $\sim_f$. The match function is defined only for the values the feature maps to. If we use multiple features, the unified feature-based match function is defined as a disjunction. For some feature set $f_1, ...f_k$ and entities $e, e' \in E$:

$$e \sim_{f_1,...,f_k} e' \Leftrightarrow e \sim_{f_1} e' \vee e \sim_{f_2} e' \vee ... \vee e \sim_{f_k} e'.$$

Algorithm 1. Feature-based ER by Map-Reduce

input: Entity set E over a distributed file system.
output: $E' = ER(E)$

for all features f_i **do**
  sort all $r \in E$ records by representative values $rep_i(r)$
  for all representative values $rep_i(r)$ of $r \in E$ **do**
    for all pairs of records r, r' with $rep_i(r) = rep_i(r')$ **do**
      if $f_i(r) \sim_i f_i(r')$ **then**
        write $(ID(r), ID(r'))$ to graph G
Map-Reduce connected components(G)
sort records by component ID and merge groups of identical ID

The third aspect of a feature is indexability. To ensure the efficient computation of feature-based matches, we expect features and their match functions to be indexable. Let the **feature mapping function** of a feature f be a partial function $rep_f : X \to 2^O$ with some ordered set O. The mapping produces representative elements used to index the entities, to produce match candidates.

The goal is to construct feature mappings that produce identical elements for matching entities by that particular feature. For **indexable features** a feature mapping can be constructed so that all entity e' matching a given e by $\sim_f$ can be found through the equality of representative values:

$$\forall e, e' \in E, e \sim_f e' \Rightarrow \exists o \in O : o \in rep_f(f(e)) \wedge o \in rep_f(f(e')).$$

Constructing indexable feature mappings is not always easy and may involve similarity indexes. In a similarity index, entities may have many representative values. For example, string similarity can be indexed by n-grams, or by the words they contain, based on the given similarity function.

5 Distributed Algorithm

Next we modify our existing distributed algorithm published in [37]. It is based on Hadoop [42], an open source implementation of the Map-Reduce framework [16]. The following version solves the entity closure problem based on indexable features.

The algorithm works in two rounds. The first round, Algorithm 1, iterates through all features. For each, it sorts attribute values and records all potential matches in a graph file. Then the connected component Algorithm 2 is called that assigns a component ID to all records. Finally, the last line of the main algorithm merges all records with the same ID. In this step, additional split heuristics can be implemented to undo some of the unnecessary merges if necessary.

In Algorithm 1 we assign IDs to records as follows. If there are entities that consist of more than one record at start, we split it into two records, both with the same ID.

Algorithm 2. Map-Reduce connected components

input: Graph G of record IDs.
output: Component ID for all record IDs.

1: sort G to form sequences $S_i = \{i,\ ID_i,\ \text{list of edges } (i,j)\}$
2: change = true
3: **while** change = true **do**
4:     change = false
5:     **Map:**
6:     **for all** IDs i **do**
7:         **for all** IDs j with $(i,j) \in S_i$ **do**
8:             emit ID_i to reducer j
9:         emit entire S_i to reducer i
10:     **Reduce:**
11:     **for all** reducers j **do**
12:         ID' = min of all ID values received
13:         **if** $ID' < ID_j$ **then**
14:             change = true
15:             replace ID_j by ID in S_j
16:         write S_j

We describe the connected component Algorithm 2 in detail. The algorithm implements the matrix multiplication based all-pairs reachability algorithm of [15, Section 25] in a way similar to [25]. Two ingredients are the reduction of the problem to iterated matrix multiplication with a modified associative operation and the implementation of the matrix operation over Hadoop. For the first, let us replace addition by the minimum function and let

$$ID_j = min(ID_j, min_{i:(ij) is\ an\ edge}(ID_i)). \tag{5}$$

In iteration s, this method selects the minimum value in the s step neighborhood of every record. If we record the fact that some ID_j decrease in an iteration then we can terminate if there is no change.

