Eliminating Fuzzy Duplicates in Data Warehouses

Rohit Ananthakrishna[1]
Cornell University
rohit@cs.cornell.edu

Surajit Chaudhuri Venkatesh Ganti
Microsoft Research
{surajitc, vganti}@microsoft.com

Abstract

The *duplicate elimination* problem of detecting multiple tuples, which describe the same real world entity, is an important data cleaning problem. Previous domain independent solutions to this problem relied on standard textual similarity functions (e.g., edit distance, cosine metric) between multi-attribute tuples. However, such approaches result in large numbers of false positives if we want to identify domain-specific abbreviations and conventions. In this paper, we develop an algorithm for eliminating duplicates in dimensional tables in a data warehouse, which are usually associated with hierarchies. We exploit hierarchies to develop a high quality, scalable duplicate elimination algorithm, and evaluate it on real datasets from an operational data warehouse.

1. Introduction

Decision support analysis on data warehouses influences important business decisions; therefore, accuracy of such analysis is crucial. However, data received at the data warehouse from external sources usually contains errors: spelling mistakes, inconsistent conventions, etc. Hence, significant amount of time and money are spent on *data cleaning*, the task of detecting and correcting errors in data.

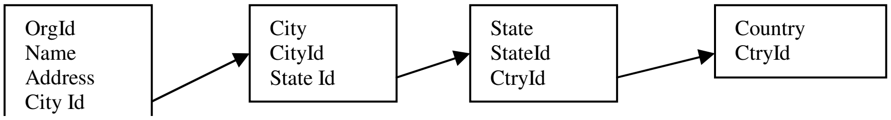
The problem of detecting and eliminating duplicated data is one of the major problems in the broad area of data cleaning and data quality [e.g., HS95, ME97, RD00]. Many times, the same logical real world entity may have multiple representations in the data warehouse. For example, when Lisa purchases products from SuperMart twice, she might be entered as two different customers—[Lisa Simpson, Seattle, WA, USA, 98025] and [Lisa Simson, Seattle, WA, United States, 98025]—due to data entry errors. Such duplicated information can significantly increase direct mailing costs because several customers like Lisa may be sent multiple catalogs. Moreover, such duplicates can cause incorrect results in analysis queries (say, the number of SuperMart customers in Seattle), and erroneous data mining models to be built. We refer to this problem of detecting and eliminating multiple distinct records representing the same real world entity as the *fuzzy duplicate elimination problem*, which is sometimes also called merge/purge, dedup, record linkage problems [e,g., HS95, ME97, FS69]. This problem is different from the standard duplicate elimination problem, say for answering "select distinct" queries, in relational database systems which considers two tuples to be duplicates if they match exactly on all attributes. However, data cleaning deals with *fuzzy duplicate elimination,* which is our focus in this paper. Henceforth, we use *duplicate elimination* to mean fuzzy duplicate elimination.

Duplicate elimination is hard because it is caused by several types of errors like typographical errors, and *equivalence errors*—different (non-unique and non-standard) representations of the same logical value. For instance, a user may enter "WA, United States" or "Wash., USA" for "WA, United States of America." Equivalence errors in product tables ("winxp pro" for "windows XP Professional") are different from those encountered in bibliographic tables ("VLDB" for "very large databases"), etc. Also, it is important to detect and clean equivalence errors because an equivalence error may result in several duplicate tuples.

The class of equivalence errors can be addressed by building sets of rules. For instance, most commercial address cleaning software packages (e.g., Trillium) use rules to detect errors in names and addresses. In this paper, we focus on domain independent duplicate elimination techniques. Domain-specific information when available complements these techniques. Previous domain-independent methods for duplicate elimination rely on textual similarity functions (e.g., edit distance or cosine metric) predicting that two tuples whose textual similarity is greater than a pre-specified similarity threshold are duplicates [FS69, KA85, Coh98, HS95, ME96]. However, using these functions to detect duplicates due to equivalence errors (say, "US" and "United States") requires that the threshold be dropped low enough, resulting in a large number of *false positives*—pairs of tuples incorrectly detected to be duplicates. For instance,

Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002

[1] Work done while visiting Microsoft Research



OrgId	Name	Address	CityId
O1	Clintstone Assoc.	#1, Lake View Blvd.	C1
O2	Compuware	#20, Main Street	C2
O3	Compuwar	#20, Main Street	C3
O4	Clintstone Associates	#1, Lake View	C4
O5	Ideology Corp.	#10, Vancouver Pl.	C5
O6	Victoria Films	#5, Victoria Av.	C6
O7	Ideology Corporation	#10, Vanc. Pl.	C7
O8	Clark Consultants Ltd.	#8, Cherry Street	C8
O9	Clark Consultants	#8, Cherr St.	C9

Organization (at Level 1)

CityId	City	StateId
C1	Joplin	S1
C2	Jopin	S2
C3	Joplin	S4
C4	Joplin	S3
C5	Victoria	S5
C6	Victoria	S6
C7	Vancouver	S5
C8	Aberdeen	S7
C9	Aberdeen	S8

City (at Level 2)

StateId	State	CtryId
S1	MO	1
S2	MO	2
S3	MO	3
S4	Missouri	3
S5	BC	4
S6	British Columbia	4
S7	Aberdeen shire	5
S8	Aberdeen	5

State (at Level 3)

CtryId	Country
1	United States of America
2	United States
3	USA
4	Canada
5	UK

Country (at Level 4)

Figure 1: An Example Customer Database

tuple pairs with values "USSR" and "United States" in the country attribute are also likely to be declared duplicates if we were to detect "US" and "United States" as duplicates using textual similarity.

In this paper, we exploit dimensional hierarchies typically associated with dimensional tables in data warehouses to develop an efficient, scalable, duplicate elimination algorithm called Delphi,[2] which significantly reduces the number of false positives without missing out on detecting duplicates. We rely on hierarchies to detect an important class of equivalence errors in each relation, and to significantly reduce the number of false positives.

For example, Figure 1 describes the schema maintaining the Customer information in a typical company selling products or services. The dimensional hierarchy here consists of four relations—Organization, City, State, and Country relations—connected by key—foreign key relationships (also called referential links). We say that the Organization and the Country relations are the *bottom* and the *top* relations, respectively. Consider the tuples USA and United States in the Country relation in Figure 1. The state attribute value "MO" appears in tuples in the State relation joining with countries USA and United States, whereas most state values occur with only one Country tuple. That is, USA and United States *co-occur* through the state MO. In general, country tuples are associated with sets of state values. The degree of overlap between sets associated with two countries is a measure of co-occurrence between them, and can be used to detect duplicates (e.g., USA and United States).

The above notion of co-occurrence can also be used for reducing the number of false positives. Consider the two countries "USA" and "UK" in Figure 1. Because they are sufficiently closer according to the edit distance function, a commonly used textual similarity function, we might (incorrectly) deduce that they are duplicates. Such problems can occur even with other textual similarity functions like the cosine metric. Using our notion of co-occurrence through the State relation, we observe that the sets—called *children sets* of USA and UK—of states {MO, Missouri} and {Aberdeen, Aberdeen shire} joining with USA and UK, respectively, are disjoint. Hence, we conclude that USA and UK are unlikely to be duplicates.

