LINDA: Distributed Web-of-Data-Scale Entity Matching

Christoph Böhm
Hasso Plattner Institute
Potsdam, Germany
christoph.boehm@
hpi.uni-potsdam.de

Gerard de Melo
ICSI Berkeley
Berkeley, CA, USA
demelo@
icsi.berkeley.edu

Felix Naumann
Hasso Plattner Institute
Potsdam, Germany
felix.naumann@
hpi.uni-potsdam.de

Gerhard Weikum
Max Planck Institute
Saarbrücken, Germany
weikum@
mpi-inf.mpg.de

ABSTRACT

Linked Data has emerged as a powerful way of interconnecting structured data on the Web. However, the cross-linkage between Linked Data sources is not as extensive as one would hope for. In this paper, we formalize the task of automatically creating "sameAs" links across data sources in a globally consistent manner. Our algorithm, presented in a multi-core as well as a distributed version, achieves this link generation by accounting for joint evidence of a match. Experiments confirm that our system scales beyond 100 million entities and delivers highly accurate results despite the vast heterogeneity and daunting scale.

Categories and Subject Descriptors

I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms

1. (NOT SO) LINKED OPEN DATA

Linked Open Data (LOD) is emerging as a way of interconnecting structured-data sources on the Internet and creating a "Web of Data". In total, the Web of Data currently contains about 30 billion triples. The key point of LOD is to provide extensive cross-linkage between sources at the level of entities. For example, two `sameAs` links suffice to connect data about the director David Lynch in DBpedia, Freebase, and the BBC. This way, we can answer join queries that require biographic data from DBpedia as well as data about compositions from the BBC. Clearly, this has enormous potential. Unfortunately, this cross-linkage between LOD sources is not nearly as extensive as one would hope. Although exact numbers are not known, an estimate for the number of `sameAs` links is in the order of 500 million. A large number of trivial links exist between major knowledge bases like DBpedia, Freebase and Yago, which are derived from Wikipedia and can use article titles as a common denominator. Our aim is to develop fully automated domain-independent methods that discover high-quality links, while scaling to the immense proportions of the Web of Data.

At first glance, this seems to be identical to the classical *record linkage* task, also known as *entity resolution*. The problem has received much attention in the literature [1, 8]. A closer look, however, reveals fundamental differences between the typical record-linkage setting and our problem of entity matching in LOD. First, the fine-grained and loose-schema nature of RDF triples makes it much harder to identify similarity features. Second, in contrast to the traditional setting, we are faced with the vast heterogeneity of LOD sources, spanning many different domains. Third, we deal with a much larger number of sources. This is an additional challenge, but also an opportunity, since we can potentially exploit `sameAs` transitivity over many sources. Finally, many billions of triples is a daunting scale that has not yet been tackled in a joint approach. Recent work geared towards LOD [5, 6, 9, 14] has either only been applied to small datasets or has partitioned the entity matching into separate jobs without accounting for their interactions.

Our approach is based on an optimization model that captures the *joint* evidence for entities in one source matching entities in other sources. The evidence is based on neighboring nodes of an entity and those of another entity to which a `sameAs` link could possibly exist. Consider the example of the entity `David Lynch` on LinkedMDB and a candidate entity in Freebase. For inferring equivalence, we consider both sides' attributes such as birthplace, awards and relationships to movies, compositions, etc.

This situation motivates our joint-reasoning approach: After matching `David Lynch` correctly, we have more evidence to also choose the right `Mulholland Drive` in Freebase, i.e., the movie and not the street in the Los Angeles area. This reasoning proceeds recursively, i.e., a decision about equivalence of neighboring entities may in turn affect further neighboring entities.

This paper makes the following technical contributions: (1) A new optimization model for joint evidence and consistency of `sameAs` linkage between LOD entities. (2) An efficient and scalable algorithm for computing an approximate solution of the optimization problem, with a multi-core as well as a distributed MapReduce-based version. (3) Large-scale experiments that demonstrate the viability of our approach on graphs with $>$ 100 millions of entities.

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

2. PROBLEM FORMALIZATION

Goal. Given Linked Data RDF triples, we would like to match entity identifiers that come from different sources and represent the same real-world entity.

Input. Our input is a set of RDF triples of the form (s, p, o), i.e., with subject, predicate, and object. These are cast into an *entity graph* G, with nodes corresponding to entity URIs, and edge labels representing triple predicates.