Finally we show how to compute the matrix-vector multiplication type step of 5 by Map-Reduce. Starting at line 5, mapper i sends its current ID to reducer j for all edges ij in the graph to prepare the data needed to compute 5. In addition, reducer j starting in line 10 must write data S_j suitable for the next matrix-vector multiplication iteration. In addition to ID_j, this S_j must contain the edges out of record j. For this purpose, mapper i sends its entire data S_i to reducer i, completing the description of the algorithm.

The running time of the algorithm is $O(\ell(n \log n)/t)$ for ℓ features over t servers and $O(sn/t)$ for connected components over t servers where s is the size of the largest component.

6 Examples of Indexable Features

Within the framework of features, a variety of match relations can be formulated, starting with the simplest equality based match towards more complex match relations of more than one attributes, similarity-based heuristics or probabilistic decisions. In the following we show how match relations can be developed for the given entities and business problem that enable efficient indexing and algorithm scalability.

Equality-based Match Functions. The simplest way to define features is to rely on attributes with equality. In this case attribute values can be used as representative values too, enabling efficient candidate generation: all candidate pairs match, and all matching pairs will be candidates.

Match Functions with Multiple Attributes. Combining multiple attributes to form a feature is an obvious next step. The simplest case is to concatenate the two attribute values, and then treat the concatenation as a simple attribute, used both for match functions and for feature mappings. Arbitrary features and match relations can be however defined freely on the attributes. The only restriction is that feature mappings have to be designed carefully not to miss potential record matches.

Note that sets of attributes may represent the same concept and hence form a single feature. For example, it is a common case to store multiple phone numbers for clients – a general phone number, a cell and a home phone. A common phone feature does not distinguish between these attributes, equality of any phone numbers may mean an entity match. Such attributes can be represented by and mapped to themselves, resulting in more than one representative element for one record.

Probabilistic Match Functions. ER tasks are always interpreted in an uncertain environment, even if we define exact rules. The main ER problem arises from the fact that observations of real world entities are erroneous and vague in some sense. Therefore, entity resolution based on exact rules is not flexible enough; probabilistic models for these uncertain statements are preferred.

Besides the plain database of entity records, a-priori knowledge may be available to improve ER quality. For example, we may have information on distributions of attribute values or cardinalities of entity groups. Similarity measures shall consider statistical properties of the given entity set: two records having the name "John Smith" match with lower probability than two having "Dunstan Everitt". For an other example, the phone number "+36 20 222 2222" is valid, and two records having these numbers can be matched. However, if we look at the frequency of this particular number in our database, it turns out that this number is outstandingly common. It is used when a phone number has to be given, but the client does not provide it. Using the frequency information these matches can be avoided. The external information should be incorporated into the ER model or represented as extra attributes.

In the following we inspect events where we take two records, r_1 and r_2 of some hidden real-world entities e_1 and e_2. We think of record matching as assigning a probability to the hypothesis "r_1 and r_2 describe the same entity ($e_1 = e_2$)". We incorporate in our model external knowledge as prior probabilities, and expert knowledge as probability estimations.

For a given attribute a, let records r_1, r_2 take the values a_1 and a_2 – we denote these events as a_1 and a_2 for short. Our main goal is to provide a sound estimation for $P(e_1 = e_2 \mid a_1 \cap a_2)$. We expect events a_1 and a_2 to be independent. Using Bayes' formula,

$$P(e_1 = e_2 \mid a_1 \cap a_2) = \frac{P(a_1 \cap a_2 \mid e_1 = e_2)P(e_1 = e_2)}{P(a_1 \cap a_2)} = \\ = \frac{P(a_1 \cap a_2 \mid e_1 = e_2)P(e_1 = e_2)}{P(a_1)P(a_2)}.$$

In Algorithm 1 the estimated probability above is used to decide matches: All pairs above a given threshold are matching pairs.

$P(e_1 = e_2)$ is a constant value, and can be estimated by the help of domain knowledge. For example, if we are looking for clients in a database of client records, and the domain expert states that the company have n clients all having approximately the same number of records, then this probability is $1/n^2$.

We have prior knowledge for $P(a_1)$ and $P(a_2)$ too: a good estimation can be given using the distribution of attribute values. For example, the given name "John" is be more probable than "Everitt".

