SANDIA REPORT

SAND2018-14090
Unlimited Release
Printed November, 2018

Entity Resolution at Large Scale: Benchmarking and Algorithmics

Berry, Kincher-Winoto, Phillips, Augustine, Getoor

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology and Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2018-14090
Unlimited Release
Printed November, 2018

Entity Resolution at Large Scale: Benchmarking and Algorithmics

Jonathan W. Berry, Center for Computing Research
Sandia National Laboratories
Mail Stop 1327
P.O. Box 5800
Albuquerque, NM 87185

Cynthia A. Phillips, Center for Computing Research
California Sandia National Laboratories
Mail Stop 1326
P.O. Box 5800
Albuquerque, NM 87185

Kina Kincher-Winoto, Data Science and Cyber Analytics
California Sandia National Laboratories
Mail Stop 9105
PO Box YYY
Livermore, CA 94551-0YYY

Lise Getoor
Eriq Augustine
Department of Computer Science
University of California, Santa Cruz
1156 High Street
Santa Cruz, CA 95064

Abstract

We seek scalable benchmarks for entity resolution problems. Solutions to these problems range from trivial approaches such as string sorting to sophisticated methods such as statistical relational learning. The theoretical and practical complexity of these approaches varies widely, so one of the primary purposes of a benchmark will be to quantify the trade-off between solution quality and runtime. We are motivated by the ubiquitous nature of entity resolution as a fundamental problem faced by any organization that ingests large amounts of noisy text data.

A *benchmark* is typically a rigid specification that provides an objective measure usable for ranking implementations of an algorithm. For example the Top500 and HPCG500 benchmarks rank supercomputers based on their performance of dense and sparse linear algebra problems (respectively). These two benchmarks require participants to report FLOPS counts attainable on various machines.

Our purpose is slightly different. Whereas the supercomputing benchmarks mentioned above hold algorithms constant and aim to rank machines, we are primarily interested in ranking algorithms. As mentioned above, entity resolution problems can be approached in completely different ways. We believe that users of our benchmarks must decide what sort of procedure to run before comparing implementations and architectures. Eventually, we also wish to provide a mechanism for ranking machines while holding algorithmic approach constant.

Our primary contributions are parallel algorithms for computing solution quality measures *per entity*. We find in some real datasets that many entities are quite easy to resolve while others are difficult, with a heavy skew toward the former case. Therefore, measures such as global confusion matrices, F measures, etc. do not meet our benchmarking needs. We design methods for computing solution quality at the granularity of a single entity in order to know when proposed solutions do well in difficult situations (perhaps justifying extra computational), or struggling in easy situations.

We report on progress toward a viable benchmark for comparing entity resolution algorithms. Our work is incomplete, but we have designed and prototyped several algorithms to help evalute the solution quality of competing approaches to these problems. We envision a benchmark in which the objective measure is a ratio of solution quality to runtime.

Acknowledgment

This work was funded by the Sandia National Laboratories' Laboratory Directed Research and Development (LDRD) program in the Computing and Information Systems (CIS) investment area. We thank the University of California, Santa Cruz for a rewarding collaboration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A *benchmark* is typically a rigid specification of a computational procedure to be evaluated. For example, the Top500 [13, 15] is based upon dense linear algebra in the form of LINPACK computations [14]. The Top500 has been hugely influencial in driving the development of High-Performance Computing (HPC) platforms. Ironically, however, these Top500 computations are generally not representative of true scientific computing workloads that run on such machines. In response, the HPCG500 [12] benchmark was designed to perform more representative sparse computations which stress different portions of the machine. The machine rankings generated by these two benchmarks differ significantly.

Entity Resolution (ER) problems comprise a canonical data analysis challenge that involves both textual and relational data. This problem is fundamental in several national security contexts. Furthermore, corporations of all kinds (not just social networking companies) now recognize the benefit of applying business analytics to their data. ER is a fundamental preprocessing step for these analytics.

These incoming data will be tagged with various identifiers such as Twitter handles, email addresses, IP addresses, usernames, etc. Slight errors or irregularities in these tags have the potential to disrupt any downstream data analytics.

Of the many application contexts that give rise to ER problems, we focus our thinking on two non-sensitive applications: coauthorship networks and actor/movie networks. The benchmarks that could be based on the concepts we explore would be directly applicable to national security data as well. In other contexts, the Linqs group at UC Santa Cruz [3] has also studied familial networks [18] and other types of relational data.

1.1 Entity Resolution: basic concepts

For a comprehensive introduction to entity resolution, see the LINQS tutorial *Entity Resolution for Big Data Problems* [2].

ER problems are described using a small set of basic terms. We give intuitive descriptions of these terms here, and formalize them below in Section 2.2. A *reference* occurs when an entity is mentioned or acknowledged in some context. For example, an author's name appears

at the top of a paper, a tweet can mention a person by his/her Twitter id, and an actor is acknowledged in the list of credits that roll at the end of a movie. An *entity* is the person or object that is acknowledged or mentioned by a reference.

The reality of data collection is that references may not uniquely and unambiguously identify the entities they are associated with. Actors use stage names, Twitter handles may be arbitrarily named, textual names or identifiers can be misspelled, mistaken, corrupted, altered, ambiguous or problematic in other ways.

Understanding the textual and/or relational structure in data generally means knowing which references are associated with which entities. In its essence, obtaining this knowledge is the ER problem.

1.2 Types of ER problem

The relationships in knowledge graphs (graphs with attributes) are *events*. For example, events in coauthorship networks are co-authored papers. These group authors into one set per paper (a hyperedge). Events in actor/movie networks are movies, which group casts together into hyperedges.

Pujara and Getoor [24] describe three different types of entity resolution, and [2] gives examples of each:

1. deduplication: disambiguition of entities within one knowledge graph
2. record linkage: disambiguition between multiple knowledge graphs
3. reference matching: match noisy records to clean records in a table

Our benchmarks concern problem deduplication. The ideas we discuss could be applied to benchmarks for the other types.

1.3 Types of ER algorithm

We note two fundamental distinctions among ER algorithms:

- *collective* vs. not [10]
- *statistical* vs. not [7]

Collective ER algorithms appeal to graph topology; non-collective ER algorithms ignore that topology. Whether or not to introduce the complexity of collective ER in an enterprise solution to ER problems is a good motivation for benchmarking. An organization considering a big data solution to their ER problems would generate datasets specific to their operations (as opposed to relying on literature) and apply benchmarking to help inform the decision.