Desired Output. Our output is a square 0-1 matrix $\boldsymbol{X}$, stating whether any two URIs represent the same entity or not. The output is subject to certain constraints. We require the `sameAs` relationship to be symmetric and transitive, which corresponds to establishing equivalence classes of entities. We do not aim to discover `sameAs` links within a single data source, as single sources are usually much better maintained than links across sources. Our entity matching algorithm makes joint decisions for multiple URI pairs when producing `sameAs` links. It is initialized with a *prior similarity* between entities, based on the immediate neighborhoods of the URIs. The algorithm iteratively considers similarities between entities to reinforce or invalidate the matching between two URIs. Output matchings are stored in an *assignment matrix* $\boldsymbol{X}$, while the dynamically re-computed similarity values are tracked in a *similarity matrix* $\boldsymbol{Y}$.

Entity Graph and Assignment Matrix. For the graph, we consider only triples (s, p, o) where s, p, and o are URIs. Literals are compiled into a set of values $L(s)$. The data source from which a URI a originates is denoted as $S(a)$.

Definition 1. An *entity graph* is a directed multigraph $G = (V, E)$ with a set of nodes V representing URIs and labeled edges in E capturing their relationships. Given a set T of triples, we set $V = \{s \mid (s, p, o) \in T\} \cup \{o \mid (s, p, o) \in T\}$, and $E = \{(s, p, o) \in T\}$.

Definition 2. Given an entity graph $G = (V, E)$, an *assignment matrix* $\boldsymbol{X}$ is a symmetric $n \times n$ matrix with $n = |V|$ and $x_{a,b} \in \{0, 1\}$. An entry $x_{a,b}$ states whether our algorithm outputs "a `sameAs` b".

Definition 3. An assignment matrix $\boldsymbol{X}$ is *consistent* if it satisfies the following constraints:

1. **Reflexivity** $\forall a \in V : x_{a,a} = 1$
2. **Symmetry** $\forall a, b \in V : x_{a,b} = x_{b,a}$
3. **Transitivity** $\forall a, b, c \in V : x_{a,b} \cdot x_{b,c} \leq x_{a,c}$
4. ***optional:*** **Unique mapping per data source** $\forall a, s \neq S(a) : \sum_{b:S(b)=s} x_{a,b} \leq 1$

The entity graph is the data structure that captures our input. The output is gradually built into an assignment matrix subject to constraints. The optional last constraint states that an entity a cannot simultaneously match two entities from the same second source.This constraint allows us to focus on cross-source links and, in practice, picking only the best match within a dataset avoids many false positives.

Objective Function. Let $\text{sim}(a, b, G, \boldsymbol{X})$ be a similarity function between two entities a and b that may depend on the entity graph G as well as the current $\boldsymbol{X}$. For constant G and $\boldsymbol{X}$, sim should be a semimetric and return scores in $[-\infty, +\infty]$: positive scores for likely entity matches and negative scores for likely non-matches. To quantify the quality of the output $\boldsymbol{X}$, we define the following objective.

Definition 4. Given an entity graph $G = (V, E)$ and a similarity function sim, the *maximum consistent assignment problem* consists in choosing a consistent assignment matrix $\boldsymbol{X}$ with values $x_{a,b} \in \{0, 1\}$ that maximizes

$$\sum_{a,b \in V : S(a) \neq S(b)} x_{a,b} \ \text{sim}(a, b, G, \boldsymbol{X}).$$

The sim function often becomes negative and deliberately depends on the assignment matrix $\boldsymbol{X}$: For instance, in our previous example, the similarity of two `Mulholland Drive` URIs depends on whether two other (related) URIs both represent `David Lynch`. By reduction from the CLIQUE problem one can show that this problem is *NP-hard*.

3. ASSIGNMENT ALGORITHMS

We now present an algorithm that computes a consistent assignment matrix $\boldsymbol{X}$ while attempting to maximize the objective score. This first algorithm can benefit from multiple cores during operation on a single machine. For scalability, our algorithm iteratively adjusts the values in $\boldsymbol{X}$ with carefully chosen, greedy improvements in the objective function. $\boldsymbol{X}$ is initialized with 1s on the diagonal and 0s otherwise. In addition, we use a *similarity matrix* $\boldsymbol{Y}$ with pairwise, real-valued similarity values. $\boldsymbol{Y}$ is initialized with values that reflect the pair-wise similarity of URIs (see Sec. 4). Given the previous iteration's values of $\boldsymbol{X}$, we compute new similarity values for $\boldsymbol{Y}$, which is maintained in a priority queue and used to adjust elements of $\boldsymbol{X}$. Both $\boldsymbol{X}$ and $\boldsymbol{Y}$ are sparse symmetric matrices that do not reside entirely in memory. $\boldsymbol{Y}$ entries need to be materialized only on demand for the priority queue; negative scores do not need to be retained. So the $\boldsymbol{Y}$ matrix is a purely conceptual construct only.