The most interesting part is $P(a_1 \cap a_2 \mid e_1 = e_2)$. That is, if we assume that the two records belong to the same real world entity, then what is the probability of having the given attribute values. For strict equality-based matching, we can say that this probability is zero if the attributes are not equal. More advanced heuristics can also be built, for example, it is a common case for postal addresses to build decision trees based on the parts of the address. Or, we can incorporate the probability of typos. If we have training data, then classifiers can also be trained to learn the probabilities.

Probabilistic Match Functions with Multiple Attributes. Probabilistic match can be extended to handle more than one attribute. Again, we take two records, r_1 and r_2 of some hidden e_1 and e_2 real-world entities. Now we use not only an a, but a b attribute with values b_1 and b_2 (with the same notion for the corresponding events). Using the Bayes' formula and supposing the independence of a and b events,

$$P(e_1 = e_2 \mid a_1 \cap a_2 \cap b_1 \cap b_2) = \frac{P(a_1 \cap a_2 \cap b_1 \cap b_2 \mid e_1 = e_2)P(e_1 = e_2)}{P(a_1 \cap a_2 \cap b_1 \cap b_2)} = \\ \frac{P(a_1 \cap a_2 \mid e_1 = e_2)P(b_1 \cap b_2 \mid e_1 = e_2)P(e_1 = e_2)}{P(a_1)P(a_2)P(b_1)P(b_2)}.$$

Probabilistic Match Functions and Missing Values. We must consider the semantics of null ($\emptyset$) values to decide how they should be treated. In the probabilistic model we can incorporate semantic and domain knowledge about null values.

As an example, we consider two attributes as in probabilistic match with multiple attributes. It can be generalized to more attributes as well. Since the case is symmetric regarding a and b as well as r_1 and r_2, the following relevant cases can occur:

1. One attribute is null in both records: $a_1 \neq \emptyset, a_2 \neq \emptyset, b_1 = b_2 = \emptyset$
2. One attribute is null in only one of the records: $a_1 \neq \emptyset, a_2 \neq \emptyset, b_1 \neq \emptyset, b_2 = \emptyset$
3. Each attribute has a null in one of the records: $a_1 = \emptyset, a_2 \neq \emptyset, b_1 \neq \emptyset, b_2 = \emptyset$

Adapting the Bayes formula for the above cases can be based on the principle that the null value can mean any value and we have no observation (event) on that value. The most interesting Case 2 can for example be written as follows, assuming independence:

$$P(e_1 = e_2 \mid a_1 \cap a_2 \cap b_1) = \frac{P(a_1 \cap a_2 \cap b_1 \mid e_1 = e_2)P(e_1 = e_2)}{P(a_1 \cap a_2 \cap b_1)} = \frac{P(a_1 \cap a_2 \mid e_1 = e_2)P(b_1 \mid e_1 = e_2)P(e_1 = e_2)}{P(a_1)P(a_2)P(b_1)}.$$

If we take the strict equality case, i.e. the value b of the real-world entities e_1 and e_2 the data records refer to must agree if they are equal, then the probability of data record r_2 (with a null value on b_2) referring to the same entity as record r_1 is the probability of the real value b of entity e_2 behind the missing observation b_2 is the same as the observed b_1 value. Therefore, $P(b_1 \mid e_1 = e_2) = P(b_1)$ and the above formula becomes similar to the one-attribute case. This matches our common-sense assumption that having a value for an attribute b in record r_1 is irrelevant if the same attribute in r_2 is unknown.

Probabilistic Match Functions using Similarities. Estimation of $P(a_1 \cap a_2 \mid e_1 = e_2)$ in the probabilistic match model can be facilitated by similarity metrics between a_1 and a_2, for example, edit distance, Soundex or stemming and token similarities of a client name.

Without going into details, we note that it is not straightforward to construct a feature mapping for similarity-based probabilistic match functions. For edit distance, as the most popular choice in a wide spectrum of applications, n-gram indexing enables finding all matching pairs above a given similarity threshold. N-grams may result however in too many representative elements and inefficient indexing. As an other example, [45] defines mappings from string space to integer space to support similarity-based string search; these methods can be applied when building feature mappings.