A variety of simple graph and/or string algorithms could be applied to ER, but methods such as [7] apply statistical inference to the problem with promising initial results. Once more, organizations facing enterprise-level decisions would need benchmarking to decide whether it is possible to deploy methods based on statistical inference.

1.4 Our assumptions

For this report, we assume that references or mentions are character strings, and that events are co-occurrences of mentions (which we model as sets of strings in our benchmarks). Each reference refers to exactly one entity. In Section 2.2 we will introduce notation formalizing these assumptions and other foundations for our work.

1.5 Our challenge: benchmarking ER algorithms

Algorithms mentioned in Section 1.3 vary widely in nature and computational requirements. Perhaps the simplest ER algorithm is to ignore the events and simply sort all references lexicographically, determining proposed entities by cutting the sorted list into groups according to an edit distance threshold. Variants of this are known as *sorted neighborhood* methods [16], and some run asymptotically faster than pure comparison-based sorting. Even naive versions that simply sort the references would be readily run on terabyte-sized or larger datasets. However, the solution quality will most likely be low since most of the input structure is ignored.

In contrast, collective ER algorithms that consider relational information can be approached via sophisticated *statistical relational learning* techniques. For example, an ER formulation in Probabilistic Soft Logic (PSL) [4], a software capability for performing statistical inference, will exploit the event structure to make much better proposed groupings of references into entities.

> Our primary challenge is to provide a benchmarking capability so that the trade-off between ER solution quality and computational complexity can be studied as the complexity of the problem instance varies..

Chapter 2

Background

2.1 The benchmarking process

Our approach to assessing the quality of ER algorithm solutions is inspired by Bhattacharya and Getoor [11], who propose, given an ER instance with ground truth, to compute recall scores *per-entity* and imprecision scores *per-entity-pair*. This is an important advance over the typical aproach for ER algorithm assessment, which is to compute a single, global confusion matrix [8, 10]. The jump from a global assessment to a per-entity assessment is crucial for benchmarking. Real datasets such as the Internet Movie DataBase (IMDB) [1] have the property that most of the entity-reference associations are trivial. For example, most actors use only one name and that name is accurately reported in the data. However, a small but nontrivial percentage of the entity-reference associations are quite difficult to recognize. The ideas of Bhattacharya and Getoor [11] point us to the important and difficult parts of the problem. With this granularity, distinctions between algorithms that might have similar global confusion matrices can be teased out.

In order to compute their per-entity recall and imprecision measures, Bhattacharya and Getoor [11] define four concepts that we describe and modify below: *identifying attributes*, *identifying relationships*, *ambiguous attributes*, and *ambiguous relationships*. They also show how to compute the probabilities of these phenomena, given a small ER instance with ground truth.

This note primarily details our work in redefining and extending these concepts, while retaining their original spirit, so that we can evalute ER instances with millions of entities and events. This improved scalability is one of our primary contributions, and is a prerequisite to large-scale ER benchmarking.

Given this framework, one natural way to globally assess the performance of a candidate algorithm is to define and compare cumulative distribution functions (CDF) of the per-vertex recall and imprecision measures for the ground truth and the algorithm's proposed solution. If the recall and imprecision CDF's of the ground truth and algorithm solution match well, that is strong evidence that the algorithm succeeds on easy portions of the problem and struggles only where such a struggle is expected. We formalize this notion and give an example for recall in Section 4.1. At the time of this writing, scalable computation of imprecision remains an open problem.

2.2 Notation

Following [11], we say that the simplest form of the entity resolution problem consists of a set of references $R = \{r_i\}$, with attributes $\{R.A_1, R.A_2, \ldots, R.A_k\}$. Let $E = \{e_j\}$ be a ground truth set of entities, unobserved by any ER algorithm. We use $E(r_i) \in E$ to denote the mapping from references to entities. Let $H = \{h_i\}$ be a set of hyperedges representing events or relationships between references. In this notation, each reference r_j encodes exactly one hyperedge h_k. That is, r_j encapusulates the attributes of reference j and implicitly identifies the event in which that reference occurs.

We retain some of this notation, but find it convenient to simplify in one way and add complexity in another. For this note we consider a reference to have a character-string name and no other attributes. Therefore we dispense with the "A_k" notation and explicitly represent the character string attribute. Our benchmarks will be complicated enough without multiple attributes, and we believe that the challenge of entity resolution with a single string attribute and relational information is sufficient to evaluate most approaches. Our benchmarking ideas could be extended to handle multiple attributes in the future.

For the remainder of this note, let $H = \{h_0, \ldots, h_{m-1}\}$ be a set of m *events*, and $S = \{s_0, \ldots, s_t\}$ be a set of strings interpreted to be unique *reference names*. Given an event h_i, we have both $h_i \in H$ and $h_i \subseteq S$. Thus, an event is a selection of reference names associated with some object or activity (such as the set of author names in a published paper or the set of actor/actress names that appear in the credits of a movie). Note that we could have defined h_i to be a multiset, which would correspond, for example, to a movie with two different roles played by actors using the same name. However, for simplicity of presentation we assume that the set of reference names associated with an event is a simple set (as it is in the IMDB database, even when a single actor plays multiple roles). Still following [11], we use the notation E to represent a set of entities. The set of events H is sometimes referred to as the *cooccurrence structure*.

We redefine the reference set R as follows: $R \subseteq H \times S$. That is, if $(h_j, s_k) \in R$, then some entity used the reference name s_j in event h_k. Note that many different events could contain the string s_k, and that these reference names could refer to different entities. We call out the set of unique strings S in our notation because some of our algorithms described later operate explicitly on this set.

Let $E_i \subseteq 2^R$ be the set of references used by a single entity i. Let $e_i \in |E_i|$ be the number of references in that set. This departs slightly from the notation of [11], which sometimes uses e to denote a single entity. Note that there are $\binom{e_i}{2}$ pairs of references associated with entity i. We call each of these pairs an *item* and refer to the set of items as T_i. For example, if $E_i = \{r_1, r_2, r_3, r_4\}$,then $T_i = \{(r_1, r_2), (r_1, r3), (r_1, r_4), (r_2, r_3), (r_2, r_4, (r_3, r_4)\}$.

We also use the notation $H(E_i)$ to denote the set of events h_j associated with entity E_i. Formally, $H(E_i) = \{h_j : (h_j, s_k) \in E_i\}$.

