Sampling Dirty Data for Matching Attributes *

Henning Köhler
The University of Queensland
Brisbane, Australia
henning@itee.uq.edu.au

Xiaofang Zhou
The University of Queensland and NICTA
Brisbane, Australia
zxf@itee.uq.edu.au

Shazia Sadiq
The University of Queensland
Brisbane, Australia
shazia@itee.uq.edu.au

Yanfeng Shu
CSIRO, Tasmanian ICT Centre
Hobart, Australia
yanfeng.shu@csiro.au

Kerry Taylor
CSIRO, ICT Centre
Canberra, Australia
kerry.taylor@csiro.au

ABSTRACT

We investigate the problem of creating and analyzing samples of relational databases to find relationships between string-valued attributes. Our focus is on identifying attribute pairs whose value sets overlap, a pre-condition for typical joins over such attributes. However, real-world data sets are often 'dirty', especially when integrating data from different sources. To deal with this issue, we propose new similarity measures between sets of strings, which not only consider set based similarity, but also similarity between strings instances. To make the measures effective, we develop efficient algorithms for distributed sample creation and similarity computation. Test results show that for dirty data our measures are more accurate for measuring value overlap than existing sample-based methods, but we also observe that there is a clear tradeoff between accuracy and speed. This motivates a two-stage filtering approach, with both measures operating on the same samples.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design

General Terms

Algorithms, Experimentation, Measurement, Performance

1. INTRODUCTION

A central problem that arises when integrating database systems, is to identify relationships between tables and attributes. Clearly this is not a novel issue, and a large number of approaches and algorithms for determining such relationships (semi-)automatically have been developed [1]. At the core of such methods lies the identification of matching attributes. Here one can distinguish between schema-based matchers and instance-based matchers [2]. In general, instance-based matchers are much slower, since they rely on the actual data sets rather than just schema information, but they are also potentially more accurate. This is particularly true when integrating data from different sources, which do not conform to any common naming standards. But even for databases using well-standardized attribute names, schema-based matchers are limited in their power: while they may help to identify attributes with the same semantic domain, they cannot verify whether they reference the same real-world objects. To obtain the latter information, we must examine the actual values stored for each attribute.

Looking more closely, we can distinguish two different types of attribute matches: domain equivalence and value overlap. Attributes sharing many common values almost always share the same semantic domain (e.g. people, locations, companies). While distinct domains *can* share values, e.g. "Darwin", it is unlikely that attributes from different domains have significant overlap. All attribute pairs with matching domains may be relevant for data integration, and thus should be discovered. However, it is also important to know whether an overlap in values exists, as this is essential in deciding whether joins or approximate joins - particularly vital for areas such as (semi-)automatic schema integration or keyword search over relational databases - on the attributes in question may be sensible. Tables carrying information about the same entities should be integrated differently (via join on such attributes) from tables carrying information about different entities from the same domain (via union). In most cases, joins on attributes without common values can safely be considered as being 'not sensible'.

One application scenario for attribute matching is the initial ad-hoc integration in the data-space approach [3, 4], where the goal is to loosely integrate data from very different, usually distributed database systems. Here it is not realistic to assume that relationships are precise, or that data is clean and formatted in a consistent manner. To some degree, the same problem of 'dirtiness of data' is bound to arise in many other data integration scenarios. While classic data cleaning techniques exist to deal with some of these problems, they are usually too expensive to be applied as an initial step when dealing with large numbers of attributes or large data volumes, and/or require domain-specific information (e.g. ontologies) which may be tricky to obtain.

*This work is partly supported by Australian Research Council (project LP0882957), CSIRO Flagship Collaboration Fund, and the Tasmanian ICT Centre.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

We therefore require a method for quickly identifying candidate matches, with reasonable if not optimal accuracy. Afterwards, accuracy can be improved through more complex automatic techniques or human inspection, if necessary.

To deal with large volumes of potentially distributed data, we employ sampling techniques as already suggested in [5]. Sampling can be done locally where the data is stored, and only the sample set needs to be transmitted. Once all samples have been collected, they can then be analyzed to find matching attribute pairs. Even if data is not distributed, using samples can speed up computation significantly.

To cope with the problem of dirty data (typos, different formatting, etc.), we introduce a novel sampling technique which preserves the 'synchronization' property (discussed in detail in Section 2.1) of classic sampling techniques [6, 7, 5]. Importantly this also works for string values which are only similar, rather than identical. As a result of synchronization, sample sizes can be much smaller compared to independent random sampling. Furthermore, we propose new similarity measures, which allow us to match strings with small differences, and which can be computed easily and efficiently from our samples. We will focus on relational databases only, although in principle we are simply comparing sets-of-sets, which happen to be obtained by transforming each string in an attribute's value set into a set of substrings. Thus our methods should be applicable to other data models as well. Our main contributions are:

1. an extension of sampling methods to sets-of-sets (sets of 'dirty' strings)
2. efficient similarity measures between sets-of-sets (sets of 'dirty' strings)

Note that for similarity measures to be 'efficient', it must be possible to estimate them using small samples, and to compute such approximations quickly. In combination, our methods can be used to rapidly identify relationships (value overlap to be precise) between attributes, which is an important task in data integration. Our approach differs from existing work in that it specifically caters for dirty data, while remaining efficient enough to be useful for initial integration of large data sets.

Closely related work is discussed in Section 2, where we describe a framework and techniques used for clean value sets, comparing values for equality only. This is then extended in Section 3 to incorporate similarity between values. Variations to our approach which increase accuracy without slowing computation are presented in Section 4. A more accurate but computationally more expensive similarity measure is given in Section 5, followed by experimental results in Section 6. Further, more loosely related work is discussed briefly in Section 7. Section 8 concludes.

2. SET SIMILARITY

In the following we shall discuss a general framework for attribute matching, and existing work in this area. The methods described are designed for clean values, but they form the basis for our approach. Thus an in-depth discussion is necessary.

The first step towards finding matching attributes is to compute similarity for all pairs of value-sets. In the Bellman system [5], a (relatively small) signature for each attribute's value set is generated using sampling techniques. If data is distributed, these will then be sent to a central location where they are analyzed. The result is a similarity graph, with attributes as vertices, and weighted edges between them denoting the similarity of the associated value sets. A well-known measure for similarity of sets is their resemblance or Jaccard index.

DEFINITION 1. *The* resemblance *of two sets A, B is*

$$res(A, B) := \frac{|A \cap B|}{|A \cup B|}$$

When generating samples, one must assure that the resemblance of A and B can be estimated from their samples, and that such estimates are accurate for reasonably small samples. Furthermore, each attribute should only be sampled once, and the same sample be used for comparison with all other attributes.

2.1 Sample Generation

The signature for an attribute contains a unique identifier, a sample of the value set, and possibly other data such as the size of the attribute's value set or sampling parameters. Mostly though, we will be talking about the signature $S(A)$ of a value set A, and mean just the sample set.

Sampling should be *synchronized*, i.e., if a particular value x is sampled for A then it is also sampled for B (provided it lies in B), and vice versa:

$$x \in S(A) \text{ and } x \in B \quad \Rightarrow \quad x \in S(B)$$

Independent random sampling could mean that even identical value sets produce different, possibly even disjoint signatures. Consider two attributes with one million distinct values each: Even if their value sets are identical, we need to sample about 1000 values from each before we can expect an intersection size of 1. To estimate their overlap with reasonable accuracy we require at least 100,000 values as sample, whereas 100 values are sufficient for synchronized sampling [5]. Efficient sampling is necessary to make value-based attribute matching scale.

In [6] Broder describes two approaches for creating efficient signatures. The first uses a (pseudo-)random total ordering on the set of all values, and takes as signature the s smallest values in each set, for some constant s. It is important to note that the same ordering is used for all sets. Then the signatures $S(A), S(B)$ of two sets A, B can be compared as follows: For $u := \min(\max(S(A)), \max(S(B)))$ we have

$$S(A) \cap S(B) = \{x \in A \cap B \mid x \leq u\}$$

Thus any value above u in $S(A), S(B)$ is 'useless' for determining intersection sizes, so the *effective* signatures used for comparing A and B are

$$S'(A) := \{x \in S(A) \mid x \leq u\}$$
$$S'(B) := \{x \in S(B) \mid x \leq u\}$$

We can then interpret results about the intersection size $|S'(A) \cap S'(B)| = |S(A) \cap S(B)|$ as randomly selecting $|S'(A) \cup S'(B)|$ distinct values from $A \cup B$, and getting $|S'(A) \cap S'(B)|$ values in $A \cap B$. Note that although the signature size is fixed by s, the effective signature size varies.