For reasons of efficiency and scalability, we want to avoid comparing all pairs of tuples in each relation of the hierarchy. Previous approaches have considered the *windowing strategy*, which sorts a relation on a key and compares all records within a sliding window on the sorted order [HS95]. However, observe that equivalence errors (e.g., UK and Great Britain) may not be adjacent to each other in standard sort orders, e.g., the lexicographical order. We exploit the dimensional hierarchy and propose a *grouping strategy*, which only compares tuples within small groups of each relation. For instance, we only compare two State tuples if they join with the same country tuple or Country tuples that are duplicates of each other. Since such groups are often much smaller than the entire relation, the grouping strategy allows us to compare pairs of tuples in each group, and yet be very efficient.

The outline of the paper is as follows. In Section 2, we discuss related work. In Section 3, we discuss key concepts and definitions. In Section 4, we describe Delphi. In Section 5, we discuss a few important issues. In Section 6, we discuss results from a thorough experimental evaluation on real datasets.

2. Related Work

Several earlier proposals exist for the problem of duplicate elimination (e.g., [FS69, KA85, HS95, ME96, ME97, Coh98]). As mentioned earlier, all these methods rely on threshold-based textual similarity functions to detect duplicates, and hence do not detect equivalence errors unless we lower thresholds sufficiently; lower thresholds result in an explosion of the number of false positives. The

[2] DELPHI: **D**uplicate **EL**imination in the **P**resence of **HI**erarchies

record linkage literature also focuses on automatically determining appropriate thresholds [FS69, KA85], but still suffers from the false positive explosion while detecting equivalence errors. Gravano et al. proposed an algorithm for approximate string joins, which in principle can be adapted to detect duplicate records [GIJ+01]. Since they use the edit distance function to measure closeness between tuples, their technique suffers from the drawbacks of strategies relying only on textual similarity functions. In this paper, we exploit hierarchies on dimensional tables to detect an important class of equivalence errors (which exhibit significant co-occurrence through other relations) without increasing the number of false positives.

Significant amount of work exists in other related aspects of data cleaning: the development of *transformational cleaning operations* [RH01, GFS+01], the detection and the correction of formatting errors in address data [BDS01], and the design of "good" business practices and process flows to prevent problems of deteriorating data quality [Pro, NR99]. Automatic detection of integrity constraints (functional dependencies and key—foreign key relationships) [MR94, KM95, HKPT98] so that they can be enforced in future to improve data quality are complementary to techniques for cleaning existing data. Because of the commercial importance of the data cleaning problem, several domain-specific industrial tools exist. Galhardas provides a nice survey of many commercial tools [Gal].

Our notion of co-occurrence between tuples is similar to that used for clustering categorical data [e.g., GKR98, GRS99, GGR99] and that for matching schema [MBR01].

3. Concepts and Definitions

A dimensional hierarchy consists of a chain of relations linked by key—foreign key dependencies. Figure 1 illustrates an example. An entity described by the hierarchy also consists of a chain of tuples (one from each relation) each of which joins with the tuple from its parent relation. For example, [<o1, Walmart, c1>, <c1, Redmond, s1>, <s1, WA, t1>, <t1, USA>] describes an organization entity where o1, c1, etc. are identifiers typically generated for maintaining referential links. For clarity in notation, we do not explicitly list identifiers in tuples unless required.

Consider two organization entities: [<Walmart>, <Redmond>, <WA>, <USA>] and [<Walmart>, <Seattle>, <WA>, <USA>] in the Customer information with a dimensional hierarchy shown in Figure 1. The corresponding pairs of tuples in the Name, State, or Country relations individually are identical. However, they are not duplicates on the City relation, and in fact this difference makes the two entities distinct. This phenomenon is characteristic of dimensional hierarchies. For example, publications with the same title may appear in the proceedings of a conference as well as in a journal; and, they are two distinct entities in the publications database. Motivated by these typical scenarios, we consider two entities in a dimensional hierarchy to be duplicates if corresponding pairs of tuples in each relation of the hierarchy either match exactly or are duplicates (according to duplicate detection functions at each level). For example, two entities in Figure 1 are duplicates if the respective pairs of Country, State, City, and Organization tuples of the two entities are duplicates. Below, we formally introduce dimensional hierarchies, definition of duplicate entities, and our duplicate detection functions.

3.1. Dimensional Hierarchies

Relations $R_1,\ldots, R_m$ with keys $K_1, \ldots, K_m$ constitute a dimensional hierarchy if and only if there is a key—foreign key relationship between R_{i-1} and R_i, ($2\leq i \leq m$). R_i is the i^{th} *level* relation in the hierarchy. R_1 and R_m are the *bottom* and the *top* relations, respectively, and R_i the *child* of R_{i+1}.

Let the *unnormalized dimension table* R be the join of $R_1,\ldots, R_m$ through the chain of key—foreign key relationships. We say that a tuple v_i in R_i *joins* with a tuple v_j in R_j if there exists a tuple v in R such that the projections of v on R_i and R_j equal v_i and v_j, respectively. Specifically, we say that v_i in R_i is *a child of* v_{i+1} in R_{i+1} if v_i joins with v_{i+1}. For example, in Figure 1, [S3, MO, 3] in the State relation is a child of [3, USA] in the Country relation. We say that a tuple combination (or a row in R) $[r_1,\ldots,r_m]$ is an *entity* if each r_i joins with r_{i+1}.

In typical dimensional tables of data warehouses, the values of key attributes $K_1,\ldots, K_m$ are artificially *generated* by the loading process before a tuple v_i is inserted into R_i. Such generated keys are not useful for fuzzily matching two tuples, and can only be used for joining tuples across relations in the hierarchy. From now on, we overload the term "tuple" to also mean only the *descriptive attribute values*—the set of attribute values not including the generated key attributes. We clarify when it is not clear from the context.

3.2. Definition of Duplicates

We now formally define our notion of duplicate entities assuming duplicate detection functions at each level. Let $f_1,\ldots,f_m$ be binary functions called *duplicate detection functions* where each f_i takes a pair of tuples in R_i, and returns 1 if they are duplicates, and -1 otherwise. Let $r=[r_1, \ldots, r_m]$ and $s=[s_1,\ldots,s_m]$ be two entities. We say that r is a *duplicate* of s if and only if $f_i(r_i, s_i)=1$ for all i in {1, …, m}. For instance, we consider the two entities [<Compuware, #20 Main Street>, <Jopin>, <MO>, <United States>] and [<Compuwar, #20 Main Street>, <Joplin>, <Missouri>, <USA>] in Figure 1 to be duplicates only if the following pairs are duplicates: “United States” and “USA” on the Country relation, “MO” and “Missouri” in the State relation, “Jopin” and “Joplin” in the City relation, and “Compuware, #20 Main Street” and “Compuwar, #20 Main Street” in the Organization relation. Observe that we can easily extend this definition to sub-entities $[r_i,\ldots,r_m]$ and $[s_i,\ldots,s_m]$.

3.3. Duplicate Detection Functions

We exploit dimensional hierarchies to measure co-occurrence among tuples for detecting equivalence errors and for reducing false positives. This is in conjunction with the textual similarity functions (like cosine metric and edit distance), which have traditionally been employed for detecting duplicates. Our final duplicate detection function is a *weighted voting* of the predictions from using co-occurrence and textual similarity functions. Intuitively, the weight of a prediction is indicative of the importance of the information used to arrive at the prediction.