Let P be a partitioning of R into $p = |P|$ partitions, denoted as follows: $P = \{P_0, P_1, \ldots, P_n\}$,

where $R = \cup_i P_i$ and $P_i \cap P_j = \emptyset$ for $i \neq j$. P is therefore a hard grouping of references into entities. Note that since there is a ground truth solution in ER problems, we do not consider mixture models. In our benchmarking work, both ground truth and algorithm solutions take this form. We use the notation P_g to represent the ground truth of an entity resolution problem.

Formalizing the intuitive descripion of the benchmarking process given in Section 2.1,

1. Generate or obtain an ER instance with ground truth, which consists of:
   - *Ground truth*: the partitioning P_g containing correct reference groupings of references into entities.
   - *Algorithm input*: the set R of references grouped into events.
   - *Expected algorithm output*: a partitioning P_A capturing the reference-entity associations found by Algorithm A.
2. Assess the inherent difficulty of this instance, per-entity, by computing per-entity recall and imprecision measures on the ground truth. *See Sections 4.1 and 4.3.*
3. Run a candidate algorithm on the algorithm input to obtain P_A.
4. Assess the algorithm's performance compared to expectations (See Section 5.1).

We will measure differences between strings using an unweighted *edit distance* metric. Let $d(s_1, s_2)$ be the edit distance between strings s_1 and s_2, and we assume that this distance is an integer count of two kinds of difference: a *gap* and a *mismatch*. We weight each of these equally, so the edit distance is the sum of the numbers of gaps and mismatches. For example, $d(s_1 = abcde, s_2 = acxe) = 2$ because the b in the second position of s_1 is a gap in s_2, and (d, x) is a mismatch in position 4 of s_1. The edit distance $d(s_1, s_2)$ quantifies the steps that transform s_1 into s_2, and it is convenient for us to think of this transformation as a sequence of *alignment* steps (removing gaps) followed by a sequence of *mismatch* steps (removing mismatches). After removal of gaps, but before any mismatch steps, we say that strings are *aligned*. For example, in the $d(s_1, s_2)$, example above the single alignment step yields aligned strings *acde* and *acxe*, and the single mismatch step convert the former into the latter by changing d into x. We omit the description of our subquadratic algorithm to exactly find all pairs of strings with edit distance $\leq d$, but it exists in code documentation form.

Chapter 3

Efficiently finding all close pairs in a set of strings

Our benchmark entities have string attributes, and a kernel operation of the benchmarking process we propose is to find all "close" pairs in a corpus of n strings. To produce and evaluate benchmark instances with millions of entities, the brute force approach of comparing all $O(n^2)$ pairs for closeness is not practical. There are efficient data analysis methods (primarily based on clustering) to find approximate solutions. However, we need exact solutions in order to produce effective benchmarks.

A typical clustering approach would be to embed the strings in some space (metric or non-metric) and using clustering to find the pairs with small edit distance. For example, the Python library NMSlib [20] has applicable methods. The computational biology literature abounds with string alignment algorithms, most notably the classical BLAST algorithm [5]. However, this type of algorithm finds alignments of strings and does not solve our all-pairs closeness problem.

We propose a new algorithm. Given a corpus of n strings and a threshold d, We find all pairs of strings with edit distance at most d. It runs in time $O(n \log n + x)$, where x is the number of pairs that have edit distances strings

The description below is informal. In future work we plan to formalize the correctness argument and publish this algorithm with an efficient implementation. At the time of this writing we have an inefficient Python implementation and the following description obtained from its header comments.

```
Usage: AM <strings.dat> <limit> <d> <outfile>
         compute on only the first <limit> strings in <strings.dat)

Example:
test.dat:
abcde
abde
abxe
ayze
abd
```

```
python AM test.dat 5 2 test.out

test.out:
abcde abde
abcde abxe
abcde abd
abde abxe
abde ayze
abde abd
abxe ayze
abxe abd
```

Description: A brute force approach would compute the edit distance between each of the $O(n^2)$ pairs of strings. Instead we introduce the concept of "AM (Alignment/mismatch) signatures" and use them to generate substrings that are comparable via standard sorting. Merge steps then yield the close strings.

Given a longer string s_1 and a shorter string s_2 (where the lengths could be the same), we perform a number of 'alignment' steps on each string. This generates sets of substrings of uniform length that have characters removed. Next, we perform a series of 'mismatch' steps, which generate further sets of substrings (one set for each element in the sets of alignment substrings). The mismatch substrings encode potential mismatch positions. The final sets of substrings are comparable, and any exact matches indicate that the associated original string pairs have edit distance $\leq d$.

The AM steps for a given string pair are encoded into a signature, which is a triple. Given s_1 and s_2, the signature (a_1, a_2, m) indicates that we will generate all substrings of s_1 that have a_1 characters missing (call that set A1) and all substrings of s_2 that have a_2 characters missing (A2). Note that each substring $s \in A1$ has length $|s_1| - a_1$, and each substring $s \in A2$ has length $|s_2| - a_2$. Below, we will describe an algorithm to generate signatures. Any signature produced by our algorithm will ensure that

$$|s_1| - a_1 = |s_2| - a_2$$

.

Now, for each substring $s \in A1$, we generate all substrings missing m characters. However, we append to each substring an encoding of the character positions that were dropped. Call the resulting sets A1m and A2m. For example, suppose that string $s_1 = abcde \in S$, $d = 3$, $a_1 = 1$, and that $m = 2$. The alignment step dropping e will yield the substring $abcd \in A1$. We insert into $A1m$ the $\binom{(|s_1|-a_1}{m)}$ substrings:

$$\{ab23, ac13, ad12, bc03, bd02, cd01\}$$

.

Suppose that the original string s_1 was abcde, and was the p'th input string. We associate the index of the original string to each element of A1m, yielding:

$$\{(ab23, p), (ac13, p), (ad12, p), (bc03, p), (bd02, p), (cd01, p)\}$$

.

Once A1m and A2m have been constructed, they are each sorted individually. A final merge step finds, with respect to signature (a_1, a_2, m), all pairs of original strings in which the edit distance can be characterized by a_1 missing characters in s_1, a_2 missing characters in s_2, and at most m mismatches in the aligned substrings.

If we explore the full set of possible signatures, we encounter all pairs of strings that have edit distance at most d. The algorithm for computing this set of signatures follows.