Algorithm 1 describes the method more formally. In line 3, the updatable priority queue Q stores the initial similarity scores $y_{a,b} = \text{sim}(a, b, G, \boldsymbol{X})$. In practice, sim yields negative scores for most entity pairs. With MapReduce, we can efficiently determine entity pairs with positive scores without computing a quadratic number of similarities (see Sec. 4).

The algorithm then repeatedly dequeues the entity pair a, b with the highest similarity score from Q and sets $x_{a,b}$ to 1. To ensure transitivity, it further considers all *equivalents* $E(a, \boldsymbol{X}) = \{a' | x_{a,a'} = 1\}$ of a and $E(b, \boldsymbol{X}) = \{b' | x_{b,b'} = 1\}$ of b already identified with them. The for-loop (line 7) essentially merges the equivalence classes of a and b.

Then, the algorithm determines pairs of entities which may need updating (line 9) due to modifying $\boldsymbol{X}$. For the similarity function we define later in Sec. 4, these are the candidate pairs involving entities in $E(a, \boldsymbol{X})$ as well as entity graph neighbors of entities in $E(a, \boldsymbol{X})$. The algorithm adjusts the scores of these pairs in $\boldsymbol{Y}$ in parallel, making use of multiple cores. The new similarity scores $y_{a',b'}$ computed in line 12 reflect the new $\boldsymbol{X}$with $E(a, \boldsymbol{X}) = E(b, \boldsymbol{X})$, highlighting the joint mapping strategy.

For space reasons, we omit the proofs that our algorithm produces a consistent matrix $\boldsymbol{X}$ and is guaranteed to converge for well-behaved similarity functions. Convergence to local maxima can be overcome with simple randomization or stochastic optimizations.

Assignment with MapReduce. The previously described Algorithm 1 is bounded by the number of CPU cores available for parallelism and by memory available to store $\boldsymbol{X}$ and (parts of) the priority queue Q.

Algorithm 1 Multi-Core Assignment Algorithm

```
1: procedure LINDA(G)
2:     X ← I_|V|                                   ▷ identity matrix
3:     Q ← initial similarities                    ▷ with MapReduce
4:     while Q non-empty do
5:         dequeue (y_{a,b}, {a, b}) with highest y_{a,b} from Q
6:         X_0 ← X
7:         for all a' ∈ E(a, X_0), b' ∈ E(b, X_0) do   ▷ assignment
8:             x_{a',b'} ← 1, x_{b',a'} ← 1
9:         S ← {(a', b') | sim(a', b', X, G) ≠ sim(a', b', X_0, G)}
10:        for all (a', b') ∈ S : x_{a',b'} = 0 do in parallel
11:            fetch (y_{a',b'}, {a', b'}) from Q
12:            y* = sim(a, b, GX)
13:            if y* ≠ y_{a',b'} then               ▷ similarity changed
14:                remove (y_{a',b'}, {a', b'}) from Q
15:                if y* > 0 then enqueue (y*, {a', b'}) in Q
16:    return X
```

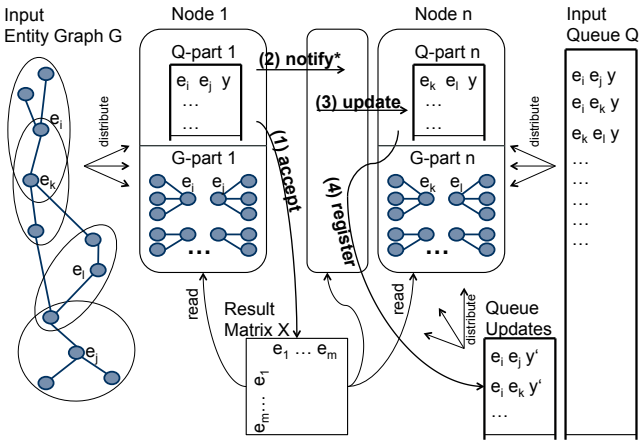

Figure 1: MapReduce Workflow

To overcome these bounds, we report on a MapReduce-based version of the assignment algorithm that allows scaling our solution to immense amounts of data, since today's large cluster sizes range in the thousands. With MapReduce, a problem is divided into independent map tasks that consume and emit key-value-pairs as well as reduce tasks that aggregate intermediate output. Fig. 1 illustrates the workflow of our MapReduce-based approach.