We adopt the standard thresholded similarity function approach to define duplicate detection functions [HS95]. That is, if the textual (or co-occurrence) similarity between two tuples is greater than a threshold, then the two tuples are predicted to be duplicates according to textual (or co-occurrence) similarity. In this section, we assume that thresholds are known. In Section 4.3, we relax this assumption and describe automatic threshold determination. First, we introduce the notion of *set containment,* which we use to define similarity functions. We only consider textual attributes for comparing tuples, and assume default conversions from other types to text, e.g., integer zipcodes are converted to varchar.

Given a collection of sets each defined over some domain of objects, an intuitive notion of how similar a set S is to a set S’ is the fraction of S objects contained in S’. This notion of containment similarity has been effectively used to measure document similarity [BGM+97]. We extend this notion to take into account the importance of objects in distinguishing sets. For example, the set {Microsoft, incorporated} is more similar to {Microsoft, inc} than it is to {Boeing, incorporated} because the token *Microsoft* is more distinguishing than the token *incorporated.* The *IDF (inverse document frequency) value* of an object has been successfully used in the information retrieval literature to quantify the notion of importance [BYRN99]. We now formalize this intuition.

Let $\mathcal{O}$ be a set of objects. Let G be a collection of sets of objects from $\mathcal{O}$. Let B(G) be the bag of all objects contained by any set in G. The *frequency* $f_G(o)$ of an object o with respect to G is the frequency of o in B(G). The IDF value $IDF_G(o)$ with respect to G of o is $\log(\frac{|G|}{f_G(o)})$. Also, we define the *IDF value* $IDF_G(S)$ of a set S (subset of $\mathcal{O}$) to be $\sum_{s\in S} IDF_G(s)$.

Containment Metric: We define the *containment metric* $cm_G(S_1, S_2)$ with respect to G between two sets S_1 and S_2 to be the ratio of the IDF value $IDF_G(S_1 \cap S_2)$of their intersection with the IDF value $IDF_G(S_1)$ of the first set S_1.

For clarity in presentation, we drop the subscript G from the above notation when extending them to define textual and co-occurrence similarity metrics.

3.3.1. Textual Similarity Function (tcm)

We assume that each tuple v can be split into a set of tokens using a tokenization function (say, based on white spaces). Treating each tuple as a set of tokens, the *token containment metric* between v and v’ is the IDF-weighted fraction of v tokens that v’ contains.

Let $G=\{v_1, \ldots, v_n\}$ be a set of tuples from R_i. Let *TS(v)* denote the set of tokens in a tuple v. Let $B_t(G)$ be the bag (multi-set) of all tokens that occur in any tuple in G. Let *tf(t)* denote the frequency of a token t in $B_t(G)$. The *token containment metric tcm(v, v’)* with respect to G between tuples v and v’ in G is given by the containment metric cm(TS(v), TS(v’)) with respect to G between their token sets. For example, if all tokens have equal IDF values then tcm([“MO”, “United States”], [“MO”, “United States of America”]) is 1.0; And, tcm([“MO”, “United States of America”], [“MO”, “United States”]) is 0.6.

Observe that when two tokens differ slightly due to a typographical error, token containment metric still treats them as two distinct tokens. To address this shortcoming, we treat two very similar tokens—with *edit distance*[3] less than a small value (say, 0.15)—in $B_t(G)$ to be synonyms.

3.3.2. Co-occurrence Similarity Function (fkcm)

In a dimensional hierarchy, a tuple in the parent relation R_i joins with a set, which we call its *children set*, of tuples in the child relation. We measure the co-occurrence between two distinct tuples by the amount of overlap between children sets of the two tuples. An unusually significant co-occurrence (more than the average overlap between pairs of tuples in R_i or above a certain threshold) is a cause for suspecting that one is a duplicate of the other. For example, in Figure 1, duplicate states MO and Missouri co-occur with the city “Joplin” whereas other distinct states do not co-occur with any common cities. Informally, our co-occurrence measure—called the *foreign key containment*

[3] The edit distance between tokens t_1 and t_2 is the minimum number of edit operations (delete, insert, transpose, and replace) required to change t_1 to t_2; we normalize this value with the sum of their lengths [AEP01].

metric (fkcm)—between two tuples is the containment metric between the children sets of the first and the second tuples.

If i > 1, we say that two tuples v_1 and v_2 in R_i *co-occur* through a tuple v in R_{i-1} if they both join with v. In general, two distinct tuples v_1 and v_2 in R_i join with two sets S_1 and S_2 (usually with little overlap) of tuples in R_{i-1}. We call S_1 the *children set CS(v_1)* of v_1, and S_2 the *children set CS(v_2)* of v_2. Let G={v_1, …, v_n} be a set of tuples from R_i. Let B_c(G) be the bag (multi-set) of all children tuples in R_{i-1} with any tuple in G as parent. The *child frequency cf(c) of a child tuple c* with respect to G is the number of times c occurs in B_c(G). The *FK-containment metric fkcm(v, v')* with respect to G between v and v' in G is the containment metric cm(CS(v), CS(v')) with respect to B_c(G) between the children sets CS(v) and CS(v'). For example, the FK-containment metric between values "Missouri" (whose State.Id is S4) and "MO" (whose State.Id is S3) in the State relation of Figure 1 is 1.0 because their children sets are identical ({Joplin}).

Note that while measuring co-occurrence between two tuples in R_i, we only use R_{i-1} and disregard information from relations further below for two reasons. First, the restriction improves efficiency because the number of distinct combinations joining with a tuple in R_i increases as we go further down the hierarchy. For example, the number of state tuples pointing to "United States" in the Country relation is less than the number of [city, state] tuple pairs that point to it. Therefore, the restriction enables efficient computation of our co-occurrence measure between tuples. Second, the co-occurrence information between tuples in R_i provided by relations R_j (j < i-1) is usually already available from R_{i-1}. Tuples in R_j (j < i-1) which join with the same tuple in R_i are also likely to join with the same tuples in R_{i-1} if the children sets of distinct tuples are very different from each other. We discuss two exceptional cases in Section 5.

3.3.3. Combination Function

We use thresholded similarity metrics for detecting duplicates. That is, when the similarity cm(v, v') between v and v' is greater than a threshold, then the duplicate detection function using cm predicts that v is a duplicate of v'. We now discuss the combination of predictions obtained from both functions. We adopt a *weighted voting* of the predictions where the weight of a prediction is proportional to the "importance of the information" used to arrive at the prediction.[4] As discussed earlier, IDF values of the token and children sets capture the concept of amount of information because sets containing more distinguishing tokens or children tuples have higher IDF values.

For a tuple v in R_i (i > 1) let w_t=IDF(TS(v)), and w_c=IDF(CS(v)). Let *tcm_threshold* and *fkcm_threshold* be the textual and co-occurrence similarity thresholds, respectively. Let pos: R→{1,-1} be a function defined as follows: pos(x) = 1, if x > 0, and -1, otherwise. Our weighted voting combination function is:

*pos(w_t*pos(tcm(v,v')-tcm_threshold)+w_c*pos(fkcm(v,v')-fkcm_threshold)).*

Essentially, the combination function returns the prediction, 1 (duplicate) or -1 (not a duplicate), of the similarity function that has a higher weight. Suppose that in Figure 1, "UK" is considered a duplicate of "USA" according to a textual similarity function. Because they do not co-occur with any state tuple, fkcm contradicts this prediction. Since the children set of UK has a higher IDF value than its token set, UK is not a duplicate of USA.