Let p be the difference between the lengths of two strings s_1 and s_2 (wlog, s_1 is longer), and let d be the maximum allowable edit distance. Note that $p \leq d$.

Suppose that to align s_1 and s_2, i.e., to remove characters in order to produce substrings of equivalent length) we remove a_l characters from s_1 and a_s characters from s_2. Note that $|a_l - a_s| = p$.

The set of signatures T_p is all triples (a_l, a_s, m) such that a_l, a_s, and m are integers and

$$a_l + a_s + m = d$$

As an initialization step, we compute T_p for all $0 \leq p \leq d$. Next, we put all input strings into buckets by their string length and compute, for each string length l, where min_length(S) <= l <= max_length(S), a set of 'responsibilities.' These are the numbers of dropped characters required by the full set of signatures to obtain alignment substrings, subject to boundary constraints (i.e., l is long enough). Specifically, the bucket of length-l strings is considered responsible for generating all alignment substrings required by the signatures. Note that this computation is embarrassingly parallel (though our code at the time of this writing implements a serial version of the algorithm).

Once We have computed all alignment substrings for each bucket, we loop through each length l and compute the mismatch substrings from the alignment substrings, once more appealing to the signatures to know the numbers of mismatches permitted.

For each pair (l,l), (l,l+1), ..., (l,l+d), we construct a sorting problem. The sets of mismatch substrings (along with the index associating them back to original strings) are sorted, then the two sorted lists are merged. Matching substrings indicate pairs of original strings that have edit distance $\leq d$.

TODO: plot showing asymptotic performance vs. brute force

Chapter 4

Computing the inherent difficulty of entity resolution instances

Our benchmarking process features the quantification of solution quality. Only with a careful accounting of this quality can we understand tradeoffs between cost (algorithm runtime complexity) and benefit (usefulness of solution). Our assessment of solution quality breaks down into the familiar notions of recall and imprecision, but in a non-traditional way: we assess this quality *per-entity*, even when the problem size is in the millions of entities and events.

4.1 Recall

In [11], Bhattacharya and Getoor propose a notion of *recall* given entity $E_i \in P$. Their recall is parameterized by *level*, which enables neighbors and neighbors-of-neighbors of E_i in its cooccurrence structure H to influence E_i. The recall of E_i can be interpreted as the degree to which problem structure (in terms of both attribute similarity and relational structure) helps identify and group together its set of references. An entity with high recall should be easy to resolve, while one with low recall is inherently difficult to resolve.

Bhattacharya and Getoor's recall is a function with three components: an attribute component, a relationship component, and a neighbor component. These components are computed by counting "identifying" attributes and relationships, which are constructs that quantify the help an ER algorithm will get, respectively, from the attributes and relationships of an entity. These definitions are parameterized by a closeness ϵ. In this note, we measure closeness in terms of the simple edit distance metric defined in Section 2.2 (smaller ϵ means greater similarity).

Of the three components of recall, two are fundamental:

- **attribute identification probability** $a_I(E_i, \epsilon)$: the probability that the reference names of a *pair of references chosen randomly from* E_i (the set of references of entity i) are ϵ-similar to each other.
- **identifying relationship probability** $r_I(E_i, \epsilon)$: the probability a *pair of events*

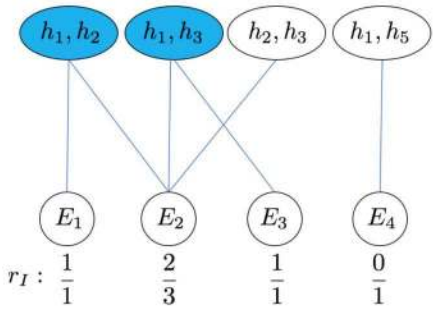


Figure 4.1. Bhattacharya/Getoor's concept of identifying relationship probability. A pair of events is blue if two or more entities participate in both events. The identifying relationship probability of an entity is the fraction its neighboring event pairs colored blue.

(h_j, h_k) *chosen randomly from* $H(E_i)$ is an *identifying relationship*, i.e., that there exists an entity $E_x \neq E_i$ such that $h_k \in H(E_x)$ and $h_k \in H(E_x)$.

Figure 4.1 illustrates the concept of identifying relationship probability. For example, $H(E_2) = \{h_1, h_2, h_3\}$. Each pair of these events is represented as a vertex, and all such vertices are laid out in the top row of the figure. Vertices on the bottom row represent entities, and an edge between rows indicates participation by the entity in both events of the pair. We note that any top-row vertex with degree greater than one indicates an identifying relationship. In the figure, such vertices are colored blue. The identifying relationship probability of an entity (a bottom-row vertex) is the proportion of its neighborhood colored blue.

4.2 Efficiently computing recall

We must perform three computations per-entity to compute recall:

4.2.1 Computing identifying attribute probabilities

In small ER instances, it is acceptable to compute these probabilities by brute force. Given an entity E, simply test all pairs of attributes for closeness, and report the fraction of pairs that are close. However, we must handle large datasets in our benchmarking work. In As part of our LDRD project, we introduced an algorithm that will find all close pairs of a set of n strings in time $O(n \log n + x)$, where x is the global number of close pairs. We use this algorithm to compute identifying attribute probabilities.

The all-pairs close strings problem can also be approached using inexact clustering methods. For example, the open-source Python code NMSlib [**?**] could be used. However, we compute the solution exactly.

4.2.2 Computing identifying relationship probabilities

Algorithm 1 Identifying relationship probabilities

```
1:  procedure RIPROBS(E,H, Δ, r_I)
2:      ▷ E : the set of entities
3:      ▷ H : the set of events
4:      ▷ Δ : the largest event to consider (e.g. O(1000) references
5:      ▷ rI : result: identifying relationship probabilities.
6:
7:      T = { a threadsafe key-value store }
8:      ▷ In Parallel:
9:      for (h_j, h_k) ∈ H(E_i) × H(E_i)   ∀_i : |H(E_i)| ≤ Δ do
10:         if (h_j, h_k) ∉ T then                  ▷ degree((h_j, h_k) is changing from 0 to 1
11:             T(h_j, h_k) = i                                  ▷ we remember Entity i
12:         else
13:             if x = T(h_j, h_k) ∈ {0, n − 1} then   ▷ degree((h_j, h_k) is changing from 1 to 2
14:                 rI[x] = rI[x] + 1                    ▷ give remembered Entity x credit
15:                 rI[i] = rI[i] + 1
16:                 T(h_j, h_k) = n                  ▷ indicate that we no longer remember
17:             else
18:                 rI[i] = rI[i] + 1
19:     ▷ In Parallel:
20:     for E_i ∈ E do                              ▷ convert from counts to probabilities
21:         rI[i] = rI[i]/|H(E_i)|
```