The second approach in [6], originally proposed in [8], takes as signature all values having a hash value that is divisible by a constant m. Here we have

$$S(A) \cap S(B) = \{x \in A \cap B \mid x = 0 \mod m\}$$

We can again see this as randomly selecting $|S(A) \cup S(B)|$ distinct values from $A \cup B$. However, the approach becomes troublesome when sets vary greatly in size, which is often the case. Then a small constant m generates large signatures for large sets and thus high transport and computation costs, while a large constant generates small signatures for small sets, making estimates for their intersection size inaccurate.

To address this problem, Broder proposes a variant of the second approach, where the value $m = 2^i$ for some i depends on the size of the value set, chosen to generate signatures of roughly equal size. This allows for more accurate comparison of small sets without causing signatures of large sets to grow too large. However, when comparing two signatures which use different constants i_A, i_B, then only the values which are $0 \mod 2^{\max(i_A,i_B)}$ can contribute to the intersection, so we have again 'useless' values, and effective signatures

$$S'(A) := \{x \in S(A) \mid x = 0 \mod 2^{i_B}\}$$
$$S'(B) := \{x \in S(B) \mid x = 0 \mod 2^{i_A}\}$$

The methods of signature generation discussed so far can all be performed quickly in near-linear time, and comparing two signatures with respect to intersection size amounts to randomly selecting $|S(A) \cup S(B)|$ or $|S'(A) \cup S'(B)|$ distinct values from $A \cup B$. In [5] a different approach is used known as 'min hashing', originally proposed by Broder in [9] for finding near-duplicate documents. Instead of taking as signature the s smallest elements w.r.t. a single random ordering, the signature consists of the minimum values w.r.t. s different random orderings (hash functions). Then given the signatures for two sets A, B, the minimal value $\min_\sigma(A \cup B)$ in $A \cup B$ w.r.t. an ordering σ can be computed as $\min_\sigma(\min_\sigma(A), \min_\sigma(B))$. This value lies in $A \cap B$ iff $\min_\sigma(A) = \min_\sigma(B)$. Thus we are effectively selecting a single value from $A \cup B$ at random for each of the s random orderings. A nice property of this sampling approach is that one can easily group minimal values into blocks, and then compare signatures block-wise by reducing each block to a single hash value [9]. This is a trade-off between speed and accuracy, which is very favourable for finding pairs with high resemblance threshold (say 0.9), but unfortunately not for low resemblance thresholds (say 0.1), which are needed for attribute comparison.

Signature size is constant for min hashing, and at the same time we have no 'useless' elements. However, we are effectively using only one element of every pair of elements $\min_\sigma(A)$, $\min_\sigma(B)$, which gives us an effective sample size equal to one of the signatures, whereas the approaches in [6] provide effective sample sizes at least as large as one of the signatures, and usually larger. Furthermore, multiple independent sampling has a slightly larger variance than selecting the full sample set at once (without chance of duplicates). Finally, computation of the signature can take longer than the approaches proposed in [6], as we need to compare elements w.r.t. s different, independent orderings. For these reasons we consider the signature generation methods in [6] preferable for our purposes. We will use and extend 'mod 2^i'-sampling, since computation of effective signature size is slightly easier and faster than for Broder's first 'min'-sampling technique, but note that 'min'-sampling can be extended in a similar manner as well.

While we only discussed comparison of two attributes A, B, it is easy to see that $S(A)$ depends only on A, with no knowledge of B required. Thus, when comparing three or more attributes, we still only need to sample each attribute once, using the same sample $S(A)$ to compare A to B and to C.

3. SET-OF-SETS SIMILARITY

Resemblance is a measure for value set similarity which works well for reasonably clean data sets where data formats are uniform. However, there are many cases where values of different string-attributes don't match exactly, but are similar, and we would like to treat them as matching, considering small differences to be due to coding errors, i.e., dirty data. One way to deal with this is to replace each string by a set of *chunks* - e.g. words, q-grams (substrings of fixed length [10]) or possibly v-grams (substrings of variable length [11]) occurring in it, and then to compare these sets of chunks. A method for converting strings into sets of chunks is called a *chunking strategy*. A very simple way for comparing attributes based on chunks is to form for each attribute the set (or multi-set) of chunks occurring in any of its values, and then compare these sets. Such an approach has been suggested in [5]. Technically this can be done exactly as before - the only difference is that we are now comparing sets of chunks instead of sets containing the original values. While this gives us a rough estimate on whether two value sets have something in common, it is not very accurate since correlations between chunks (i.e., whether they originate from the same string value) are not taken into account.

EXAMPLE 1. *Consider the following three (very small) value sets, corresponding to three different attributes:*

A_1 :	*James Bond*
	Jason Bourne

A_2 :	*Bond James*
	Bourne Jason

A_3 :	*Bourne James*
	Bond Jason

If we form their 3-gram sets as described above (using only substrings of words), we get

A_1 :{*Jam,ame,mes,Bon,ond,Jas,aso,son,Bou,our,urn,rne*}
A_2 :{*Bon,ond,Jam,ame,mes,Bou,our,urn,rne,Jas,aso,son*}
A_3 :{*Bou,our,urn,rne,Jam,ame,mes,Bon,ond,Jas,aso,son*}

Here A_2 and A_3 map to exactly the same sets of chunks, and thus appear to be identical and equally similar to A_1, although A_2 should really be considered closer to A_1 and somewhat different from A_3.

What we propose is a new measure for attribute similarity. For this we also map each string value to the set of its chunks, but don't combine all chunk sets into one. For the tables from Example 1 we then get

A_1 :{{Jam,ame,mes,Bon,ond}, {Jas,aso,son,Bou,our,urn,rne}}
A_2 :{{Bon,ond,Jam,ame,mes}, {Bou,our,urn,rne,Jas,aso,son}}
A_3 :{{Bou,our,urn,rne,Jam,ame,mes}, {Bon,ond,Jas,aso,son}}

We therefore have a new problem to solve: instead of measuring similarity between sets, where elements are just equal or not, we measure similarity between sets-of-sets, where

elements of the inner sets (the chunks) are compared for equality. Note that as in the example above, different (but usually quite similar) values can have identical chunk sets. This may be an advantage or a disadvantage, depending on whether such values are considered to be equivalent.

At this point one may wonder why it isn't possible to just apply existing sampling techniques as discussed in Section 2.1, and then compare samples using some similarity measure for strings. The reason is that similar (but not identical) pairs of strings are just as likely to appear in samples as dissimilar ones. Thus for dirty data without any (or very few) exact matches, the sampling approaches discussed effectively become independent random samples, which are painfully ineffective for identifying shared values.

3.1 Set-of-Sets Sampling

When selecting samples for measuring set intersection, the synchronized sampling approaches discussed in Section 2.1 ensured that an element from A, B either gets selected for both samples (if present), or neither:

$$x \in S(A) \text{ and } x \in B \quad \Rightarrow \quad x \in S(B)$$

That condition, as opposed to independent random sampling, allowed us to obtain accurate estimates with small samples. When selecting samples for sets-of-sets, we must now also consider similar elements. That is, when $x \in A$ is similar to $y \in B$ (written as $x \sim y$), we would like for either both or neither of them to appear in their respective samples:

$$x \in S(A) \text{ and } x \sim y \in B \quad \overset{\text{'probably'}}{\Rightarrow} \quad y \in S(B)$$

For this we developed a sampling method where the probability of both x, y appearing in the respective samples will depend on the similarity of x and y.

Definition 2 (SoS-sample). *Let A be a set-of-sets over some universe $\mathcal{U}$, $\sigma \subseteq \mathcal{U} \times \mathcal{U}$ a linear ordering, and $\mathcal{V} \subseteq \mathcal{U}$. The* Set-of-Sets-sample *(SoS-sample) of A w.r.t. $\sigma, \mathcal{V}$ is*

$$SoS(A) := \{a \in A \mid min_\sigma(a) \in \mathcal{V}\}$$

In the context of database integration, think of $\mathcal{U}$ as the set of all strings. For practical purposes, we will simply use a hash function H to map $\mathcal{U}$ to $\mathbb{N}$, as in [6, 5]. Then σ is induced by H using the natural ordering of $\mathbb{N}$, that is, $x \leq_\sigma y$ iff $H(x) \leq H(y)$, and $\mathcal{V} := \{u \in \mathcal{U} \mid H(u) = 0 \mod n\}$. Just as for 'mod n'-sampling in the context of sets, n can be fixed or 2^i where i depends on the set size.