Consider an input entity graph G (Fig. 1, left) distributed across a cluster. Each node holds a portion Q_i of the queue Q and a respective partition G_i of G. An entity graph partition G_i comprises all information for vertices in queue partition Q_i. Partitions are stored as sorted lists, which enables fast merge-join-like access instead of the graph and the queue being shuffled across the cluster. The only information sent from mappers to reducers are messages about which pairs of vertices require similarity score recomputations.

Each mapper reads Q_i and stores the top K entries in a buffer B. We refer to K as the *acceptance rate*. Mappers also forward messages from previous phases. After mapper completion, a procedure accepts all pairs a', b' of equivalents for entries of B for the resulting $\boldsymbol{X}$ (step (1) in Fig. 1).

Remember that the acceptance of a `sameAs` edge between vertices a and b induces a series of score adjustments. In our setup, reducers handle specific graph partitions. Hence, if t is the number of edge hops to nodes whose sim scores may change, then the algorithm uses t reducer steps to trigger the respective score recomputations. For our sim function (see Sec. 4), a new `sameAs` edge induces two sorts of recomputations, i.e., (1) for the candidate pairs involving entities in $E(a, \boldsymbol{X})$ or $E(b, \boldsymbol{X})$ and (2) for the entity graph neighbors of entities in $E(a, \boldsymbol{X})$ or $E(b, \boldsymbol{X})$. It thus requires two steps: one to reach the equivalents (*notify* step in Fig. 1), another to reach their neighbors in the graph (*update* step).

After accepting pairs $a', b' \in E(a, \boldsymbol{X}) \times E(b, \boldsymbol{X})$, the mapper emits two sorts of notification messages (step (2) in Fig. 1): `notification` triggers similarity score updates for entity graph neighbors of a' and b'; `updateTargets` triggers candidate pair updates for a' and b'.

A reducer reacts according to the message it receives. Given `notification`, it emits `update` messages for respective neighbors. It additionally emits `update`s for all affected Q_i entries. Given an `updateTargets` message, it emits `update`s for all affected Q_i entries on other compute nodes. This is step (3) in Fig. 1. The actual update of queue entries is triggered when a reducer *receives* such an `update` message. Then, the reducer checks whether the unique mapping constraint still holds and computes the new y' value (step (4) in Fig. 1).

Note that the round-trip of the effect of a new entry in $\boldsymbol{X}$ takes two iterations (since we do two edge traversals). Therefore, the unique mapping constraint may not be fulfilled when accepting `sameAs` edges while queue updates for decisions from previous mappers are still pending.

4. THE LINDA SYSTEM

The LINDA (LINked Data Alignment) system implements the these two assignment algorithm variants in combination with judiciously designed similarity functions. LINDA computes two kinds of similarities between entities a, b: (i) a *prior similarity* sim_0 based on literals and constraints, which is computed once in advance and used to initialize the similarity matrix $\boldsymbol{Y}$ and later as a smoothing prior, and (ii) a *contextual* similarity sim_C, which is recomputed in each iteration and considers the current state of the assignment matrix $\boldsymbol{X}$. The two similarity measures are combined to an overall *similarity score* $\text{sim}(a, b, G, \boldsymbol{X})$. Given an entity graph $G = (V, E)$, an assignment matrix $\boldsymbol{X}$, and two parameters α, θ, the *similarity score* for entities $a, b \in V$ is:

$$\text{sim}(a, b, G, \boldsymbol{X}) = \text{sim}_0(a, b) + \alpha\, \text{sim}_C(C(a), C(b), G, \boldsymbol{X}) - \theta.$$

$C(a)$ and $C(b)$ denote the contexts of a, b and are defined later. The parameter α controls the contextual influence and θ is used for renormalization to values around 0 – positive scores should reflect likely mappings and negative scores imply dissimilarities as required by Def. 4. We experimentally found $\alpha = 1.0$ to perform well.

Prior Similarities. Prior similarities $\text{sim}_0(a, b)$ reflect the direct evidence of two entities matching. Given two entities a and b and their sets of normalized literal n-grams $N_a = N(L(a))$, $N_b = N(L(b))$, the prior similarity $\text{sim}_0(a, b)$ is set to $-\infty$ if they stem from the same data source, and to 0 if they have no common n-grams. In all other cases, $\text{sim}_0(a, b)$ is computed as

$$\text{sim}_0(a, b) = \frac{|N_a \cap N_b|}{\min(|N_a|, |N_b|) + \ln(|\,|N_a| - |N_b|\,| + 1)}.$$