7 Experiments

Experiments were performed on 15-server Linux farms containing identical dual core 3 GHz Pentium CPUs, 4 GB of main memory. Software versions were Sun

Java 1.6 and Hadoop 0.20.3. We configured Hadoop to use all available internal memory. The largest data sets do not fit in the memory of one node but still fit in the entire 15×4 GB of the cluster.

The data set is provided by AEGON Hungary Insurance Ltd. containing approximately 13 million client records. Records consist of both personal attributes (names, birth data, tax number, etc.), internal identifiers, postal addresses and phone numbers. The data set can be considered as a snapshot of clients; the newest available attributes are contained for all records if the source system supports historical data. The size of the input with a rich set of attributes is around 3 GB in a flat CSV file.

We used random sampling to obtain smaller subsets. We also used selection heuristics to influence the count of records per client and per household. For example, selecting all records for the family name 'Smith' instead of a random sample will increase the match count. We created larger data sets by replication and random permutation. In each replica, we added a version tag to all attributes so that the original structure of matches was preserved by replica but no new matches were introduced between replicas. For edit-distance-based match we shifted the codes of characters in the strings by constant values.

7.1 Features and Candidate Generation

Personal features that enable finding records of the same people were used in all settings. Postal address and phone numbers were used only for household identification. We applied equality-based match on personal attributes or on attribute combinations, with the attributes themselves used for indexing in most cases.

The following features were used:

f_p	name & maiden name, birth date, mother name
f_s	social security number
f_x	tax number
f_{c1}, f_{c2}	source system-specific codes
f_t	phone number & mobile phone number
f_t^{b1}, f_t^{b2}	phone number(s) in two different probabilistic models
f_a	full postal address
f_a^f	postal address (index) with family name
f_{a1}	postal address up to house number
f_{a1}^f	postal address up to house number (index) with family name
f_{a2}	postal address up to street and postal code

Multi-attribute features are indexed by string concatenation of attributes. Probabilistic phone number features are for avoiding false merges by frequent dummy phone numbers. Combined features with postal address and family name are indexed by postal address. Since each record contains two postal address attributes, these are merged into one address-related feature, just as phone and mobile phone numbers into another.

For experiments 1–9, we used the following feature combinations:

Features →	personal data					phone			address-related				
Case ↓	f_p	f_s	f_x	f_{c1}	f_{c2}	f_t	f_t^{b1}	f_t^{b2}	f_a	f_a^f	f_{a1}	f_{a1}^f	f_{a2}
1	•	•	•	•	•								
2	•	•	•	•	•	•							
3	•	•	•	•	•		•						
4	•	•	•	•	•			•					
5	•	•	•	•	•	•			•				
6	•	•	•	•	•	•				•			
7	•	•	•	•	•	•					•		
8	•	•	•	•	•	•						•	
9	•	•	•	•	•	•							•

The resolved entities correspond to persons (clients) in cases 1–4, and households in cases 5–9.

7.2 Scalability

Previous work that we are aware of assume only 10 to 100 thousand records as input. To give an example, in [27] a distributed algorithm is described with similarity-based match, tested on 114 thousand records. Algorithms of [39] were tested closest to our database size with 10 million records.

As also shown in [37], our MapReduce ER algorithm is scalable to hundred millions of records. Figure 2 demonstrates the running times on the client data set, solving the ER problem of clients. We applied the 1st feature combination of Section 7.1, not using the address and phone number attributes, and using strict attribute equality-based feature match.

Figure 2 also provides an inaccurate, but useful overview of several ER algorithms on client data, namely the following:

- Java-F-Swoosh: a Java implementation of the best previously known generic entity resolution algorithm (F-Swoosh [2]),
- DB-GER: our best relational entity resolution algorithm, based on a commercial relational database (see [35]),
- index-ER-BDB: our best efficient indexing algorithm built on Java and Berkeley DB (see [36]),
- MapReduce: our best distributed ER algorithm using 15 computer nodes and Hadoop implementation.

We implemented these conceptually different ER algorithms in more or less different environments, therefore the comparison is not entirely correct. Nevertheless, the superiority of the MapReduce algorithm is clearly visible.