As the following theorem shows, the probability of both a, b appearing in their respective SoS-samples does indeed depend directly on their similarity.

Theorem 1. *Let $\mathcal{U}$ be finite and σ chosen uniformly at random and independent from $\mathcal{V}$, which is created by selecting each element in $\mathcal{U}$ independently with probability p. Then for $a \in A, b \in B$ the probability that a, b have the same minimal element w.r.t. σ and are both sampled is*

$$P(min_\sigma(a) = min_\sigma(b) \in \mathcal{V}) = p \cdot res(a, b)$$

When transmitting SoS-samples for identifying similar attributes, it is not always space-efficient to use sets of chunks, as chunks may overlap (depending on the chunking strategy used). Instead, we will send the original string values from which the chunk sets were generated, and reconstruct the chunk sets after transmission. As an added bonus, this allows us to use the same signatures to estimate set similarity, as described in Section 2, since the chunk sets of identical values from different value-sets A, B appear in both SoS-samples if they appear in one. Also, samples consisting of original values can be used for later human inspection of attribute pairs where automated similarity results aren't decisive, or for other automatic similarity measures operating on strings rather than sets-of-sets (in particular oracle-resemblance, as described in Section 5).

3.2 IR-Sum

We are looking for a similarity measure which can be approximated using samples of small size. In particular, if we want to allow unbiased estimates using sample size one, our measure for sets (of sets) A, B must take the form

$$F(A, B) = \sum_{\substack{a \in A \\ b \in B}} f(a, b) \tag{1}$$

for some function $f(a, b)$. While there is no inherent requirement to allow samples of size one (which would be extremely unreliable), equation (1) still gives us an idea for how to construct our similarity measure in general.

Intuitively, to obtain a similarity measure for A, B, the function $f(a, b)$ should measure the similarity between a and b. Perhaps the most simple 'similarity' measure is equality: $f(x, x) = 1,\ f(x, y \neq x) = 0$. This results in $F(A, B) = |A \cap B|$. Another very simple measure is $f(a, b) = |a \cap b|$. Closer inspection reveals that this simply defines a measure between the bag-unions $\biguplus A, \biguplus B$, ignoring correlations:

$$F(A, B) = \sum_{x \in (\bigcup A) \cap (\bigcup B)} mult(x, \biguplus A) \cdot mult(x, \biguplus B)$$

where $mult(x, \biguplus A)$ denotes the multiplicity of x in the bag $\biguplus A$. Using $f(a, b) = res(a, b)$ leads to similar results. While the induced measure F doesn't ignore correlations completely, they still have rather low impact (e.g. for the sets of Example 1 we get $F(A_1, A_2) = 2, F(A_1, A_3) = 1.32$, which shows that A_1 and A_2 are more similar, but the difference is small). The basic measure we propose is the following:

Definition 3 (IR-Sum). *Given two sets-of-sets A, B, the* Intersection-Resemblance-Sum *of A, B is*

$$IR\Sigma(A, B) := \sum_{a \in A, b \in B} |a \cap b| \cdot res(a, b) = \sum_{a \in A, b \in B} \frac{|a \cap b|^2}{|a \cup b|}$$

We shall also refer to it as IR-Sum for short.

Applying this to the attributes from Example 1 we get

$$\text{IR}\Sigma(A_1, A_2) = \frac{5^2}{5} + \frac{0^2}{12} + \frac{0^2}{12} + \frac{7^2}{7} = 12$$

$$\text{IR}\Sigma(A_1, A_3) = \frac{3^2}{9} + \frac{2^2}{8} + \frac{4^2}{10} + \frac{3^2}{9} = 4.1$$

which indicates that A_1, A_2 are much closer than A_1, A_3, as we would expect. We will discuss scaling to $[0 \ldots 1]$ to obtain a similarity measure later on.

Of course, there are many other ways one could define set-of-sets similarity. Suitable candidates include $\sum |a \cap b|^n$ or $\sum res(a, b)^n$ for $n \geq 2$, to name just a few. All of them

could sensibly distinguish cases like those in Example 1. So what makes IR-Sum stand out? The answer is that we can approximate it easily and efficiently.

3.3 Approximating IR-Sum

Exact computation of the IR-sum is expensive, and requires access to the whole value sets (while complexity is 'only' quadratic, this can be prohibitive for large data sets). Instead we will estimate it using SoS-samples:

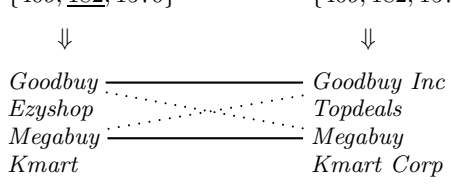
DEFINITION 4 (SOS-SUM). *The* Set-of-Sets-sum *of A, B (short SoS-sum) w.r.t. $\sigma, \mathcal{V}$ is*

$$SoS\Sigma(A,B) := \sum_{\substack{a \in A, b \in B \\ min_\sigma(a) = \\ min_\sigma(b) \in \mathcal{V}}} |a \cap b|$$

EXAMPLE 2. *Consider the value sets of retailers below. SoS-samples are created by splitting each string into chunks and mapping them to integers using a fixed random function. The minimal element is used to check whether to sample the value, and which other values to compare it to. In this example, we selected all values where the minimal element is even (0 mod 2^1).*

As indicated by the full and dotted lines, both matching and non-matching sample values can map to the same minimal element, and thus contribute to the SoS-sum, but the probability for non-matching values is lower (assuming they have lower resemblance). Also, it can happen that matching values ('Kmart' and 'Kmart Corp') both appear in the sample set, but map to different minimal elements. We don't compare such values since it would raise computation costs significantly (several orders of magnitude for large sets[1]) with little benefit, since the vast majority of such value pairs will be non-matching.

Goodbuy ↦ 134 {*Goo*, *ood*, *odb*, *dbu*, *buy*} {637, 524, 166, 373, <u>134</u>}	*Goodbuy Inc* ↦ 134 {*Goo*, *ood*, *odb*, *dbu*, *buy*, *Inc*} {637, 524, 166, 373, <u>134</u>, 225}
Payless ↦ 51 {*Pay*, *ayl*, *yle*, *les*, *ess*} {<u>51</u>, 907, 804, 1206, 281}	*Payless Ltd* ↦ 51 {*Pay*, *ayl*, *yle*, *les*, *ess*, *Ltd*} {<u>51</u>, 907, 804, 1206, 281, 335}
Ezyshop ↦ 102 {*Ezy*, *zys*, *ysh*, *sho*, *hop*} {293, 647, <u>102</u>, 388, 815}	*Topdeals* ↦ 128 {*Top*, *opd*, *pde*, *dea*, *eal*, *als*} {469, 807, 411, <u>128</u>, 909, 583}
Megabuy ↦ 134 {*Meg*, *ega*, *gab*, *abu*, *buy*} {197, 764, 562, 233, <u>134</u>}	*Megabuy* ↦ 134 {*Meg*, *ega*, *gab*, *abu*, *buy*} {197, 764, 562, 233, <u>134</u>}
Kmart ↦ 182 {*Kma*, *mar*, *art*} {409, <u>182</u>, 1570}	*Kmart Corp* ↦ 94 {*Kma*, *mar*, *art*, *Cor*, *orp*} {409, 182, 1570, <u>94</u>, 1347}



[1]This increase is only in part caused by the extra value comparisons. Equally significant are the need to explicitly calculate resemblance between chunk sets, and loss of the minbag optimization described in Section 3.5.2.

We can now use SoS-sum, which can be computed from samples, to obtain an unbiased estimate for the IR-sum.

THEOREM 2. *Let samples be generated as in Theorem 1, with p as sampling probability. Then the approximation*

$$IR\Sigma(A,B) \approx \frac{SoS\Sigma(A,B)}{p} \tag{2}$$

is an unbiased estimate.

3.4 Scaling to SoS-resemblance

Bigger sets generally have bigger IR-sums, so e.g. a result of $\mathrm{IR}\Sigma(A,B) = 127.5$ alone does not really tell us whether or not A, B are very similar. Just as intersection size must be scaled to obtain a similarity measure like set resemblance, we must scale IR-sum to obtain a more meaningful value in $[0 \dots 1]$. Let us begin with the following observation:

PROPOSITION 1. *Let A, B be sets-of-sets such that for all $a \in A, b \in B$ we have $|a| = |b|$ and either $a = b$ or $a \cap b = \emptyset$. Then with $n := |a| = |b|$ we have $IR\Sigma(A,B) = n \cdot |A \cap B|$.*

Thus, up to a fixed factor n, we have the correspondences

$$|A \cap B| \leftrightarrow \mathrm{IR}\Sigma(A,B)$$
$$|A \cup B| \leftrightarrow \mathrm{IR}\Sigma(A,A) + \mathrm{IR}\Sigma(B,B) - \mathrm{IR}\Sigma(A,B)$$

While these correspondences only hold for sets meeting the conditions of Proposition 1, they can provide us with a generalized definition for set-of-set resemblance.