4. Delphi

We now describe Delphi. Recall that we consider two entities to be duplicates if the respective pairs of tuples in each relation of the hierarchy are duplicates. That is, two entities in the customer information of Figure 1 are duplicates only if the Organization tuples, City tuples, State tuples, and Country tuples are all duplicates of each other. Therefore, a straightforward duplicate detection algorithm would be to independently determine sets of duplicate tuples at each level of the hierarchy and then determine duplicate entities over the entire hierarchy. For the example in Figure 1, we can process each of the Organization, City, State, and Country relations independently to determine duplicate pairs of tuples in these relations. We may then identify pairs of duplicate entities if their corresponding tuples at each level in the hierarchy (Organization, City, State, and Country) are either equal or duplicates.

We can be more efficient by exploiting the knowledge from already processed relations. Suppose we know that only "United States of America" and "United States" are duplicates of "USA" and the rest are all unique tuples in the Country relation. While processing the State relation, we do not compare the tuple "BC" with "Missouri" because the former joins with Canada and the latter with (duplicates of) USA. Observe that this usage requires us to process a parent relation in the hierarchy before processing its child. As we move down the hierarchy, the reduction in the number of comparisons is significant. For instance, the Organization relation may have millions of tuples whereas the number in Seattle, WA, USA may be a few thousands.

We adopt a *top-down traversal* of the hierarchy. After we process the topmost relation, we group the child relation

[4] For the lowest relation R_1 in the hierarchy, we return the prediction of tcm.

below into relatively smaller groups (compared to the entire relation) and compare pairs of tuples within each group. Let S_i be the join of R_{i+1}, …, R_m through key—foreign key attribute pairs. We use the knowledge of duplicates in S_i to group relation R_i such that we place tuples r_{i1} and r_{i2} which join with combinations s_{i1} and s_{i2} from S_i in the same group if s_{i1} and s_{i2} are equal or duplicates (i.e., corresponding pairs of tuples in s_{i1} and s_{i2} either match exactly or are duplicates). We then process each group of R_i independently. Observe that we require S_i to be grouped into sets of duplicates. Due to efficiency considerations, we further restrict that these sets be disjoint. Otherwise, same sets of tuples in R_i may be processed in multiple groups causing repeated comparisons between the same pairs of R_i tuples.

Considering the example in Figure 1, our top-down traversal of the dimensional hierarchy is as follows. We first detect duplicates in the Country relation, then process the State relation grouping it with the processed Country relation, then process the City relation grouping it with the processed [State, Country] combination, and then finally process the Organization relation grouping it with the processed [City, State, Country] combination.

The remainder of this section is organized as follows. In Section 4.1, we discuss the procedure for detecting duplicates within a group of tuples from a relation in the hierarchy. In Section 4.2, we discuss the top-down traversal of the hierarchy coordinating the invocation of the group wise duplicate detection procedure. We do not explicitly discuss the special case of the lowest relation where we cannot use fkcm. The following discussion can easily be extended to this special case.

4.1. GroupWise Duplicate Detection

We now describe a procedure to detect duplicates among a group G of tuples from a relation in the hierarchy. The output of this procedure is a partition of G into sets such that each set consists of variations of the same tuple. First, we determine pairs of duplicates and then partition G.

As discussed earlier, our duplicate detection function requires the predictions from threshold-based decision functions using tcm and fkcm metrics. A straightforward procedure is to compare (using tcm and fkcm) all pairs of tuples in a group G, and then to choose pairs whose similarity is greater than the (tcm or fkcm) threshold. We reduce the number of pair wise comparisons between tuples by pruning out many tuples that do not have any duplicates (according to tcm or fkcm) in G. We describe each step in detail below first assuming that the tcm and fkcm thresholds are known. In Section 4.3, we describe a method to dynamically determine thresholds for each group.

4.1.1. Duplicate Detection using tcm

We want to detect all pairs (v_1, v_2) of tuples where v_1 is a duplicate, according to tcm, of v_2; i.e., tcm(v_1, v_2) > tcm-threshold. To reduce the number of pair wise tuple comparisons, we use a *potential duplicate identification filter* for efficiently isolating a subset G' consisting of all potential duplicates. That is, a tuple in G-G' is not a duplicate of any tuple in G. Duplicate detection on G consists of: (i) identifying the set G', and (ii) comparing each tuple in G' with tuples in G it may be a duplicate of.

Since tcm compares token sets of tuples, we abuse the notation and use *tcm(v, S)* to denote the comparison between the token set of a tuple v and the multi-set union of token sets of all tuples in the set S. We use similar notation for fkcm as well.

Potential Duplicate Identification Filter

The intuition behind our *filtering strategy* to determine the set G' of all potentially duplicate tuples is that the tcm value between any two tuples v and v' in G is less than that between v and G-{v}. Therefore, a tuple v for which tcm(v, G-{v}) is less than the specified threshold is not a duplicate of any other v' in G. We only perform |G| comparisons to identify G', which potentially is much smaller than G. Therefore, comparing pairs involving tuples in the *filtered* set can be significantly more efficient than comparing all pairs of tuples in G.

The intuition behind our filtering strategy is captured by the following observation for tcm (and fkcm). The observation follows from the fact that the multi-set union of token sets of all tuples in G-{v} is a superset of token set of any v' in G-{v}.

Observation 4.1: Let cm denote either tcm or fkcm metric, and v and v' be two tuples in a set G of tuples. Then,

$$cm_G(v, v') \leq cm_G(v, G\text{-}\{v\})$$

Computing tcm(v, G-{v}) using Token Tables

We now describe a technique to efficiently compute tcm(v, G-{v}) for any tuple v in G. The intuition is that tokens in the intersection of the token set TS(v) of v and the multi-set union of token sets of all tuples in G-{v} have a frequency, in the bag of tokens $B_t(G)$ of G, of at least 2. Any other token is unique and has a frequency 1.

We build a structure called the *token table* of G containing the following information: (i) the set of tokens whose frequency tf(t) w.r.t. $B_t(G)$ is greater than one, (ii) the frequencies of such tokens, and (iii) the list of (pointers to) tuples in which such a token occurs. The difference between a token table and an inverted index over G is that the token table only contains tokens whose frequency with respect to G is *greater than 1*, and hence potentially smaller if a large percentage of tokens in $B_t(G)$ are unique.

We maintain lists of tuple identifiers only for tokens which are not very frequent. The frequency at which we start ignoring a token—called the *stop token frequency*—is set to be equal to 10% of the number of tuples in G. As mentioned earlier, we enhance tcm by treating tokens which are very close to each other according to edit distance (less than 0.15, in our implementation) to be synonyms. Due to space constraints, we skip the details of token table construction.

Example 4.1.1: In Figure 1, suppose we are processing the State relation grouped with the Country relation, and that we detected the set {United States, United States of America, USA} to be duplicates on the Country relation. For the group of State tuples joining with USA and its duplicates, the token table consists of one entry: {[token=MO, frequency=3, tupleId-list=<S1, S2, S3>]}.

The computation of tcm(v, G-{v}) requires frequencies with respect to $B_t(G)$ of tokens in TS(v), which can be obtained by looking up the token table. Tokens absent from the token table have a frequency 1. Now, any tuple v such that tcm(v, G-{v}) is greater than tcm-threshold is a *potential duplicate tuple*, and is added to G'.