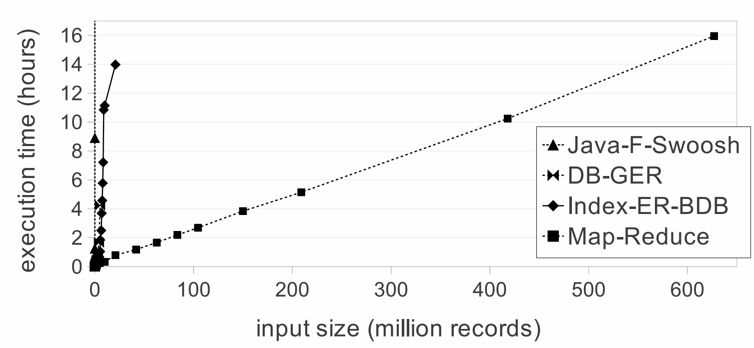

Fig. 2. Scalability of different entity resolution algorithms

7.3 Features and Entity Sizes

To find all households contained in the data set we used the different feature combinations of Section 7.1. Figure 3 shows how execution time increases with average entity size. The algorithm still seems to be useful when relatively large entities are generated, containing 5-6 records in average, which is much more than our motivating problem of households require.

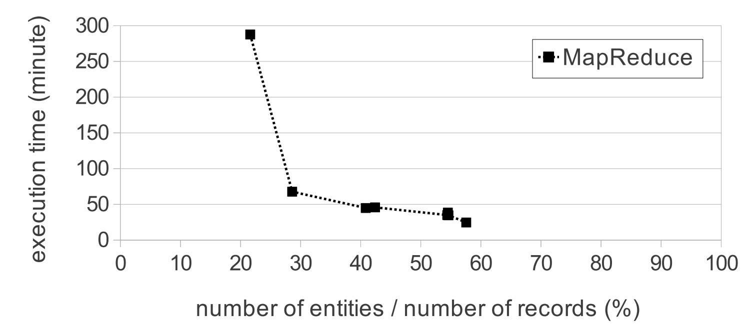

Fig. 3. Running time of the distributed algorithm with different entity sizes

Figure 4 shows the entity size distribution for one of the cases, when households are generated so that the last part of the address (the unit number inside a house) is omitted. This case corresponds the longest execution process shown in Fig. 3. The input contains 13.3M records and the total number of output entities is about 2.9M.

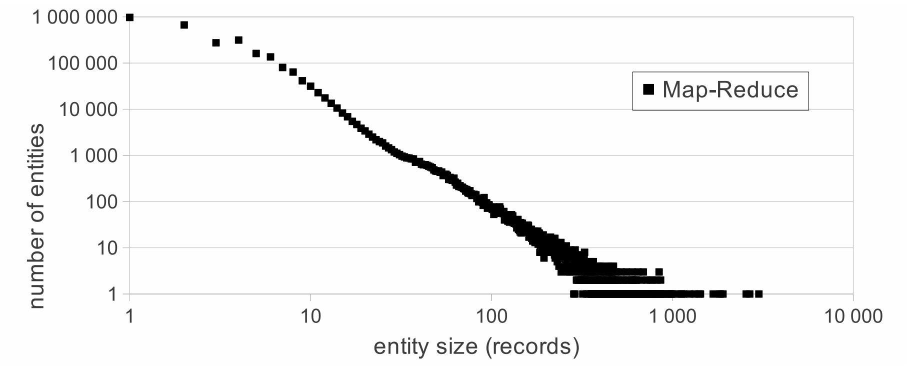

Fig. 4. Distribution of the number of entities and their sizes (households based on partial address)

8 Conclusions and Future Work

In this paper we introduced a generalized theoretical approach and the the accessible record subset model to formulate entity resolution problems in a flexible way. We gave a framework to define probabilistic and similarity-based match relations. We showed that resolution of large entities can be efficiently solved by a scalable distributed algorithm. We demonstrated the usability of our methods by identifying households of insurance client records.

There are several important areas of research to pursue. One issue is how to switch from similarities to probabilities. Several similarity metrics and indexing methods for similarity searches can be found in literature. The methods working efficiently with our distributed algorithm and also feasible for domain experts have yet to be found.