DEFINITION 5 (SOS-RESEMBLANCE). *We define the* Set-of-Set-resemblance*, short SoS-resemblance, as*

$$SoS\text{-}res(A,B) := \frac{IR\Sigma(A,B)}{IR\Sigma(A,A) + IR\Sigma(B,B) - IR\Sigma(A,B)}$$

It remains to show that this definition provides us with a similarity measure for arbitrary sets A, B.

DEFINITION 6. *A function $f : U \times U \to \mathbb{R}$ is a* similarity measure *on U if*

(i) $f(a,b) \in [0 \dots 1]$ *for all* $a, b \in U$

(ii) $f(a,b) = 1$ *iff* $a = b$

THEOREM 3. *Let $\mathcal{P}^+(U)$ denote the set of all finite, non-empty subsets of U. SoS-resemblance is a similarity measure on $\mathcal{P}^+(\mathcal{P}^+(U))$.*

We can now estimate SoS-resemblance using our approximations for IR-sum:

$$\text{SoS-res}(A,B) \approx \frac{\mathrm{SoS}\Sigma(A,B)}{\mathrm{SoS}\Sigma(A,A) + \mathrm{SoS}\Sigma(B,B) - \mathrm{SoS}\Sigma(A,B)}$$

It is important to note that for this estimate we assume that the sampling probability p is the same for $\mathrm{SoS}\Sigma(A,A)$, $\mathrm{SoS}\Sigma(B,B)$, and $\mathrm{SoS}\Sigma(A,B)$. However, we may use different probabilities of the form $p = 2^{-i}$ for different attributes. This mismatch can be resolved by considering only the effective samples, as discussed in Section 2.1.

EXAMPLE 3. *For the attribute sets from Example 1 we get*

$$SoS\text{-}res(A_1,A_2) = \frac{12}{12+12-12} = 1 \ (100\%)$$
$$SoS\text{-}res(A_1,A_3) = \frac{4.1}{12+12-4.1} \approx 0.2 \ (20\%)$$

which allows us to clearly distinguish them.

3.5 Efficient Computation

While sampling is central in making our method scale, efficient analysis of samples is important as well.

3.5.1 Combined Similarity Computation

After computing the signature for each attribute and sending them to a central location, we now need to analyze them. For each pair of signatures $S(A), S(B)$ we need to determine their intersection size and/or IR-sum, from which we can then estimate their (SoS-)resemblance. For intersection size and IR-sum we need not worry about 'effective' signatures. This is because for all the signature generation methods in [6] (respectively SoS-sampling), we have the equalities

$$S'(A) \cap S'(B) = S(A) \cap S(B)$$
$$\text{SoS}\Sigma(S'(A), S'(B)) = \text{SoS}\Sigma(S(A), S(B))$$

Pairwise comparison of attributes is inefficient though, since each value is processed multiple times. To avoid unnecessary comparisons, we identify for each value the set of all attributes containing it (for dirty data this is done similarly, as described in Section 3.5.2), and then increase the intersection size for each pair in that set. Perhaps the easiest way to do so is to create an inverted index by value over all attributes. We followed this approach for simplicity of presentation and implementation. For medium-sized data sets with up to a few thousand attributes (value set size is (mostly) irrelevant since we use samples), memory consumption wasn't a problem. For larger data sets, sorting based approaches [7, 12] can be used.

In scenarios where many identical or near-identical value-sets are expected, e.g. due to data duplication, clustering methods (e.g. [7, 9]) can improve performance. We will not discuss this further here.

3.5.2 Minbag-Sum

We now look for ways of making computation of SoS-sum more efficient. For that, consider the 'Goodbuy' and 'Megabuy' values from Example 2. Since they both map to the same minimal element, they will always be compared to the same values from other attributes. We thus combine their chunk sets to reduce the number of comparisons.

DEFINITION 7 (MINBAG). *Let $A, \mathcal{U}, \sigma, \mathcal{V}$ be as in Definition 2. For each $x \in \mathcal{U}$ the* minbag *of x, A w.r.t. σ is the bag (multiset)*

$$mb(x, A) := \biguplus\{a \in A \mid min_\sigma(a) = x\}$$

The minbag-signature *of A is then the set*

$$MBSig(A) := \{(x, mb(x, A)) \mid x \in \mathcal{V} \wedge mb(x, A) \neq \emptyset\}$$

For two sets-of-sets A, B over $\mathcal{U}$, the minbag-sum *of A, B is*

$$MB\Sigma(A, B) := \sum_{x \in \mathcal{V}} \sum_{y \in \mathcal{U}} mult(y, mb(x, A)) \cdot mult(y, mb(x, B))$$

Here $mult(y, mb(x, A))$ denotes multiplicity of y in $mb(x, A)$.

THEOREM 4. *$MB\Sigma(A, B) = SoS\Sigma(A, B)$.*

Minbag-signatures can be derived from SoS-samples, and given the minbag-signatures of attributes it is possible to compute their minbag-sum. By avoiding unnecessary comparisons, this can be done quite efficiently for large sets of attributes simultaneously, as described in Algorithm 1.

Algorithm 1 Compute Minbag-Sum

Input: $\text{MBSig}(A_i)$ for $i = 1 \ldots n$
Output: $\text{MB}\Sigma(A_i, A_j)$ for $i, j = 1 \ldots n$

```
 1: MBΣ(∗, ∗) := 0
 2: min-index(∗) := ∅
 3: for all A_i do
 4:    for all (x, mb) ∈ MBSig(A_i) do
 5:       min-index(x) := min-index(x) ∪ {A_i}
 6: for all x with min-index(x) ≠ ∅ do
 7:    elem-index(∗) := ∅
 8:    for all A_i ∈ min-index(x) do
 9:       for all y ∈ mb(x, A_i) do
10:          elem-index(y) := elem-index(y) ∪ {A_i}
11:    for all y with elem-index(y) ≠ ∅ do
12:       for all A_i, A_j ∈ elem-index(y) do
13:          MBΣ(A_i, A_j) += mult(y, mb(x, A_i)) ·
                              mult(y, mb(x, A_j))
14: return MBΣ
```

We first index all attributes by the min-values x in their minbag-signatures. Then for each value $x \in \mathcal{V}$ we construct another inverted index by elements of $\mathcal{U}$, i.e., chunks (or their hash-values). Just as for set-intersection, we then consider for each $y \in \mathcal{U}$ in the index all pairs of attributes A, B indexed under y and add $mult(y, mb(x, A)) \cdot mult(y, mb(x, B))$ to $\text{MB}\Sigma(A, B)$. Let size(MBSig) denote the total number of elements (chunks) in $\text{MBSig}(A_1), \ldots, \text{MBSig}(A_n)$.

THEOREM 5. *Algorithm 1 terminates in time*

$$O(size(MBSig) \cdot n)$$

Note that this worst-case bound is reached when all (or a fixed percentage) of sets-of-sets are similar. However, in such cases, the number of positive minbag-sums returned is quadratic in n, so that linear complexity in the input size is impossible. If we impose a bound on the size of each minbag-signature (which is practical for attribute samples), Algorithm 1 becomes output-linear.

Extending an existing similarity graph to include new attributes can easily be done in an efficient manner. All we need to do is to compute signatures for the new attributes, and compare them amongst themselves and to all existing signatures. For this Algorithm 1 can simply be modified to only loop over attribute pairs A_i, A_j in line 12 where at least one attribute is new. By processing new attributes first, it is possible to immediately drop 'useless' entries during index construction, thus reducing memory requirements.

Instead of using inverted indices, it is also possible to adapt sorting-based approaches such as the 'Spider' Algorithm from [12], which require less memory.

To summarize, we employ the same general framework as Bellman [5], but with different methods for sample creation and analysis. For sampling we use SoS-samples as described in Section 3.1. During the analysis phase, these are then converted into minbag-signatures and compared using Algorithm 1, to obtain the minbag-sum of all attribute pairs. From these we can then estimate their SoS-resemblance.