Computing Pairs of Duplicates

We compare each tuple v in G' with a set S_v of tuples, which is the union of all tuples sharing tokens with v. S_v can be obtained from the token table. (For any tuple v'' *not in* S_v, tcm(v, v'') = 0.) For any tuple v' in S_v such that tcm(v, v') > tcm-threshold, we add the pair (v, v') to the pairs of duplicates from G.

4.1.2. Duplicate Detection using fkcm

We predict that a tuple v is a duplicate, according to fkcm, of another tuple v' in G if fkcm(v, v') > fkcm-threshold. Using Observation 4.1, we determine a set of potential duplicates by efficiently computing fkcm(v, G-{v}) using *children tables*. The computation of the set G' of potential duplicates and then duplicates, according to fkcm, of tuples in G' is the same as for tcm. Hence, we only describe the construction of the children table for a group G of tuples.

Children Tables

The *children table of G* is a hash table containing a subset of the union of children sets of all tuples in G. It contains: (i) each child tuple c from R_{i-1} joining with some tuple in G, and whose frequency cf(c) in $B_c(G)$ is greater than one, (ii) the frequencies of such children tuples, and (iii) the list of (pointers to) tuples in G with which c joins. We maintain lists of tuples only for children that have a frequency less than the *stop children frequency* fixed at 10% the number of tuples in G.

Example 4.1.2: Consider the example in Figure 1. We process the State relation grouped with the Country relation. Suppose {United States, United States of America, USA} is a set of duplicates on the Country relation. For the group of State tuples joining with USA and its duplicates, the children table contains one entry: {child=Joplin, frequency=3, tupleId-list=<S1,S3, S4>}.

Note: Recall that the frequency of a child tuple in $B_c(G)$ is based only on its descriptive attribute value combinations and ignores the generated key attributes in R_{i-1}. In the above example, the tuple Joplin has a frequency 3 because we ignore the CityId attribute values.

Building the Children Table: The procedure is similar to that of building the token table except for one difference: The multi-set union of all children sets $B_c(G)$ can be large, e.g., all street addresses in the city [Illinois, Chicago], and hence may not fit in main memory. Therefore, we follow the steps below. We refer to tuples in $B_c(G)$ with frequency greater than one as *non-unique* tuples.

(i) We fetch all *non-unique* tuples in $B_c(G)$ into a hash table.
(ii) We fetch tuples in G and their children, one pair at a time, and associate non-unique tuples in $B_c(G)$ with the list of G tuples they join with.

Combination

After detecting duplicates according to tcm and fkcm, we combine (using the combination function of Section 3.3.3) predictions for each pair of tuples detected to be duplicates using either tcm or fkcm or both.

4.1.3. Grouping Duplicate Pairs into Sets

Coordinating the top-down traversal of the hierarchy requires us to partition G into sets of duplicates, and to determine a representative tuple—called the *canonical tuple*—for each set to be able to exploit database systems for processing. (This issue will be clearer in the next section.) To partition G into sets of duplicates, we adapt a method from [HS95] to handle asymmetric similarity functions. The essential idea is to divide G into connected groups and choose a canonical tuple for each group.

Following the standard approach [HS95, ME96], we elevate the relationship "is a duplicate of" between tuples to be a transitive relation. That is, if v_1 is a duplicate of v_2 and v_2 that of v_3, we consider v_1 to be a duplicate of v_3. The intuition behind the partitioning method is to identify *maximal connected sets* of duplicates such that for any pair of tuples v and v' in each set, we can either deduce using transitivity that v is a duplicate of v' or vice versa. A connected set is maximal if we cannot add any more tuples to it without making it disconnected. For each connected set, we choose the tuple with the highest IDF value (of token sets for R_1 and of children sets for higher level relations) as the canonical tuple. Because the relationship "is a duplicate of" is asymmetric, a tuple may end up in multiple connected sets. For such a tuple v, we place it in

View Definitions	Qi =	Qi’ =
L_m = Select * From R_m, …, R_1 L_i = Select $L_{i+1}.A_m$, …, (Case When $T_{i+1}.A_{i+1}$ is Null Then $L_{i+1}.A_{i+1}$ Else $T_{i+1}.A_{i+1}$), $L_{i+1}.A_i$, …, $L_{i+1}.A_1$ From L_{i+1} Left Outer Join T_{i+1} On $L_{i+1}.A_m = T_{i+1}.A_m$, …, $L_{i+1}.A_{i+1} = T_{i+1}.A_{i+1}$	Select $L_i.\mathbf{A}_m$, …, $L_i.\mathbf{A}_{i+1}$, $L_i.\mathbf{A}_{i-1}$, count(*) From (Select distinct $L_i.\mathbf{A}_m$, …, $L_i.\mathbf{A}_{i-1}$) Group By $L_i.\mathbf{A}_m$, $L_i.\mathbf{A}_{i+1}$, $L_i.\mathbf{A}_{i-1}$ Having count(*) > 1 Order By $L_i.\mathbf{A}_m$, …, $L_i.\mathbf{A}_{i+1}$, $L_i.\mathbf{A}_{i-1}$	Select $L_i.\mathbf{A}_m$, …, $L_i.\mathbf{A}_{i+1}$, $L_i.\mathbf{A}_i$, $Li.\mathbf{A}_{i-1}$ From (Select distinct $L_i.\mathbf{A}_m$, …, $L_i.\mathbf{A}_{i-1}$) Order By $L_i.\mathbf{A}_m$, …, $L_i.\mathbf{A}_i$, $L_i.\mathbf{A}_{i-1}$

Figure 2: View definitions and Queries

the set with the closest (computed using fkcm at higher levels and tcm at the lowest level) canonical tuple.

4.2. Top-down Traversal

We now describe the top-down traversal of the hierarchy. Starting from the topmost relation, we group each relation and invoke the duplicate detection procedure on each group. Therefore, the primary goal of the traversal is to group each relation appropriately. While grouping a relation R_i by a combination S_i (the join of R_{i+1},…, R_m) of processed relations, all R_i tuples which join with tuple combinations (equivalently, sub-entities) in S_i that are either exactly equal or detected to be duplicates have to be placed in the same group.

A straightforward ordering by S_i of the join of R_i and S_i does not achieve the desired grouping because duplicate tuple combinations in S_i may not be adjacent to each other in the sorted order. For example, duplicates UK and Great Britain on the Country relation are unlikely to be adjacent to each other in the sorted order. Therefore, we realize the correct sorted order by considering a new relation L_i, which is the join of R_1,…,R_m but with the duplicate tuples in processed relations (R_{i+1},…,R_m) replaced by their *canonical tuples*. We then group (the relevant projection of) L_i by the canonical tuple combinations of S_i. We avoid explicit materialization of the very large (as large as the database) relations L_i by only recording detected duplicates in *translation tables*. Translation tables can be significantly smaller than the database if the number of duplicates is much less than the number of tuples in the database.

Translation Tables

Informally, the translation table T_i records the mapping between each duplicate tuple in R_i and its *canonical tuple*, as well as the ancestral combination from the join of R_{i+1}, …, R_m to which they both point to. While storing the ancestral combination, we assume that all duplicate tuples in relations R_{i+1}, …, R_m have been replaced with their canonical tuples. For example, if USA is the canonical tuple of the set of duplicates {United States, United States of America}, and MO is that of the set {Missouri} of states pointing to USA (or United States or United States of America) the translation table at Country relation level maps both United States and United States of America to USA. And, the translation table at the State level maps [USA, Missouri] to [USA, MO].