Another issue is the potential use of probability values. Overlapping or density-based clustering of records may describe the entities of the given business problem better. Therefore, it would be profitable to use the probability values generated when matching record pairs, as weights for the edges of the record graph.

Acknowledgments. To András Vereczki and Zoltán Hans as domain experts on the AEGON Hungary side for discussion on the problem formulation and clarification of the user requirements.

References

1. Benjelloun, O., Garcia-Molina, H., Gong, H., Kawai, H., Larson, T.E., Menestrina, D., Thavisomboon, S.: D-Swoosh: A family of algorithms for generic, distributed entity resolution. In: Proc. 27th Int. Conf. on Distributed Computing Systems (2007)

2. Benjelloun, O., Garcia-Molina, H., Menestrina, D., Su, Q., Whang, S.E., Widom, J.: Swoosh: a generic approach to entity resolution. VLDB J. 18(1), 255–276 (2009)
3. Bhattacharya, I., Getoor, L.: A Latent dirichlet model for unsupervised entity resolution. In: SIAM International Conference on Data Mining, pp. 47–58 (2006)
4. Bhattacharya, I., Getoor, L.: Collective entity resolution in relational data. ACM Trans. Knowl. Discov. Data 1(1), 5 (2007)
5. Bhattacharya, I., Getoor, L., Licamele, L.: Query-time entity resolution. In: Proc. 12th ACM SIGKDD, pp. 529–534 (2006)
6. Bhattacharya, I., Godbole, S., Joshi, S.: Structured entity identification and document categorization: two tasks with one joint model. In: KDD 2008: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 25–33. ACM, New York (2008)
7. Bilenko, M., Mooney, R.: Adaptive duplicate detection using learnable string similarity measures. In: Proc. 9th ACM SIGKDD, pp. 39–48 (2003)
8. Boley, M., Horváth, T., Poigné, A., Wrobel, S.: Efficient Closed Pattern Mining in Strongly Accessible Set Systems (Extended Abstract). In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) PKDD 2007. LNCS (LNAI), vol. 4702, pp. 382–389. Springer, Heidelberg (2007)
9. Chaudhuri, S., Sarma, A.D., Ganti, V., Kaushik, R.: Leveraging aggregate constraints for deduplication. In: SIGMOD 2007, pp. 437–448. ACM (2007)
10. Chen, S., Borthwick, A., Carvalho, V.R.: The case for cost-sensitive and easy-to-interpret models in industrial record linkage. In: 9th International Workshop on Quality in Databases (2011)
11. Christen, P.: Automatic record linkage using seeded nearest neighbour and support vector machine classification. In: KDD 2008, pp. 151–159. ACM (2008)
12. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. In: IEEE TKDE preprint (2011)
13. Christen, P., Churches, T., Hegland, M.: Febrl – A Parallel Open Source Data Linkage System. In: Dai, H., Srikant, R., Zhang, C. (eds.) PAKDD 2004. LNCS (LNAI), vol. 3056, pp. 638–647. Springer, Heidelberg (2004)
14. Christen, P., Gayler, R., Hawking, D.: Similarity-aware indexing for real-time entity resolution. In: CIKM 2009, pp. 1565–1568. ACM (2009)
15. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press (2001)
16. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
17. Elmagarmid, A., Ipeirotis, P., Verykios, V.: Duplicate Record Detection: A Survey. IEEE TKDE, 1–16 (2007)
18. Fellegi, I., Sunter, A.: A theory for record linkage. Journal of the American Statistical Association 64(328), 1183–1210 (1969)
19. Getoor, L., Diehl, C.: Link mining: a survey. ACM SIGKDD Explorations Newsletter 7(2), 3–12 (2005)
20. Gravano, L., Ipeirotis, P., Jagadish, H., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate string joins in a database (almost) for free. In: VLDB, pp. 491–500 (2001)
21. Guo, S., Dong, X.L., Srivastava, D., Zajac, R.: Record linkage with uniqueness constraints and erroneous values. Proc. VLDB Endow. 3, 417–428 (2010)
22. Hall, R., Sutton, C., McCallum, A.: Unsupervised deduplication using cross-field dependencies. In: KDD 2008, pp. 310–317. ACM (2008)