4. VARIATIONS

In the following we will discuss some variations of IR-sum, each addressing certain weaknesses of it. The options proposed can be applied in any combination.

4.1 Reducing Mismatches

One problem inherent to every similarity measure, is that non-matching values will often display some similarity. For SoS-resemblance, this issue is particularly important, since pair-wise comparison of values (which was necessary to enable sampling) causes large numbers of non-matching value pairs to contribute to our measure. One way to filter out such pairs is to increase the q-gram length (if we use q-gram splitting as chunking strategy). However, this has the undesirable side-effect that small variations in matching string values will have larger impacts on their resemblance. For example, the strings 'Jason Bourne' and 'Jason Boyrne' have resemblances of $\frac{9}{13}$, $\frac{7}{13}$ and $\frac{5}{13}$ for 2-grams, 3-grams and 4-grams, respectively.

An alternative solution is to ignore all pairs which match in only a single element. This can be done by reducing the intersection size in the IR-sum definition by one:

DEFINITION 8 (RIR-SUM). *We define the* reduced intersection size *of two sets a, b as*

$$ris(a,b) := \begin{cases} |a \cap b| & \text{if } |a| = |b| = 1 \text{ or } a \cap b = \emptyset \\ |a \cap b| - 1 & \text{otherwise} \end{cases}$$

The Reduced Intersection-Resemblance-sum *(RIR-sum) of two sets-of-sets A, B is*

$$RIR\Sigma(A,B) := \sum_{a \in A, b \in B} ris(a,b) \cdot res(a,b)$$

For definition of reduced intersection size, the first case is necessary to ensure that identical singular sets contribute to the sum, and to avoid negative summands. The underlying idea for RIR-sum is that whenever a pair of values shares two or more q-grams, then this is most likely because they share a larger (q+k)-gram (this intuition becomes more accurate with growing q). As a consequence, using RIR-sum instead of IR-sum reduces the contribution of non-matching values (which typically share few q-grams) nearly as much as increasing q-gram size. On the other hand, the contribution of matching elements suffers only a minor reduction.

Computing RIR-sum (or rather, the corresponding reduced SoS-sum) is even faster than IR-sum (SoS-sum). For this, observe that we only compare sets $a \in A, b \in B$ with the same minimal element x. Consequently, they always share at least this minimal element x, so the reduction in intersection size can be performed by not adding x to $\text{mb}(x, A)$ and $\text{mb}(x, B)$, except for sets containing *only* x (first case in Definition 8).

4.2 Reducing Multi-matches

By definition of IRΣ, a tuple in A can be matched with multiple tuples in B (all of which have the same minimal element). This has the undesirable effect that tuples containing many common chunks can become the main contributers to the measure. In other words, chunks which occur frequently and thus carry little information are more important than chunks occurring rarely (and thus carry more informational value). One way to approach this problem (which we did experiment with) is to assign weights to chunks based on their frequency. However, we found a method which is not only simpler, but also provides more accurate results, and has a number of other desirable properties.

Our modification is simple: When approximating IR-sum, instead of computing the minbag-sum

$$\text{MB}\Sigma(A,B) := \sum_{x \in \mathcal{V}} \sum_{y \in \mathcal{U}} \begin{matrix} mult(y, mb(x,A)) \cdot \\ mult(y, mb(x,B)) \end{matrix}$$

we ignore multiplicity and use minsets instead of minbags:

$$ms(x,A) := \bigcup \{a \in A \mid min_\sigma(a) = x\}$$

From this we can then compute the *minset-sum*

$$\text{MS}\Sigma(A,B) := \sum_{x \in \mathcal{V}} |ms(x,A) \cap ms(x,B)|$$

The result is a measure where each chunk in A (respectively B) contributes to the measure at most once, less if it occurs in many (similar) sets in A, and more if it appears in many (dissimilar) sets in B.

In particular, one nice property of the resulting measure is that it is stable w.r.t. duplicates in A: if a set-of-chunks appears several times in A, these are eliminated when constructing minsets. While one could immediately object that A was defined as a set, so duplicates can't occur in the first place, A may still contain near-duplicates, e.g. when a value is entered twice with slightly different spellings. Such near-duplicates are also treated appropriately for the most part. That is, in all likelihood (if they possess the same minimal q-gram, i.e., with a probability equal to their resemblance), the duplicate chunks are eliminated.

Furthermore, since every chunk contributes to the measure at most once, we can use it to not only measure 'dirty' resemblance, but also 'dirty' subset containment. Classic subset containment for clean data is measured as $\frac{|A \cap B|}{|A|}$. However, the corresponding IR-sum fraction $\frac{\text{IR}\Sigma(A,B)}{\text{IR}\Sigma(A,A)}$ is not a suitable measure, since $\text{IR}\Sigma(A,B)$ can be bigger than $\text{IR}\Sigma(A,A)$. For $\text{MS}\Sigma(A,B)$ and $\text{MS}\Sigma(A,A)$ this can not happen, so that the resulting fraction always lies in $[0,1]$.

DEFINITION 9 (SOS-CONTAINMENT). *Let A, B be sets-of-sets. We define the* set-of-sets-containment *measure as*

$$SoS\text{-}con(A,B) := E_{MS\Sigma(A,A) \neq 0}\left(\frac{MS\Sigma(A,B)}{MS\Sigma(A,A)}\right)$$

It is clear by definition that $\frac{\text{MS}\Sigma(A,B)}{\text{MS}\Sigma(A,A)}$ (with re-sampling if the sample for A is empty) is an unbiased estimate for SoS-containment. Furthermore, SoS-containment extends the classic subset-containment measure $\frac{|A \cap B|}{|A|}$: if every set is a singleton (contains only the original value) then the two measures become identical.

5. ORACLE-RESEMBLANCE

Unfortunately, while SoS-resemblance can quickly provide us with a rough initial estimate whether (and how much) two dirty value-sets overlap, we found that (especially for large sets) the resemblance estimate wasn't as close to the actual resemblance as one may hope for (Section 6). To improve on this, we propose a two-step process. In the first step, SoS-resemblance is used to eliminate most unrelated attribute pairs from consideration. This can be done quickly even for large data sets. In a second refinement step, we then employ more accurate (but slower) estimation functions to the remaining candidate matches. We will now investigate how

more accurate measures can be designed to allow (reasonably) efficient estimates based on the same SoS-signatures we used to compute SoS-resemblance. Two problems that limit the accuracy of SoS-resemblance are the following:

- Unrelated values which have only low similarity can still increase the overall SoS-resemblance. This effect increases with shorter chunks and larger value sets, but can be mitigated by using min-sets (instead of min-bags) as described in Section 4.2.
- Equivalent but not identical ("dirty") values don't increase the overall SoS-resemblance as much as they should. This effect increases with larger chunks and higher error (typo) frequency.

Ideally, unrelated values should contribute nothing to the overall similarity, while equivalent values should contribute fully, i.e., as much as identical values. Let us assume for a moment that we have some oracle Ω which decides for each pair of string values whether or not they are equivalent. Then we can make use of this oracle to construct a similarity function between dirty value-sets as follows:

DEFINITION 10 (ORACLE-RESEMBLANCE). *Let W denote the universe of all possible values (e.g. all strings) and $\mathfrak{C} : W \to \mathcal{P}(U)$ a chunking strategy mapping values to finite sets of chunks. Then for an oracle function $\Omega : W \times W \to \{0, 1\}$ we define the* oracle-sum *of two value-sets $A, B \subseteq W$ as*

$$\Omega\Sigma(A, B) := \sum_{a \in A, b \in B} \Omega(a, b)$$

and scale it to oracle-resemblance *as*

$$\Omega\text{-}res(A, B) := \frac{\Omega\Sigma(A, B)}{\Omega\Sigma(A, A) + \Omega\Sigma(B, B) - \Omega\Sigma(A, B)}$$

Typically we will employ some string similarity/distance function (e.g. edit-distance) to construct Ω, returning 1 if the similarity/distance between the two values exceeds a predefined threshold. While the result is hardly ever a perfect oracle (i.e., one which makes no mistakes), we found that even simple similarity/distance functions perform quite well in terms of accuracy.

Furthermore, we can easily approximate oracle-resemblance using our SoS-sample[2]. For this we only need to compensate for the reduced probability of two equivalent but different values having the same minimal chunk.