Let Canonical_R_i represent the relation R_i where each duplicate tuple has been replaced with its canonical tuple. The translation table T_i has the schema: [R_i, R_i AS Canonical_R_i, Canonical_R_{i+1},...,Canonical_R_m]. T_i records each duplicate tuple v and its canonical tuple v’ along with the canonical tuple combination s_v from the grouping combination [Canonical_R_{i+1},...,Canonical_R_m] of relations with which v and v’ join.

Coordination

We form two SQL queries Q_i and Q_i’ whose results contain the information required for processing any group in R_i. We scan portions of these query results, pause and process a group of R_i tuples, and then continue the scans. First, we define the set of views used by these queries.

The sequence of views L_m, …, L_i are defined in Figure 2. Informally, L_i represents the current state of the unnormalized relation R (the join of R_1,…, R_m) after all duplicate tuples (in R_{i+1},…,R_m) are collapsed with their canonical tuples. Each L_j has the same schema as the unnormalized dimension relation R. Considering the translation table on the Country relation, an *outer join* between the original unnormalized relation R and the translation table on the country attribute results in a new unnormalized relation L with a canonical_Country attribute. In L, United States and United States of America are always replaced by USA, which is their canonical equivalent.

The queries Q_i and Q_i’ are defined in Figure 2 in which A_i denotes the set of descriptive attributes (not including generated keys) in R_i. For the sake of clarity, we omit the key—foreign key join conditions in the where clause in Figure 2. Both queries Q_i and Q_i’ order (a projection of) L_i on S=[$L_i.A_m$,…,$L_i.A_{i+1}$]. Let s be a tuple combination in S, and let G_s be the group of tuples in R_i joining with s. We invoke the duplicate detection procedure discussed in Section 4.1 for each group G_s as follows. We scan the result of Q_i to fetch a group G1 of tuples joining with s, scan the corresponding group G2 from the result of Q_i’, process G_s using G1 and G2, and then move on to a subsequent group. The group G1 consists of the information required for building the hash table of non-unique children $B_c(G_s)$, and G2 that for associating non-unique children with parent tuples as well as for building the token table. Note that we do not maintain all of G2 in memory and only require a tuple at a time.

4.3. Dynamic Thresholding

In many cases, it is difficult for users to set tcm and fkcm thresholds. Hence, we develop a technique to dynamically determine thresholds for each group. Moreover, treating each group independently allows us to set qualitatively better thresholds by adapting to the characteristics of that group. For example, the numbers of tokens may vary significantly across groups (names in Argentina may be longer than they are in USA).

The intuition behind our threshold determination is that when the fraction of duplicates in a group is small (say, around 10%), a duplicate tuple v is likely to have a higher value for containment metric (tcm or fkcm) between v and G-{v} than a unique tuple. Therefore, we expect them to be outliers in the distribution of tcm and fkcm. We use standard outlier detection methods based on Normality assumptions to set thresholds. In Section 6, we demonstrate experimentally that our threshold determination procedure is quite effective.

4.4. Resource Requirements

For processing each relation R_i in the hierarchy, we send two queries (Q_i and Q_i') to the database system where each query computes the join of relations R_m, …, R_1. Key—foreign key joins can be made very efficient if we create appropriate join indexes. We expect the number of duplicates and hence the translation tables to be small. Hence, outer joins with translation tables are efficient.

Main Memory Requirements: The group level duplicate elimination procedure ideally requires for each group G, the token table, the children table, and the tuples in G to be in main memory. If the frequency distribution of children or tokens follows the Zipfian distribution, which is true for most real datasets [Zipf49], then less than half the tokens or children tuples have frequencies greater than 1, and are maintained in memory. In rare cases where a group being processed is very large, we may materialize the token and children tables on disk and build appropriate indexes.

5. Discussion

We now discuss several interesting issues starting with a note that we do not require the dimensional information to be normalized into relations R_m, …, R_1. We can adapt Delphi to work with an unnormalized relation R (the join of R_m,…, R_1) as long as the sets of attributes which form the hierarchy are known.

FKCM Measurement

Recall that the fkcm metric only uses information from one level below. Such a strategy is very efficient and sufficient for most but the following two exceptional cases. We now discuss these two cases.

S*mall children sets:* When the children set of a tuple v_1 is so small that even a single erroneous tuple in $CS(v_1)$ is a significant fraction, we may incorrectly believe that v_1 is unique when in fact it is a duplicate of v_2. If we want to detect such errors, we modify the children table construction and processing as follows. We first add all children tuples in $B_c(G)$ (even those with frequency 1) to the children table. We treat all pairs of duplicate (according to tcm) tuples as synonyms when measuring the FK-containment metrics between their parents. Since we have to temporarily maintain all children tuples—even those with frequency 1—we require additional main memory.

Correlated errors: Consider two sets of tuples in each relation where one uses abbreviations and the other uses expanded versions while reporting the country and state values. Then, a tuple ("United States", "Washington", **) may be a duplicate of ("USA", "WA", **) where ** represents the same set of values in both tuples. We may not detect that "United States" is a duplicate of USA through co-occurrence unless we look one level below the States relation. It is possible to overcome this limitation by measuring, with significant computational overhead, co-occurrence through lower level relations. However, the number of combinations may sometimes be too high (e.g., all organizations in USA) to even fit in main memory.

Definition of Duplicates

We now discuss a limitation of our definition of duplicates. Consider the following pair of entities: [<Smith>, <98052>, <WA>, <USA>] and [<Smith>, <98052>, <Washington>, <Canada>]. If the tuples "Canada" and "USA" are not (and rightly so) considered duplicates of each other on the Country relation, then according to our definition, the two entities are not duplicates. Observe that the second tuple violates an implicit or explicit functional dependency or rule: "zipcode=98052 and state=WA → country=USA." If we correct the violation and detect that

MP-CM	Windowing, no hierarchy, no co-occurrence, global thresholds, Cosine metric
MP-ED	Windowing, no hierarchy, no co-occurrence, global thresholds, Edit distance
Delphi-Global	Grouping, hierarchy, co-occurrence, global thresholds
Delphi	Grouping, hierarchy, co-occurrence, dynamic thresholding
Delphi-Stripped	Grouping, hierarchy, no co-occurrence, dynamic thresholding

Table 1: Algorithms

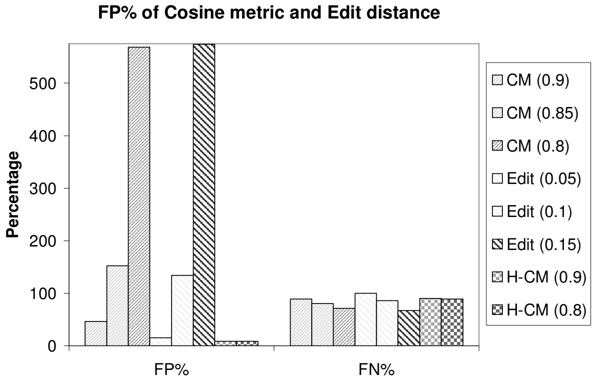


Figure 3: False Positive Explosion

WA and Washington are duplicates (using co-occurrence information), then the two customer entities are duplicates. Thus, even though our definition of duplicates does not directly allow such inconsistencies, we can correct them in conjunction with other cleaning operations.