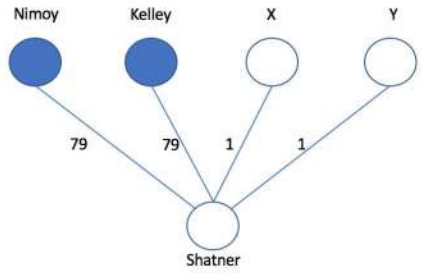
We could explicitly create the graph illustrated in Figure 4.1 and run a simple graph algorithm to compute identifying relationship probabilities for all entities. However, this would require two passes through the entire dataset: one to build the graph and one to read off the identifying relationship probabilities. Instead, we introduce Algorithm 1, a parallel algorithm that makes only one pass through the data.

We maintain a threadsafe key-value store mapping event pairs (h_j, h_k) to counts: the number of entities in that event pair. We parallelize over all pairs of events within any single entity. We accept an implausibility parameter Δ and exlude any entity in more than Δ events. For example, if a dataset owner reasons that no entity is likely to be in more than 1000 events, Δ is set accordingly.

With these parameters, our runs in time $O(n\Delta^2)$, and in practice is efficient enough to process the entire IMDB dataset (containing more than three million entities) on a modern

workstation with a terabyte of RAM. Our implementation uses the C++ MultiThreaded Graph Library (MTGL) [9] and its Manhattan loop collapse feature to handle the main loop.

4.2.3 Computing neighborhood information



$N(\text{Shatner}) = \{Nimoy, Kelley\}$

Figure 4.2. Blue vertices represent the set $N(E_i)$ when E_i is a notional version of the actor William Shatner with 81 appearances (79 episodes of the original *Star Trek* and exactly two others).

Bhattacharya and Getoor [11] define the concept of an *entity neighbor* of Entity E_i to be one of a small, select set of other entities with which E_i co-occurs frequently. They denote E_i's set of entity neighbors as $N(E_i)$. Note that this is not necessarily the entire set of E_i's neighbors in the co-occurrence structure.

For example, consider Figure 4.2. We depict a notional version of the actor William Shatner in a small actor/movie network. Suppose that all 79 episodes of the original *Star Trek* series are in the network, but it is very sparse otherwise. The only two non-Star Trek appearances Shatner makes in this network are two-person dramas, one with X and one with Y. We might define $N(E_i)$ to be Shatner's two Star Trek co-stars.

Bhattacharya and Getoor [11] also use the notation $p^i(j)$ to denote the probability that a randomly-selected co-occurence from $H(E_i)$ contains a given $E_j \in N(E_i)$. Supposing that Nimoy is the jth element of $N(E_i)$ in Figure 4.2, then $p^i_j = \frac{79}{81}$.

For our proposed benchmarks, we compute *level-1 recall*, which [11] defines to be:

$$R(E_i) = a_I(E_i) + (1 - a_I(E_i)) \times r_I(E_i) \times \sum_{j=1}^{|N(E_i)|} p^i_j a_I(j)$$

We interpret this expression as follows. If the identifying attribute probability $a_I(E_i)$ is

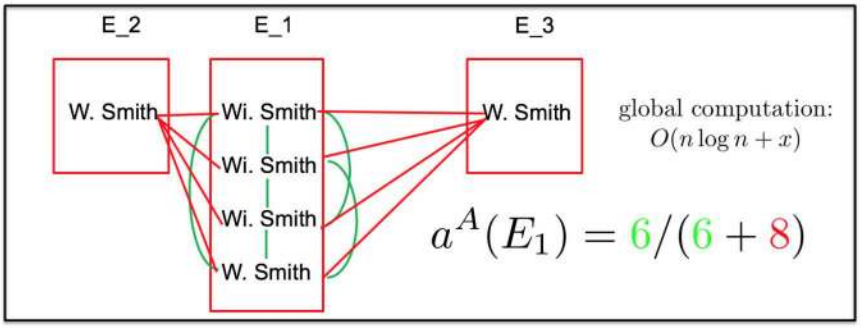


Figure 4.3. Attribute ambiguity

near 1.0, then the recall for E_i is at least that high. In addition, if the identifying relationship probability $r_I(E_i)$ is positve, recall is increased still further.

The *level-k* recall $R^k(E_i)$ is similar, except that the last term involves products of $p_j^i R^{k-1}(E_j)$. We do not deal with that complication here; neighbors inform $R(E_i)$ through their identifying attribute probabilities. Thus, in our computations, neighbor influence is limited to the identifying attribute probabilities of the entity neighbors of E_i.

4.3 Imprecision

In [11], Bhattacharya and Getoor propose a notion of *imprecision* to assess the false positive performance of ER algorithms. Their imprecision is to be computed for all pairs of entities (or a sampling of all pairs). We wish our benchmarking process to be computable without sampling, so we have revised these definitions.

As in [11], the spirit of our definitions is to quantify the likelihood that the data will cause ER algortithms to go wrong by grouping together references to different entities.

4.3.1 Computing attribute ambiguity

We first define a per-entity notion of attribute ambiguity:

- **ambiguous attribute probability** $a_A(E_i, \epsilon)$: the probability that a *pair of references* $\{r_j, r_k\}$ *chosen randomly from all* ϵ*-similar pairs involving at least one reference to* E_i has $E(r_j) = E_i$ and $E(r_k) = E_i$.

For example, consider Figure 4.3. Entity E_2 goes by the name "Wi. Smith" three times and by the name "W. Smith" once. There are 14 pairs of ϵ-close references involving at least one reference to E_2. Among those, only six pairs refer entirely to that entity. This is a case

of significant ambiguity, as the "W. Smith" references to E_1 and E_3 will complicate the job of any ER algorithm.

Given a small ϵ such as edit distance $d = 2$, we can compute $a_A(E)$ for all E in time $O(n \log n + x)$ using our string n is the number of references in the dataset and x is the number of ϵ-close pairs of strings in the dataset.