DEFINITION 11. *Let $A \subseteq W$. The* chunk-based *sample $S_{\mathfrak{C}}(A)$ of A contains the original values of the SoS-sample (Definition 2) of $\mathfrak{C}(A)$:*

$$S_{\mathfrak{C}}(A) := \{a \in A \mid min_\sigma(\mathfrak{C}(a)) \in V\}$$

The set-of-sets-oracle-sum *is*

$$SoS\text{-}\Omega\Sigma(A, B) := \sum_{\substack{a \in S_{\mathfrak{C}}(A) \\ b \in S_{\mathfrak{C}}(B) \\ min_\sigma(\mathfrak{C}(a)) = \\ min_\sigma(\mathfrak{C}(b))}} \frac{\Omega(a, b)}{res(\mathfrak{C}(a), \mathfrak{C}(b))}$$

Recall from Section 3.1 that we actually transmit chunk-based samples, rather than SoS-samples. Just as SoSΣ approximates IRΣ (Theorem 2), SoS-$\Omega\Sigma$ approximates $\Omega\Sigma$.

[2]To be precise, we use the values from which our SoS-sample is constructed by applying the chunking strategy $\mathfrak{C}$.

THEOREM 6. *Let Ω be* compatible *with $\mathfrak{C}$, i.e.,*

$$\Omega(a, b) = 1 \quad \Rightarrow \quad \mathfrak{C}(a) \cap \mathfrak{C}(b) \neq \emptyset$$

for all $a, b \in W$. Then

$$\Omega\Sigma(A, B) \approx \frac{SoS\text{-}\Omega\Sigma(A, B)}{p}$$

is an unbiased estimate.

Note that oracle-resemblance is not restricted to comparing sets of strings - we can compare sets of arbitrary data items. All we require is a chunking strategy (for sets-of-sets this could simply be the identity mapping) and a compatible oracle function.

6. EXPERIMENTAL RESULTS

We applied our methods to two publicly available data sets, the WildFinder database (www.worldwildlife.org), and a collection of medical data sets (www.medicare.gov). Our main goals here were to establish accuracy and efficiency of our approach (and existing ones for comparison).

6.1 Accuracy

In order to test accuracy of our sos-similarity measure, we picked one attribute "common_names" (containing animal names) from the WildFinder database, as well as two attributes "corp_name" and "NursingHomeName" from the medical data set. We chose these value sets simply because they contain real string data (as opposed to e.g. zip codes stored as strings), and because they were large enough (15,000-35,000 distinct values).

To obtain a controlled amount of dirtiness necessary to judge the accuracy of different measures in an objective and easily quantifiable manner, we first created 'clean' sets with known resemblance by selecting tuples from the value set uniformly at random. Afterwards we introduced small errors into each value[3] of one data set. Here we considered different types of errors (for each error introduced, the type was chosen uniformly at random):

- INS: insert random character at random position
- DEL: delete character at random position
- REP: replace character at random position
- SWP: swap adjacent characters at random position
- PER: permute adjacent words at random position

This 'dirty' data was then compared to the clean data set using SoS-resemblance. We used a rather large (maximal) sample size of 1000 to limit inaccuracies due to sampling. Furthermore, we also computed oracle resemblance, using a simple oracle function which returns 1 whenever the two strings have an edit-distance of 3 or less, or a 2-gram resemblance of 0.8 or more.

For comparison with existing methods, we measured q-gram-resemblance, i.e., resemblance of the sets of all q-grams as proposed in [5] for the Bellman System. Another approach we will briefly investigate are 'data sketches', also

[3]While it is unlikely that a real data set would contain typographical errors in every value, small differences in formatting can easily cause situations where all matching values are represented by slightly different strings.

used in [5]. Here multi-sets of q-grams are regarded as vectors over the space of all q-grams, and attributes are compared via the euclidian distance of their normalized vectors. As far as we know, these are the only sample-based similarity measures for 'dirty' sets of strings proposed to date.

Naturally there are a number of parameters which may impact on the result. Table 1 lists the different parameters we vary and their default setting.

parameter	range	default
error number	1 - 4	1
value set size	100 - 5,000	1,000
q-gram length	2 - 4	3

Table 1: Test Parameter Settings

For SoS-resemblance, we report results with different variants applied, namely IRΣ, RIRΣ, MSΣ and RMSΣ (which combines both variants).

Figure 1 shows the different resemblance measures and sketch-distance graphs for different data sets. Unfortunately, none of the measures for initial matching (i.e., all measures except oracle-resemblance) return results very close to the actual (clean) resemblance.

To measure accuracy, we consider two components:

- the *range* r of a measure: the difference between results for clean resemblances 0 and 1
- the *offset* Δ of a measure: the result for clean resemblance 0

Most desirable would be a large range and a small offset. As all similarity measures are fairly linear (as shown in Figure 1 for the default case, and observed for other configurations as well), it seems appropriate to ignore intermediate results.

To combine our two accuracy components into a single measure, which simplifies presentation (without providing an unfair comparison - in almost all cases measures with larger range also had a lower offset), we use

$$\text{accuracy} := \frac{r^2}{r+\Delta}$$

As a results, any constant measure (range 0) will have accuracy 0, while the perfect "clean resemblance" measure (with range 1, offset 0) has accuracy 1. For any positive range, smaller offsets mean higher accuracy, while for any offset (including offset 0) larger ranges mean higher accuracy.

Note that sketch-distance doesn't quite fit this measure: a small distance indicates similar rather than dissimilar sets, and results are in $[0, \sqrt{2}]$ rather than $[0, 1]$. We therefore scale it accordingly for the purpose of measuring accuracy:

$$\text{sketch-res}(A, B) := 1 - \frac{\text{sketch-dist}(A, B)}{\sqrt{2}}$$

Finally, we computed the different similarity measures and their accuracy. In each experiment we varied one parameter within the range described in Table 1, while keeping the other parameters at their default value. The results are given in Figures 2.

The first thing we observe is that oracle-resemblance is the clear winner in all configurations. The next best measure (in terms of accuracy) is minset-based resemblance. Standard SoS-resemblance and q-gram resemblance are both worse, and neither is better than the other in all settings. The RIR-sum modification consistently improves accuracy, albeit not by much. Sketch-based resemblance does not appear to be suitable for distinguishing sets with different resemblance. However, it is important to note that sketch-based resemblance is not useless. It can be used for detecting attributes with the same semantic domain (which is an equally important task), thus complementing the other measures.

Not surprisingly, all measures become less accurate as the number of errors (typos) increases. When varying q-gram lengths, we found that 2-grams were too small to provide good results. 3-grams where slightly more accurate than 4-grams for SoS-resemblance (IR-sum, RIR-sum) and minset-based resemblance (MS-sum, RMS-sum), while 4-grams were best for q-gram resemblance. Oracle-resemblance is unaffected by choice of q-gram length (the variance in accuracy is due to sampling). For value sets of different sizes, we observed that larger value-sets result in lower accuracy. This is true for all measures, although the impact is greatest on standard SoS-resemblance and least on oracle-resemblance. The reason for this is that larger sets (with fixed string length) are more likely to contain very similar but unrelated strings. Thus it can easily happen that dirty but equivalent strings are less similar than unrelated strings, which will necessarily introduce errors.

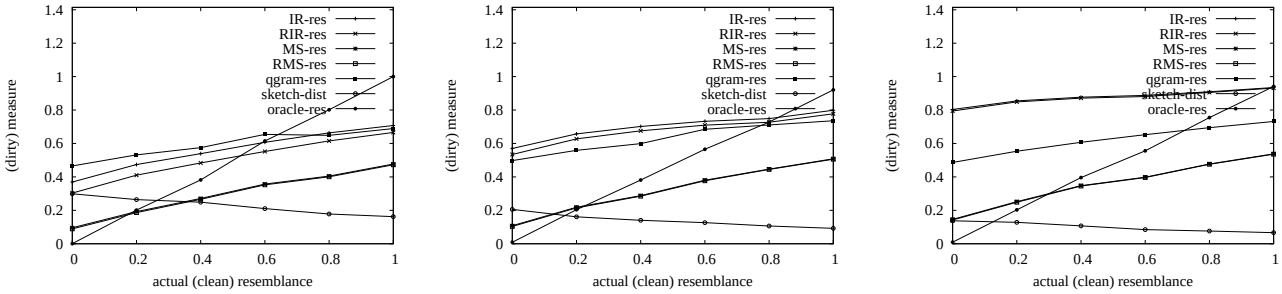
For sampling accuracy, we found that a sample size of about 200 (sets of chunks) was sufficient to keep the standard deviation of SoS-resemblance well below 5%. Using larger sample sets brought no significant improvements to the overall accuracy of our resemblance measures, i.e., the inaccuracies shown in Figure 2 are inherent to the measures, and not due to approximation with samples.