Potential Duplicate Identification Filter

Imagine a set G of tuples where most of the tokens in $B_t(G)$ occur in at least two tuples in G. In such cases, the filtering strategy is not very effective because we may mark many tuples as potential duplicates. Our experiments on real data illustrate that such a case does not typically occur in practice. However, developing appropriate filters for such rare cases is still an open issue.

We note that it is possible to consider similarity and combination functions other than the ones we used. However, Observation 4.1, which summarizes our filtering strategy, may not be valid for all similarity functions, and one may have to design suitable filters where possible.

6. Experimental Evaluation

Using real datasets, we now evaluate the quality and efficiency of Delphi and compare with earlier work.

6.1. Datasets and Setup

We consider clean Customer information from an internal operational data warehouse and introduce errors.[5] The Customer dimensional hierarchy has four relations: Name (level 1), City (level 2), State (level 3), Country (level 4) with 269678, 21856, 1250, and 115 tuples, respectively. Because we start from real data all characteristics of real data—variations in the lengths of strings, numbers of tokens in and frequencies of attribute values, co-occurrence patterns, etc.—are preserved. Since we know the duplicate tuples and their correct counterparts in the erroneous dataset, we can evaluate duplicate elimination algorithms.

Error Introduction

We introduce two types of errors common in data warehouses [For01]: *equivalence errors, spelling & truncation errors*. The generator has three parameters: The first *percentage error* parameter controls the error to be introduced in each relation. The second (*equivalence fraction)* and the third (*spelling fraction)* parameters control the fractions of equivalence errors, spelling and truncation errors, respectively. Suppose the percentage error is 10% and the equivalence fraction is 50% then we will introduce 10% duplicate tuples into the input table out of which 50% will be due to equivalence errors.

Equivalence Errors: Consider the tuple combination [<Key Associates>, <Joplin>, <MO>, <USA>] in the customer table. Suppose we want to create an equivalence error for "MO" in the state relation. We first garble "MO" into, say, "xMykOz" so that the new value is undetectable by standard textual similarity functions. Since equivalence errors usually occur in multiple tuples, we choose around 5% (5-x%, 5+x%) of all entities with R.country="USA" and R.state="MO" and modify the value of MO to "xMykOz." For 10% of these modified tuples, we also introduce errors in the tuple from the child relation, when one exists. We insert these erroneous tuples into R. At the lowest level of the hierarchy, we garble a randomly picked token from the token set and insert the modified tuple in R.

Spelling and Truncation Errors: We modify a token in a tuple by changing, deleting, adding characters or truncating the token. 50% of the time, we modify characters, and the remaining 50% we just truncate the token. The number of characters modified or truncated is a linearly decreasing function with a maximum of half the token length.

Token Permutation: Consider the example where a user enters first name followed by the last name instead of the stipulated last name followed by the first name. To reflect such types of errors, we randomly permute tokens in about 10% of the erroneous tuples being added to R.

Algorithms

Table 1 summarizes the algorithms we evaluate in this study. MP-CM and MP-Edit are derivatives of the windowing-based MergePurge (MP) algorithm using cosine metric and edit distance, respectively [HS95, ME97, Coh98]. Delphi-global is a variant of Delphi that uses global thresholds for both tcm and fkcm. Delphi-Stripped is a variant of Delphi which only uses tcm and completely ignores co-occurrence information.

We run variants of MP on the unnormalized relation of Name, City, State, and Country relations, and sort on the key (name, city, state, country). In both MP-CM and MP-Edit, we fix the window size at 20, and vary the thresholds. We use MP-CM(x) (MP-Edit) to denote that the threshold for the cosine metric (edit distance) is set to x. For Delphi-

[5] We observed similar results on the publication information of a bibliography database. We omit results due to space constraints.

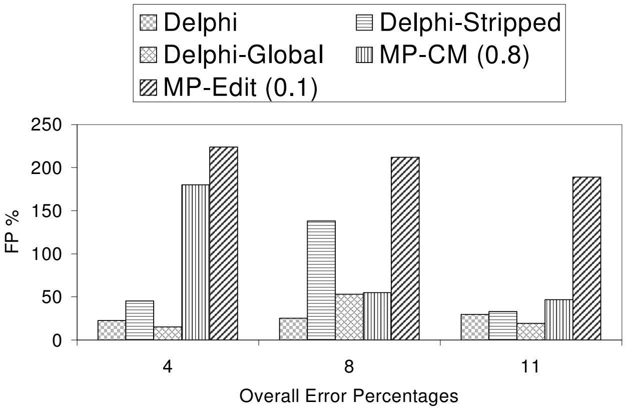


Figure 4: False Positive Percentages

Global, we arrived at the global tcm-threshold and the fkcm-threshold of 0.80 and 0.85, respectively, after several trials. To compare the quality of algorithms, we do not group duplicate tuples for the lowest Name relation and output all pairs of duplicates detected by Delphi.

Quality Metrics

We now describe the quality metrics for evaluating algorithms.

False positives: The percentage of *incorrect* pairs of tuples which an algorithm detects as duplicates relative to the actual number of duplicates is called the *false positive (FP) percentage.* The false positive percentage can be greater than 100 if the algorithm produces many incorrect pairs. Lower false positive percentage indicates higher confidence in the algorithm's results.

False negatives: The percentage of undetected duplicates in the input dataset relative to the number of duplicates is called the *false negative percentage.* Lower false negative percentages indicate good duplicate detection.

6.2. Analysis of Results

6.2.1. False Positive Explosion

We now demonstrate that the use of cosine metric or edit distance can result in large false positive percentages. We consider a dataset with 8% overall error where the equivalence and spelling & truncation fractions at 0.5 each. Figure 3 shows the results of applying the windowing strategy on four different sort orders: [Name, City, State, Country], [City, State, Country, Name], [State, Country, Name, City], and [Country, Name, City, State]. *CM*(x) (*Edit(x)*) denotes the results from using cosine metric (edit distance) with a threshold x, and *H-CM* from using cosine metric with the restricted definition of duplicates in the presence of dimensional hierarchies. From Figure 3, we observe that lowering thresholds drastically increases false positive percentages for cosine metric and edit distance.

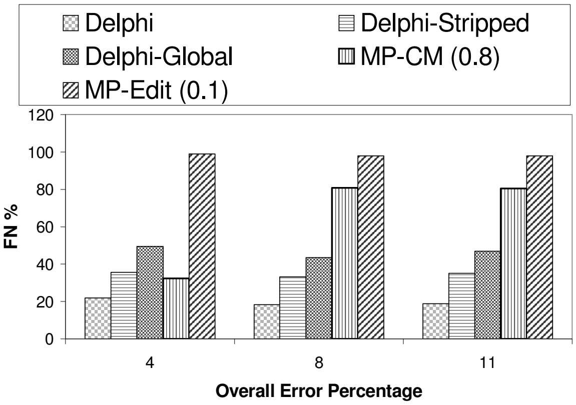


Figure 5: False Negative Percentages

6.2.2. Quality

In the following two experiments, we generated erroneous datasets from the input dataset by introducing 4%, 8%, and 11% errors with relative fractions of equivalence error and spelling & truncation errors fixed at 0.5.