23. Han, H., Xu, W., Zha, H., Giles, C.: A hierarchical naive Bayes mixture model for name disambiguation in author citations. In: Proc. 2005 ACM Symposium on Applied Computing, pp. 1065–1069 (2005)
24. Hernández, M., Stolfo, S.: Real-world data is dirty: Data cleansing and the merge/purge problem. Data Mining and Knowledge Discovery 2(1), 9–37 (1998)
25. Kang, U., Tsourakakis, C., Faloutsos, C.: Pegasus: A peta-scale graph mining system implementation and observations. In: ICDM, pp. 229–238. IEEE (2009)
26. Kim, H.-S., Lee, D.: Parallel linkage. In: CIKM 2007. ACM (2007)
27. Kirsten, T., Kolb, L., Hartung, M., Gross, A., Köpcke, H., Rahm, E.: Data partitioning for parallel entity matching. Computing Research Repository (2010)
28. Köpcke, H., Rahm, E.: Training selection for tuning entity matching. In: QDB/MUD, pp. 3–12 (2008)
29. Köpcke, H., Rahm, E.: Frameworks for entity matching: A comparison. Data Knowl. Eng. 69, 197–210 (2010)
30. Köpcke, H., Thor, A., Rahm, E.: Evaluation of entity resolution approaches on real-world match problems. Proc. VLDB Endow. 3, 484–493 (2010)
31. McCarthy, J., Lehnert, W.: Using decision trees for coreference resolution. In: Proc. 14th Int. Conf. on Artificial Intelligence, pp. 1050–1055 (1995)
32. Menestrina, D., Benjelloun, O., Garcia-Molina, H.: Generic entity resolution with data confidences. In: CleanDB Workshop, pp. 25–32 (2006)
33. Menestrina, D., Whang, S.E., Garcia-Molina, H.: Evaluating entity resolution results. Proc. VLDB Endow. 3, 208–219 (2010)
34. Sarawagi, S., Bhamidipaty, A.: Interactive deduplication using active learning. In: SIGKDD, pp. 269–278 (2002)
35. Sidló, C.I.: Generic Entity Resolution in Relational Databases. In: Grundspenkis, J., Morzy, T., Vossen, G. (eds.) ADBIS 2009. LNCS, vol. 5739, pp. 59–73. Springer, Heidelberg (2009)
36. Sidló, C.I.: Entity resolution with heavy indexing. In: Proc. ADBIS, CEUR Workshop Proceedings (2011)
37. Sidló, C.I., Garzó, A., Molnár, A., Benczúr, A.A.: Infrastructures and bounds for distributed entity resolution. In: 9th International Workshop on Quality in Databases (2011)
38. Talburt, J.R.: Entity Resolution and Information Quality, 1st edn. Morgan Kaufmann (2010)
39. Weis, M., Naumann, F., Jehle, U., Lufter, J., Schuster, H.: Industry-scale duplicate detection. Proc. of the VLDB Endow. 1(2), 1253–1264 (2008)
40. Whang, S.E., Garcia-Molina, H.: Entity resolution with evolving rules. Proc. VLDB Endow. 3, 1326–1337 (2010)
41. Whang, S.E., Menestrina, D., Koutrika, G., Theobald, M., Garcia-Molina, H.: Entity resolution with iterative blocking. In: Proc. 35th Int. Conf. on Management of Data, pp. 219–232. ACM (2009)
42. White, T.: Hadoop: The Definitive Guide. Yahoo Press (2010)
43. Wick, M.L., Rohanimanesh, K., Schultz, K., McCallum, A.: A unified approach for schema matching, coreference and canonicalization. In: KDD 2008, pp. 722–730. ACM (2008)
44. Yakout, M., Elmagarmid, A.K., Elmeleegy, H., Ouzzani, M., Qi, A.: Behavior based record linkage. Proc. VLDB Endow. 3, 439–448 (2010)
45. Zhang, Z., Hadjieleftheriou, M., Ooi, B.C., Srivastava, D.: Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In: SIGMOD, pp. 915–926. ACM (2010)