6.2 Efficiency

To establish performance of our methods in terms of computation time, we applied it to a collection of data sets about medical plans, freely available from www.medicare.gov. Together these databases measured over 5 GB in size and contained 587 string-valued attributes (out of 791). We also applied it to the smaller Wildfinder database (35MB, 90 string attributes). Our implementation was in Java, running on a 2.6 GHz PC with data stored in a MySQL database.

We varied the maximal sample size from 200 to 1000 tuples (q-grams for q-gram resemblance, random projections for sketchs). Note that for '$mod\ 2^i$'-sampling we cannot fix the sample size, only provide a maximum (or minimum), with the result that most samples will contain between $\frac{max}{2}$ and max tuples (between min and $2 \cdot min$ tuples for minimum). We found that q-gram length had no significant impact on computation time (results reported are for a q-gram length of 3). Neither had the RIR-sum or minset variant (for SoS-resemblance we report times using basic IR-sum).

Note that we separated extraction of data from the database into dump files and signature generation from the dump files. For creating the dump (which actually was the most time consuming step for the medical data set, as it contained many duplicate values) we just read one table at a time (projected onto all string attributes) and filtered out duplicate values after extraction. Compared to a previous approach of simply using SQL queries of the form 'SELECT DISTINCT attribute FROM table' for each string attribute, this reduced time for dump creation from 14 minutes to just 2 minutes for the medical data set, and from 15 to 4 seconds



(a) animals (b) corporations (c) nursing homes

Figure 1: similarity/distance measures for different data sets (default setting)

for the Wildfinder data set. Time taken for sample creation (from dump files) as well as actual size of the signature file is given in Table 2, for the medical data set (M) as well as the Wildfinder database (W).

	max=200		max=500		max=1000	
	time	size	time	size	time	size
SoS(M)	9.0s	659KB	8.9s	1.1MB	8.7s	1.7MB
q-gram(M)	42s	330KB	42s	654KB	38s	1.1MB
sketch(M)	43s	1.0MB	90s	2.5MB	166s	5.1MB
SoS(W)	0.6s	38KB	0.6s	71KB	0.6s	109KB
q-gram(W)	3.1s	26KB	3.0s	41KB	3.0s	58KB
sketch(W)	3.6s	105KB	6.3s	261KB	12s	519KB

Table 2: Signature Generation

For SoS and q-gram signatures, maximum sampling size had little effect on computation time - if anything, larger sample sizes decreased sampling time slightly. This happens because the main cost lies in reading and hashing the values, and since for larger sample sizes we get more attributes where the number of distinct values lies below the maximal sample size. In such cases we can skip detailed analysis and simply store the entire value set. SoS-sampling is faster than q-gram sampling, as we only need to process minimal q-grams, rather than all q-grams. The size of the signature files grew, but less than proportional to the maximal sample size. Again this is caused by attributes with small value sets (e.g. less than 200) for which an increase in maximal sample size has no or little effect. SoS signatures are larger than q-gram signatures, as they store entire words rather than just q-grams. The difference in size is less than one might expect, and again is due to attributes with fewer values than the sample size allows[4]. For these the q-gram signatures can actually grow larger than the SoS-signature, due to the larger number of q-grams.

For sketch signatures, computation time grew almost linearly with sample size (number of random projections), as did signature size. Both computation time and sample size exceeded those of SoS and q-gram signatures, with the difference increasing for larger sample sizes.

Computation time for SoS-resemblance is given in Table 3. Larger signatures increased computation time in a roughly linear manner for all measures. Computation of q-gram resemblance is faster than SoS-resemblance, which in turn is faster than sketch-distance. For the smaller WildFinder data set, computation time is quite close, while sketch-distance

[4]Storage overheads also reduce the impact of string length.

	max=200	max=500	max=1000
SoS(M)	1.6s	4.3s	5.7s
q-gram(M)	1.1s	2.4s	4.3s
sketch(M)	6.8s	17s	33s
oracle(M)	8m	20m	50m
SoS(W)	67ms	165ms	195ms
q-gram(W)	29ms	37ms	48ms
sketch(W)	73ms	193ms	383ms
oracle(W)	5.5s	12s	20s

Table 3: Resemblance Computation

computation becomes relatively slower for the medical data set. The reason for this increase is that sketch-distance requires pairwise comparison of sketches, and thus computation time grows quadratically in the number of attributes. For SoS and q-gram resemblance, computation time grows quadratically in the worst and linearly in the best case.

Oracle-resemblance is much slower than all other measures, and thus best applied as a second refinement step, testing only attribute pairs with sufficiently high SoS-resemblance. For our medical database, over 98% of the 172,578 attribute pairs had an SoS-resemblance below 0.05, and thus could be omitted from testing for oracle-resemblance.

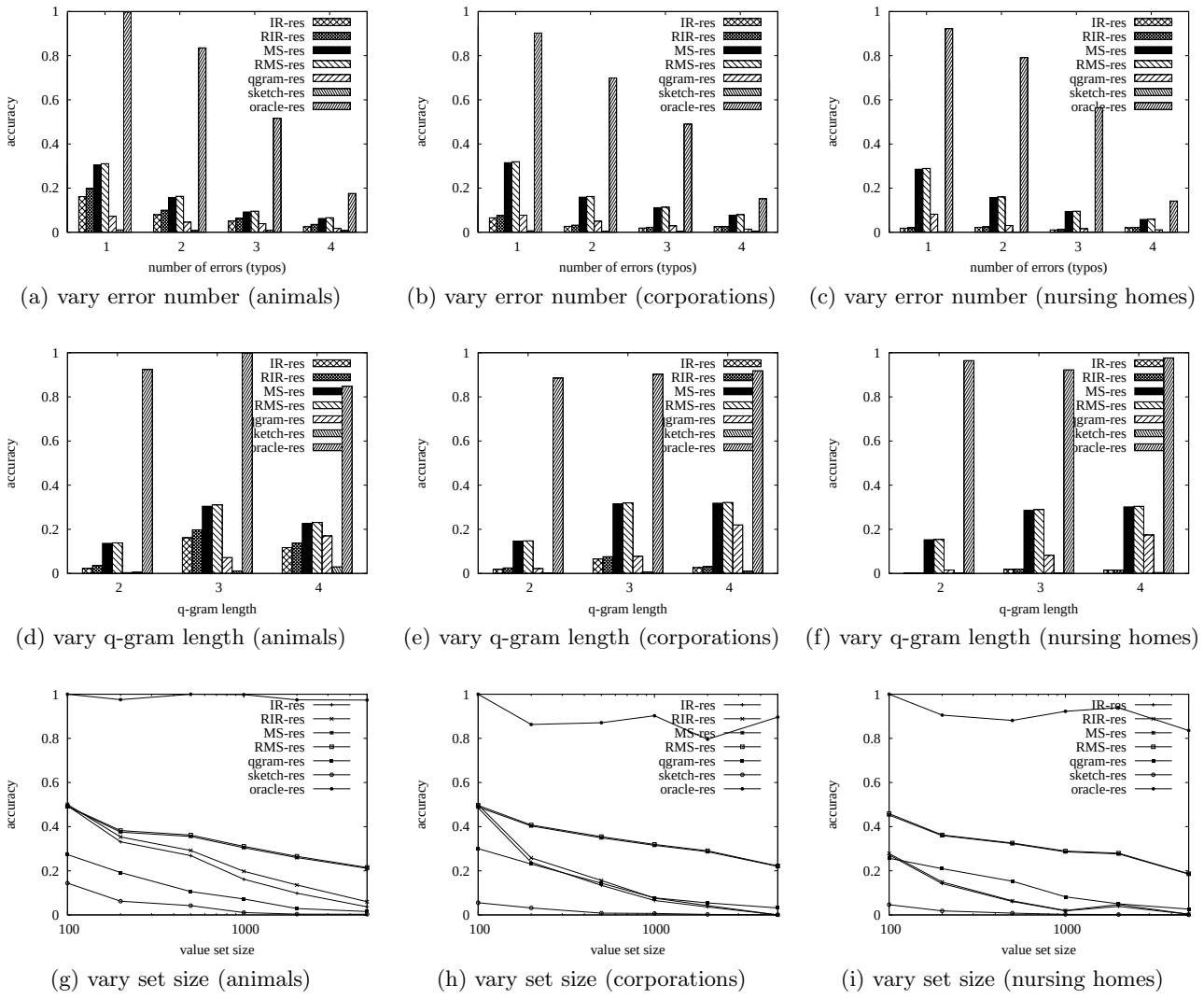
7. RELATED WORK

Data integration is an old topic, and we cannot possibly cover the vast body of literature available. At the same time, we already provided an in-depth discussion of the most closely related work in Section 2.