4.3.2 Computing relational ambiguity

In [11], Bhattacharya and Getoor propose a notion of *relational ambiguity* that quantifies the effect of ambiguous relationships causing relational ER algorithms to go wrong by grouping together references to different entities. Their notion is intuitive, but requires $O(n^4)$ computation. The idea is to consider, for each pair of entities, all pairs of relationships that might have this effect.

We seek a version of this relational ambiguity that is efficient to compute. At the time of this writing, we have not found one. Therefore, in order to obtain a practical measure of imprecision, we current limit ourselves to attribute ambiguity.

Chapter 5

Evaluating entity resolution algorithms

5.1 Actual Recall and Precision

Sections 4.1, 4.2, and 4.3 concerned evaluating a given dataset and ground truth to assess the inherent difficulty of obtaining good ER solutions. Even if most of the entities have strong identifying attribute and relation probabilities and not much attribute ambiguity, a small subset might be inherently difficult to resolve. The techniques of Sections 4.1 and 4.3 are designed to help us find these difficult entities.

Suppose that we are now given a grouping P of references into entities, as found by some ER algorithm. We wish to assess the quality of this grouping. Thanks to the computations described in Section 4.1, we have an assessment, for every entity E, of the inherent difficulty (at least in terms of recall) of grouping together the references of E.

We need a per-entity notion of "actual" recall and imprecision of partitioning P. We define this below, along with a suggested method of comparing the inherent and actual recall and imprecision. In the traditional lingo of classification algorithms, *relevant items* are those pairs of references associated with a given entity in the ground truth solution, while *selected items* are pairs of references grouped together by an ER algorithm. The traditional definition of *recall* is the proportion of relevant items that are selected, while that of *precision* is the number of selected items that are relevant.

For our per-entity actual precision, we adapt the definition of *selected items* for our purposes. A pair of references (r_j, r_k) is *selected for entity* E_i iff either r_j or r_k is a reference to E_i. Figure 5.1 illustrates these concepts.

- **actual recall** $\hat{R}_A(E)$: the proportion of pairs of references to entity E that are grouped together by the partitioning A (an algorithm solution).
- **actual precision** $\hat{P}_A(E)$: the proportion of items selected for E (i.e. (r_j, r_k) such that at least one of the pair refers to entity E) that are *relevant* (i.e., both r_j and r_k refer to E).

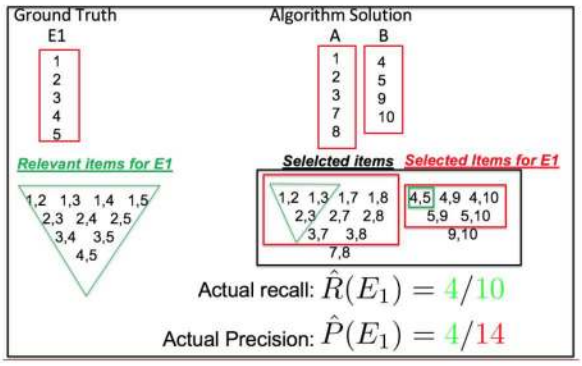


Figure 5.1. Examples illustrating the definitions of actual recall and actual precision

5.2 Benchmark evalution: comparing inherent and actual recall and precision

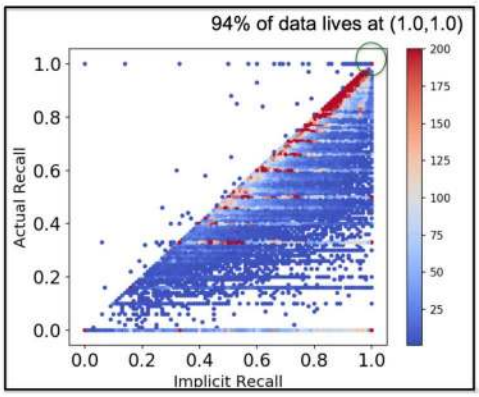


Figure 5.2. Evaluating the performance of an ER algorithm. Note that 94% of the problem is trivial, and our method illustrates algorithm performance on the non-trivial portion of the problem.

We propose to evaluate the effectiveness of ER algorithms via plots and statistics comparing inherent to actual recall and precision. For example, we construct an ER dataset from the IMDB database as follows:

- Entities are actors/actresses
- Relationships are movie casts of references (strings)
- Ground truth is available: the set of references for one person
- Algorithm input: the set of movie casts

- Algorithm output: proposed entities (groups of references)

There are roughly three million actors/actresses (entities) and a similar number of movies or shows (events). Let us evaluate the performance of a simple ER algorithm that ignores events and simply sorts all references lexicographically, then proposes entities by breaking the sorted list into partitions. A partition break occurs between the ith and $(i+1)st$ strings when their edit distance exceeds some small constant d ($d = 2$ in our example).

Figure 5.2 shows the *recall* results of this simple algorithm on the ER problem we derived from the IMDB dataset. The first thing to note is that 94% of all entities have inherent recall and precision of 1.0. Therefore, all of these entities should be easily identifiable by an ER algorithm. Figure 5.2 confirms that even this trivial algorithm succeeds in grouping the references of these entities with no false positives. A global measure (e.g. global confusion matrix, F-measure, etc.) without per-entity information would indicate at least 94% success, but this would be quite misleading.

More than 100,000 of the three million entities have some inherent difficulty. This is the hard part of the problem, and here is where we need to evaluate ER algorithm quality. Figure 5.2 shows at per-entity granularity how well the algorithm does in resolving each set of references. Intuitively, a good algorithm should produce actual recall as good or better than inherent recall. This would show up as a heat map with most of its mass at or above the diagonal $y = x$. We can see that this performance is not achieved in this case because a significant amount of mass is located below this line.

The data from this plot could be compiled into a single number such as a normalized per-entity loss in recall. However, we defer such a definition for now in favor of the plot. A superior algorithm would yield a plot with data much more aligned to $y = x$. Users of the benchmark would then evaluate the tradeoff between any extra running time and the resulting benefit in entity resolution for the hardest part of the problem.

5.3 Status of evaluating actual vs. inherent precision

Unfortunately, at the time of this writing our ideas regarding actual vs. inherent precision do not yield the intuitive plots that we see when comparing recalls. This will need to be addressed in order to enable benchmark users to assess the effects of false positives in the hardest parts of input problems.