Reduction in False Positive Percentages

Figure 4 shows the false positive percentages of each algorithm. Because Delphi and Delphi-global have significantly lower false positive percentages, we conclude that hierarchies and co-occurrence information together significantly reduce false positive percentages.

Reduction in False Negative Percentages

From Figure 5, which plots false negative percentages, we see that Delphi has the lowest false negative percentages. Therefore, co-occurrence information is useful in reducing false negatives as well. And, Delphi-Stripped is better than Delphi-Global. Hence, dynamic thresholding helps reduce false negative percentages. However, its impact on false positive reduction seems unpredictable.

6.2.3. Speed and Scalability

We ran Delphi, Delphi-Stripped, and MP-CM on datasets of size 3000, 30000, 300000, and 3000000.[6] Table 2 shows that Delphi and MergePurge are both scalable over a wide range of dataset sizes. Running times are normalized with respect to that of Delphi on a 3000 tuple dataset. We also note that maximum amount of main memory required by Delphi on any of the datasets we considered here is less than 25 MB, thus supporting our argument that token and children tables fit in memory.

#Tuples	Delphi	Delphi-Stripped	MP-CM
3000	1	0.8	0.7
30000	5.512	4.2	3.55
300000	52.5	43.7	151.5
3000000	510.4	230.6	1500

Table 2: Scalability

	#(TCM; FKCM)
Name	51582; 0
City	9997; 1093
State	434; 441
Country	30; 8

Table 3: Filtering

[6] Since the scalability characteristics of MP-Edit are similar to that of MP-CM, we do not consider it here.

6.2.4. Potential Duplicate Filter

We now evaluate our potential duplicate filtering technique. The dataset has 8% duplicate tuples. Table 3 shows the total number of potential duplicates over all groups in each relation of the hierarchy. The entry (x; y) denotes that tcm and fkcm returned x and y potential duplicates, respectively. We observe that only 20% (as compared to the minimum 16%=8% duplicates + 8% targets) of the overall set of tuples was even considered to be potential duplicates. Hence, potential duplicate filtering enhances efficiency. Also observe that fkcm returns fewer potential duplicates. Hence, we conclude that co-occurrence information is very effective at reducing false positives.

7. Conclusions

In this paper, we exploited dimensional hierarchies in data warehouses to develop a high quality, scalable, and efficient algorithm for detecting fuzzy duplicates in dimensional tables. In future, we intend to consider multiple hierarchies for detecting fuzzy duplicates.

Acknowledgements
We thank several members of the DMX group at Microsoft Research for their thoughtful comments.

References

[AEP01] A.N. Arslan, O. Egecioglu, and P.A. Pevzner. A new approach to sequence comparison: Normalized local alignment. *Bioinformatics*, 17(4):327--337, 2001.

[BDS01] Vinayak Borkar, Kaustubh Deshmukh, and Sunita Sarawagi. Automatic segmentation of text into structured records. In Proceedings of ACM Sigmod Conference, Santa Barbara, CA, May 2001.

[BGM+97] A. Broder, S. Glassman, M. Manasse, and G. Zweig. *Syntactic Clustering of the Web*. In Proc. Sixth Int'l. World Wide Web Conference, World Wide Web Consortium, Cambridge, pages 391--404, 1997.

[BGRS99] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? International Conference on Database Theory, pages 217--235. January 1999.

[BL94] V. Barnett and R. Lewis. Outliers in statistical data. John Wiley and Sons, 1994.

[BYRN99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. Modern Information Retrieval. Addison Wesley Longman, 1999.

[Coh98] W. Cohen. Integration of heterogeneous databases without common domains using queries based in textual similarity. In Proceedings of ACM SIGMOD, pages 201--212, Seattle, WA, June 1998.

[For01] Ronald Forino. Data e.quality: A behind the scenes perspective on data cleansing. http://www.dmreview.com/, March 2001.

[FS69] I. P. Felligi and A. B. Sunter. A theory for record linkage. Journal of the American Statistical Society, 64:1183--1210, 1969.

[Gal] Helena Galhardas. Data cleaning commercial tools. http://caravel.inria.fr/~galharda/cleaning.html.

[GFS+01] Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian Saita. Declarative data cleaning: Language, model, and algorithms. In Proceedings of the 27th International Conference on Very Large Databases, pages 371--380, Roma, Italy, September 11-14 2001.

[GFSS99] Helena Galhardas, Daniela Florescu, Dennis Shasha, and Eric Simon. An extensible framework for data cleaning. In ACM Sigmod, May 1999.

[GIJ+01] L Gravano, P Ipeirotis, H V Jagadish, N Koudas, S Muthukrishnan and D Srivastava. Approximate String Joins in a Database (Almost) for Free. In Proceedings of the VLDB 2001.

[GGR99] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Cactus--clustering categorical data using summaries. In Proceedings of the ACM SIGKDD fifth international conference on knowledge discovery in databases, pages 73--83, August 15-18 1999.

[GKR98] David Gibson, Jon Kleinberg, and Prabhakar Raghavan. Clustering categorical data: An approach based on dynamical systems. VLDB 1998, New York City, New York, August 24-27.

[GRS99] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Rock: A robust clustering algorithm for categorical attributes. In Proceedings of the IEEE International Conference on Data Engineering, Sydney, March 1999.

[HKPT98] Yka Huhtala, Juha Karkkainen, Pasi Porkka, and Hannu Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In proceedings of the 14th international conference on data engineering (ICDE), pages 392--401, Orlando, Florida, February 1998.

[HS95] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In Proceedings of the ACM SIGMOD, pages 127--138, San Jose, CA, May 1995.

[KA85] B. Kilss and W. Alvey. Record linkage techniques--1985. Statistics of income division. Internal revenue service publication, 1985. Available from http://www.bts.gov/fcsm/methodology/.

[KM95] J.Kivinen and H. Mannila. Approximate dependency inference from relations. Theoretical Computer Science, 149(1):129--149, September 1995.

[MBR01] J Madhavan, P Bernstein, E Rahm. Generic Schema Matching with Cupid. VLDB 2001, pages 49-58, Roma, Italy.

[ME96] Alvaro Monge and Charles Elkan. The field matching problem: Algorithms and applications. In Proceedings of the second international conference on knowledge discovery and databases (KDD), 1996.

[ME97] A. Monge and C. Elkan. An efficient domain independent algorithm for detecting approximately duplicate database records. In Proceedings of the SIGMOD Workshop on Data Mining and Knowledge Discovery, Tucson, Arizona, May 1997.

[MR94] H. Mannila and K.-J. Raiha. Algorithms for inferring functional dependencies. Data and Knowledge Engineering, 12(1):83--99, February 1994.

[NR99] Felix Naumann and Claudia Rolker. Do metadata models meet iq requirements? In Proceedings of the international conference on data quality (IQ), MIT, Cambridge, 1999.

[Pro] MIT Total Data Quality Management Program. Information quality. http://web.mit.edu/tdqm/www/iqc.

[RD00] Erhard Rahm and H. Hai Do. Data cleaning: Problems and current approaches. IEEE Data Engineering Bulletin, 23(4):3--13, December 2000.

[RH01] Vijayshankar Raman and Joe Hellerstein. Potter's wheel: An interactive data cleaning system. VLDB 2001, pages 381--390, Roma, Italy.

[Zipf49] G.K. Zipf. Human behaviour and the principle of least effort. Addison-Wesley, Reading, MA, 1949.