In this section we will only briefly discuss other, less closely related approaches to schema matching and database sampling. For more comprehensive surveys on these topics see e.g. [1, 2, 13] and [14, 15].

7.1 Schema Matching

Matching individual attributes for similarity (whether value overlap or equivalence of semantic domains) is only one important task of many during schema matching. More comprehensive systems such as Bellman [5], Clio [16] or iMap [17] also consider matches between attribute sets and simple transformations. Our work solves one specific task here, e.g. that of an 'overlap searcher' in iMap, with the capability of being able to find overlap between 'dirty' value sets. Due to our focus on dirty data, our methods can also match compound attributes (up to a degree), e.g. where 'location' matches 'City' plus 'State', and thus help reduce the search space. A difference (good or bad, depending on the appli-



(a) vary error number (animals) (b) vary error number (corporations) (c) vary error number (nursing homes)

(d) vary q-gram length (animals) (e) vary q-gram length (corporations) (f) vary q-gram length (nursing homes)

(g) vary set size (animals) (h) vary set size (corporations) (i) vary set size (nursing homes)

Figure 2: Accuracy for different parameter settings and data sets

cation) to other approaches, e.g. [18], is that SoS-similarity aims at finding overlap in value-sets, rather than domain equivalence. After matching attributes have been identified by our approach, it is possible to design queries using approximate joins [19, 20].

For matching individual values (for oracle functions) or attribute names (for schema-based matchers), string similarity measures are needed. Here a number of different metrics have been proposed, including edit distance, affine gap distance, Smith-Waterman distance and q-gram distance [13].

7.2 Database Sampling

Various approaches for sampling databases have been proposed. One important application is the estimation of join sizes for query optimizations [21, 22, 23]. However, despite the fact that we are looking for approximate join candidates, we are not so much interested in estimating the join size of these queries, but are looking for similarity of their value sets. Join size is not such a good indicator for us, since a single pair of similar frequent values (or absence of such a pair) can have a huge impact. Thus any mismatch of values can easily create a very wrong similarity result. Furthermore, samples must either be very large to obtain accurate results, or otherwise require careful examination of both sets to ensure that the 'right' values are selected, which is inefficient for more than 2 attributes.

Other sampling approaches aim to estimate aggregation results [21, 24]. As they focus on a single attribute only, they are not applicable to our problem. The issue of page-level sampling vs row-level sampling is addressed in [25, 26], but is not directly relevant to our work.

[21, 27] deal with sampling methods based on query operators. For us, the relevant operator is (approximate) set intersection. However, set intersection is not handled in [27], while [21] concludes that independent random sampling "is so inefficient that it is rarely worthwhile".

To our knowledge, no methods for sampling sets to find 'similar' objects shared between them have been proposed. Synchronized sampling methods for finding identical objects between sets have been investigated for document matching [8, 6, 9] and applied to database attributes in the Bellman system [5]. We already discussed these in Section 2.

8. CONCLUSION

We have introduced the notion of SoS-resemblance for measuring similarity between sets-of-sets. This new measure is amenable to efficient sampling techniques which we have also developed, and allows efficient comparison of large numbers of sets-of-sets. One important application motivating this work is to quickly identify string-valued attribute pairs in relational databases allowing sensible approximate joins, which is a vital step for integrating large databases. Our method is designed to cope with 'dirty' data, where values may be similar but not identical. While our experiments have shown that our basic SoS-resemblance measure is not sufficiently accurate (in some cases even less accurate than existing methods), we introduced variants (RIR-sum, minset-sum) which increase accuracy greatly with no impact on computation time. Also, these variants (minset-sum to be precise) allow detection of 'dirty' subset relationships.

To improve accuracy even further in a second analysis stage, we discussed how our set-of-sets signatures can be used by arbitrary string-similarity functions, leading to the oracle-resemblance measure. As oracle-resemblance is time consuming to compute, this is best done only on a reduced set, where pairs of unrelated attributes have been filtered out using SoS-resemblance. As both measures can operate on the same samples, they work together well.

For future research, we note that our SoS-resemblance measure and SoS-sampling method are not primarily designed to operate on sets of strings, but on sets-of-sets. Thus it seems likely that they could find application in other areas, in particular for other data formats such as XML. In this context it may also be interesting to investigate how our measure and sampling approach can be extended to deeper nestings of sets, such as sets-of-sets-of-sets.

9. REFERENCES

[1] A. Y. Halevy, A. Rajaraman, and J. J. Ordille, "Data integration: The teenage years," in *VLDB*, 2006, pp. 9–16.

[2] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001.

[3] M. J. Franklin, A. Y. Halevy, and D. Maier, "From databases to dataspaces: a new abstraction for information management," *SIGMOD Record*, vol. 34, no. 4, pp. 27–33, 2005.

[4] A. Y. Halevy, M. J. Franklin, and D. Maier, "Principles of dataspace systems," in *PODS*, 2006, pp. 1–9.

[5] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk, "Mining database structure; or, how to build a data quality browser," in *SIGMOD*, 2002, pp. 240–251.

[6] A. Broder, "On the resemblance and containment of documents," in *SEQUENCES: Proceedings of the Compression and Complexity of Sequences*. IEEE Computer Society, 1997, p. 21.

[7] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Computer Networks*, vol. 29, no. 8-13, pp. 1157–1166, 1997.

[8] U. Manber, "Finding similar files in a large file system," in *USENIX Winter*, 1994, pp. 1–10.

[9] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *CPM*, 2000, pp. 1–10.

[10] C. E. Shannon, *A Mathematical Theory of Communication*. CSLI Publications, 1948.

[11] C. Li, B. Wang, and X. Yang, "Vgram: Improving performance of approximate queries on string collections using variable-length grams," in *VLDB*, 2007, pp. 303–314.

[12] J. Bauckmann, U. Leser, F. Naumann, and V. Tietz, "Efficiently detecting inclusion dependencies," in *ICDE*, 2007, pp. 1448–1450.

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

[14] D. Barbar'a, W. Dumouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik, "The New Jersey data reduction report," *IEEE Data Engineering Bulletin*, vol. 20, pp. 3–45, 1997.

[15] F. Olken and D. Rotem, "Random sampling from databases - a survey," *Statistics and Computing*, vol. 5, pp. 25–42, 1994.

[16] R. J. Miller, L. M. Haas, and M. A. Hernández, "Schema mapping as query discovery," in *VLDB*, 2000, pp. 77–88.

[17] R. Dhamankar, Y. Lee, A. Doan, A. Y. Halevy, and P. Domingos, "iMAP: Discovering complex mappings between database schemas," in *SIGMOD*, 2004, pp. 383–394.

[18] B. T. Dai, N. Koudas, D. Srivastava, A. K. H. Tung, and S. Venkatasubramanian, "Validating multi-column schema matchings by type," in *ICDE*, 2008, pp. 120–129.

[19] W. W. Cohen, "Integration of heterogeneous databases without common domains using queries based on textual similarity," in *SIGMOD*, 1998, pp. 201–212.

[20] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate string joins in a database (almost) for free," in *VLDB*, 2001, pp. 491–500.

[21] F. Olken and D. Rotem, "Simple random sampling from relational databases," in *VLDB*, 1986, pp. 160–169.

[22] S. Ganguly, P. B. Gibbons, Y. Matias, and A. Silberschatz, "Bifocal sampling for skew-resistant join size estimation," in *SIGMOD*, 1996, pp. 271–281.

[23] S. Chaudhuri, R. Motwani, and V. R. Narasayya, "On random sampling over joins," in *SIGMOD*, 1999, pp. 263–274.

[24] S. Acharya, P. B. Gibbons, and V. Poosala, "Congressional samples for approximate answering of group-by queries," in *SIGMOD*, 2000, pp. 487–498.

[25] S. Chaudhuri, G. Das, and U. Srivastava, "Effective use of block-level sampling in statistics estimation," in *SIGMOD Conf.*, 2004, pp. 287–298.

[26] P. J. Haas and C. Koenig, "A bi-level Bernoulli scheme for database sampling," in *SIGMOD*, 2004, pp. 275–286.

[27] J. Gryz, J. Guo, L. Liu, and C. Zuzarte, "Query sampling in DB2 universal database," in *SIGMOD*, 2004, pp. 839–843.