Chapter 6

Probabilistic Soft Logic (PSL)

Probabilistic Soft Logic (PSL) is a statistical-relational learning (SRL) framework for defining hinge-loss Markov random fields (HL-MRFs), a class of undirected probabilistic graphical models that supports rich modeling of relational data. A PSL model consists of weighted logical clauses that encode statistical dependencies and structural constraints. PSL supports efficient maximum a posteriori (MAP) inference. The ability to perform fast, collective inference makes it a strong tool for solving ER problems.

6.1 Extracting a sample from IMDB for PSL

The entire IMDB dataset contains 23,794,202 references, 2,562,686 movies or events, and 3,255,621 actor/actresses or entities. Because of the development and time constraints of working with such a large dataset, we extract a smaller problem of under 1,000 references.

The relational nature of the datasets makes sampling a subproblem increasingly difficult. A naive sampling approach such as taking the first 1,000 references in the IMDB database may result in getting references that are disjoint. For example, if we sample an American actor from the 1940's and a Korean actor from 2010 then they will likely have little to no relationships between them. Thereby destroying the relational nature of the dataset.

Therefore, we begin our sampling by searching for an entity with only a couple aliases and created the problem around that entity. Somewhat arbitrarily, we chose actor "James A. Baffico" by the small number of aliases he has and movie appearances. He has two aliases in the IMDB database of "Jim Baffico" and "James Baffico". In the first alias, he goes by a well-known nickname for "James" and in the second alias, his middle initial is left out. Next, we extract the movies that "Baffico" is in, which is 15 titles. From those 15 titles, we pull the entire cast of all 15. After pulling in this information surrounding "Baffico", the problem contains 621 references, 15 movies or events, and 540 actors/actresses or entities.

6.2 Formulating entity resolution problems with PSL

The earliest entity resolution research focused on developing specialized similarity measures for local features such as strings and attributes [27]. More recent work in entity resolution has focused on using relationships between references to generate relational features. These relational features introduce dependencies between co-reference decisions for different references, resulting in a collective model that can outperform conventional approaches [25][17][26]. PSL makes it easy to incorporate local and relational information into a single model.

6.2.1 Local Information

Local features are those that can be computed for a pair of entities (or references) independently of the co-reference decisions of other entities. Examples of local features include string similarity of names, image similarity of photographs, and demographic similarity. One key characteristic for a local feature is that its value does not depend on the entity resolution decisions for other pairs of entities.

Here is a PSL rule that uses the local information of name similarity to compare two references. If the names belonging to two different references are the similar, then there is evidence that the references refer to the same entity.

$$\textsc{HasName}(Name1, Ref1) \ \& \ \textsc{HasName}(Name2, Ref2) \ \& \ \textsc{Similar}(Name1, Name2) \rightarrow \textsc{Same}(Ref1, Ref2)$$

6.2.2 Collective Entity Resolution

The power of PSL comes from its ability to reason collectively, ie solve for multiple, dependent variables at the same time. A collective rule is one that contains references to multiple random variables. Bellow are general collective rules used in this model that are often used for entity resolution.

Transitive equality is a rules that is commonly desired in entity resolution problems:

$$\textsc{Same}(Ref1, Ref2) \ \& \ \textsc{Same}(Ref2, Ref3) \rightarrow \textsc{Same}(Ref1, Ref3)$$

A sparsity rule encodes the intuition that if we find a match for one reference, we are unlikely to find another match for that entity. This rule works well in an environment where a reference will typically match at most one other reference.

$$\textsc{Same}(Ref1, Ref2) \rightarrow !\textsc{Same}(Ref1, Ref3)$$

6.2.3 Domain Information

PSL is able to easily encode domain or expert information into a model. Unlike the other rules, the ones in this section are not general to all entity resolution problems. These rules are specific to the movie/IMDB domain.

In this rule we can encode the domain knowledge that actors will often appear alongside the same costars in different movies. For example, a cast can persist through sequels or actors with good chemistry will get cast together.

$$\begin{aligned}\textsc{InMovie}(Ref1, Movie1) \;\&\; &\textsc{InMovie}(CoStar, Movie1) \\ \&\; &\textsc{InMovie}(Ref2, Movie2) \;\&\; \textsc{InMovie}(CoStar, Movie2) \\ \rightarrow\; &\textsc{Same}(Ref1, Ref2)\end{aligned}$$

In the following rule we can encode the information that each actor will only act under one name in a single movie. Although an actor may take on multiple roles in a movie, they will not use multiple names for themselves. Therefore if an actor entity matches one reference from a movie, that actor will not match another reference from that same movie.

$$\begin{aligned}\textsc{InMovie}(Ref1, Movie) \;\&\; &\textsc{InMovie}(Ref2, Movie) \\ \&\; &\textsc{Same}(Ref1, Ref3) \\ \rightarrow\; &!\textsc{Same}(Ref2, Ref3)\end{aligned}$$

6.2.4 Negative Prior

Entity resolution is typically a very sparse problem, with a low percentage of reference pairs resolving to the same entity. Therefore, we will include a negative prior in our model. Without any additional evidence, we believe that two references should not resolve to the same entity.

$$!\textsc{Same}(Ref1, Ref2)$$

6.2.5 Constraints

PSL also supports hard constraints. Here we can encode symmetry on actor references.

$$Same(Ref1, Ref2) = Same(Ref2, Ref1)$$

6.3 Optimization of PSL Formulations

One of the main concerns with SRL approaches is scalability. Because the ground networks produced by an SRL are polynomial in the number of references, running a collective solution

can be difficult. For example, the transitive equality rule arbitrarily chooses three authors from the dataset. Therefore, that rule produces $\binom{621}{3} \approx 240$ million ground rules. Below are several several steps that PSL takes to be able to reason at scale.

6.3.1 Blocking

As previously mentioned, the number of potentials in a MRF can quickly grow prohibitively large. To limit the number of potentials, *blocks* [22] or *canopies* [21] can be constructed. Blocks and canopies use problem specific heuristics to eliminate infeasible groundings. In PSL, blocking structures can be constructed by treating block definitions as data and including them in the rules. In this case, potentials with components outside of the block are trivial and removed during the grounding phase.

In this problem, we can use heuristics to decide which pairs of references we can immediately throw out. Thus, we removed all candidate pairs that appeared in the same movie together. The basis for this removal is that an actor/actress would not appear under two different aliases in the same movie. Without this heuristic, we would have 384,400 pairs; instead, we have 192,510 pairs.

But putting these candidate reference pairs in the CandSame predicate, we can rewrite our transitive equality rule as follows:

$$\begin{aligned}\textsc{CandidateSame}(Ref1, Ref2) \;\&\; \textsc{CandidateSame}(Ref2, Ref3) \;\&\; \textsc{CandidateSame}(Ref1, Ref3) \\ \&\; \textsc{Same}(Ref1, Ref2) \;\&\; \textsc{Same}(Ref2, Ref3) \\ \rightarrow\; \textsc{Same}(Ref1, Ref3)\end{aligned}$$

6.3.2 Results

The aforementioned PSL program generates around 130 million ground rules and completes in 25 minutes. The server that these benchmarks were performed on was a CPU with 24 cores clocked at 2.2 GHz and 380 GB of RAM.

To put these results into context, consider two other SRL frameworks: Tuffy and FoxPSL. Tuffy is probably the most commonly used implementation of Markov Logic Networks (MLNs) [23]. MLNs are similar to HL-MRFs, but are discrete. Running on the same machine as these experiments, a Tuffy program of 2.5 million ground rules ran in about 100 minutes [6]. FoxPSL started as a distributed implementation of PSL with some different assumptions and additional features [19]. PSL and FoxPSL have diverged substantially. FoxPSL ran a program of 12.5 million ground rules distributed over four machines in about 30 minutes.

Framework	Distributed?	Program Size (M)	Runtime (min)
Tuffy	No	2.5	108
FoxPSL	Yes	12.5	33
PSL	No	128.8	25

Table 6.1. Performance of two popular SRL frameworks compared to PSL.

Chapter 7

Conclusion

Our benchmarking efforts are designed to help organizations with large data ingestion missions. DOE and its partners have such missions. At the time of this writing we have not completed a scalable method for assessing false positives, so we do not have conclusive findings regarding simple vs. sophisticated entity resolution algorithms. However, we believe that efficient ways of computing ambiguous relationship probabilities exist. If this is correct, then our process could be used by large organizations as follows.

First, they would generate representative samples of their data small enough to be assessed by our methods, but still on the order of millions of entities and relationshps. Next, they would use our methods to assess the inherent difficulty of their instances. Finally, they would apply suites of candidate ER algorithms to these datasets and compare the inherent per-entity difficulty of the problem with algorithm performance. Thus, rather than relying on static, published results from the literature, participating organizations would make their decisions based on their own data sources.

References

[1] IMDB dataset. `https://www.imdb.com/interfaces/`. Accessed: 2017-09-01.

[2] LINQS entity resolution for big data. `https://linqs.soe.ucsc.edu/node/23`.

[3] LINQS home page. `https://linqs.soe.ucsc.edu`. Accessed: 2018-09-17.

[4] Probabilistic soft logic (software). `https://psl.linqs.org`. Accessed: 2018-09-17.

[5] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[6] Eriq Augustine and Lise Getoor. A comparison of bottom-up approaches to grounding for templated markov random fields. In *SysML*, 2018.

[7] Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss markov random fields and probabilistic soft logic. *arXiv preprint arXiv:1505.04406*, 2015.

[8] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: a generic approach to entity resolution. *The VLDB JournalThe International Journal on Very Large Data Bases*, 18(1):255–276, 2009.

[9] Jonathan W Berry, Bruce Hendrickson, Simon Kahan, and Petr Konecny. Software and algorithms for graph queries on multithreaded architectures. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–14. IEEE, 2007.

[10] Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):5, 2007.

[11] Indrajit Bhattacharya and Lise Getoor. Query-time entity resolution. *Journal of Artificial Intelligence Research*, 30:621–657, 2007.

[12] Jack Dongarra and Michael A Heroux. Toward a new metric for ranking high performance computing systems. *Sandia Report, SAND2013-4744*, 312:150, 2013.

[13] Jack J Dongarra. *Performance of various computers using standard linear equations software*. University of Tennessee. Computer Science Department, 1993.

[14] Jack J Dongarra, James R Bunch, Cleve B Moler, and Gilbert W Stewart. *LINPACK users' guide*, volume 8. Siam, 1979.

[15] Jack J Dongarra, Hans W Meuer, and Erich Strohmaier. Top500 report 1993. *SUPERCOMPUTER, volumne*, 12, 1994.

[16] Mauricio A Hernández and Salvatore J Stolfo. The merge/purge problem for large databases. In *ACM Sigmod Record*, volume 24, pages 127–138. ACM, 1995.

[17] Dmitri V Kalashnikov and Sharad Mehrotra. A probabilistic model for entity disambiguation using relationships. In *SIAM International Conference on Data Mining (SDM). Newport Beach, California*, pages 21–23, 2005.

[18] Pigi Kouki, Jay Pujra, Christopher Marcum, Laura Koehly, and Lise Getoor. Collective entity resolution in multi-relational familial networks. *Knowledge and Information Systems (KAIS)*, 2018.

[19] Sara Magliacane, Philip Stutz, Paul Groth, and Abraham Bernstein. foxpsl: A fast, optimized and extended psl implementation. *International Journal of Approximate Reasoning*, 67(Supplement C):111 – 121, 2015.

[20] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *arXiv preprint arXiv:1603.09320*, 2016.

[21] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 169–178, New York, NY, USA, 2000. ACM.

[22] Howard B. Newcombe and James M. Kennedy. Record linkage: Making maximum use of the discriminating power of identifying information. *Commun. ACM*, 5(11):563–566, November 1962.

[23] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *PVLDB*, 4(6):373–384, March 2011.

[24] Jay Pujara and Lise Getoor. Generic statistical relational entity resolution in knowledge graphs. In *Sixth International Workshop on Statistical Relational AI*. IJCAI 2016, 2016. On arXiv: https://arxiv.org/abs/1607.00992.

[25] Vibhor Rastogi, Nilesh Dalvi, and Minos Garofalakis. Large-scale collective entity matching. *Proceedings of the VLDB Endowment*, 4(4):208–218, 2011.

[26] Parag Singla and Pedro Domingos. Entity resolution with markov logic. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 572–582. IEEE, 2006.

[27] William E Winkler. Overview of record linkage and current research directions. In *Bureau of the Census*. Citeseer, 2006.

DISTRIBUTION:

1	MS 0359	D. Chavez, LDRD Office, 1911
1	MS 0899	Technical Library, 9536 (electronic copy)


Sandia National Laboratories