Table 1: OAEI benchmark results

System	Person1		Person2		Restaur.		IIMB-s	
	Pr.	Rec.	Pr.	Rec.	Pr.	Rec.	Pr.	Rec.
LINDA	1.00	1.00	1.00	0.23	1.00	0.63	0.90	0.54
ASMOV	1.00	0.77	0.98	0.14	0.70	0.70	0.86	0.82
RiMOM	1.00	1.00	0.95	0.99	0.86	0.77	0.96	0.83
CODI	0.87	0.96	0.83	0.23	0.72	0.72	0.91	0.84

Contextual Similarities Given an entity graph $G = (V, E)$, the *context* $C(a)$ of an entity a is a set of *context tuples* (r, n, w), where r is a predicate (edge label in the entity graph), n is a neighboring entity of a, and w is a numeric weight. The context of an entity a includes objects n of triples with a as subject and subjects n of triples with a as object. The weights $w = \frac{1}{\log \text{freq}(r,n)}$ are higher for less frequent and thus more discriminative context tuples. Given an entity graph G, an assignment matrix $\boldsymbol{X}$, and two entities a, b with contexts $C_a = C(a)$, $C_b = C(b)$, the *contextual similarity* $\text{sim}_C(C_a, C_b, G, \boldsymbol{X})$ is defined as

$$\begin{cases} \sum_{\substack{(r_a,n_a,w_a)\\ \in C_a}} \max_{\substack{(r_b,n_b,w_b)\\ \in C_b}} w_a \cdot x_{n_a,n_b} \cdot \text{sim}(r_a, r_b) & \text{if } |C_a| \leq |C_b| \\ \sum_{\substack{(r_b,n_b,w_b)\\ \in C_b}} \max_{\substack{(r_a,n_a,w_a)\\ \in C_a}} w_b \cdot x_{n_a,n_b} \cdot \text{sim}(r_a, r_b) & \text{otherwise.} \end{cases}$$

Intuitively, this function finds matching pairs of context tuples and sums up their similarity values. It determines the smaller set C_s and then sums up weighted similarities for the best matching context tuples for each tuple $(r, n, w) \in C_s$. The similarity of individual context tuples (r_a, n_a, w_a), (r_b, n_b, w_b) depends on the current state of the entity matching and on predicate similarities $\text{sim}(r_a, r_b)$.

5. EXPERIMENTS

All input data is first processed with MapReduce to create and store the entity graph, prior similarities as well as context weights and predicate similarities. The algorithm itself uses a sparse matrix representation to store $\boldsymbol{X}$, and a priority queue to keep relevant entries in $\boldsymbol{Y}$. The multi-core version uses a thread pool for parallelization and was run on a single 80-core server. The distributed version of the algorithm operated on a 1+9-node-cluster.

Precision and Recall. We compared LINDA with traditional small-scale instance matching systems using the OAEI 2010 benchmarks. Table 1 compares LINDA ($\theta = 0.2$) with all systems that participated for all four datasets according to the official website (*oaei.ontologymatching.org*). The DI benchmark unfortunately could not be tested, as its reference alignments included URIs missing in the supplied input data. Overall, our system delivers very competitive results although it uses the same set of parameters across all datasets. Competing systems like RiMOM use different sets of highly tuned methods and settings for each of the datasets. LINDA favors precision over recall to ensure high quality results, but a trade-off is possible by lowering the θ parameter or by choosing other prior similarity functions.

Large-Scale Experiment. Next, we tested our system on very large volumes of data by creating an *augmented Billion Triple Challenge Dataset* (**BTC+**). The original BTC 2010 dataset (*http://km.aifb.kit.edu/projects/btc-2010*) contains ∼3.1 billion RDF quadruples crawled from ∼2,500 different sources (624 GB of data). We increased the size of the dataset by including all triples from the *DBpedia* dataset. For this very large dataset, building the database with prior similarities took 1 day. Removing provenance information, duplicate triples, RDF blank nodes as well as reification statements resulted in ∼350 million unique triples describing ∼95 million different URIs. Computing initial similarities for the algorithm's queue took 1h, and after that the algorithm ran for 7.5h using the same server setup as above.

At $\theta = 1.0$, LINDA delivered a total of 2,601,392 `sameAs` links, not including identity links, with a sampled precision of 0.83 ± 0.06. At $\theta = 0.25$, LINDA delivered 12.3 million links, but the accuracy drops to 66% due to the particularly noisy nature of this dataset. We observed in particular that the joint inference strategy of our algorithm allowed us to infer many new mappings that initially had negative similarity scores. A few initial matches suffice to subsequently find other matches, and so on. The greedy nature of our algorithm works well in conjunction with our unique mappings per data source constraint, as it allows our algorithm to select the most likely mappings, while ignoring other mappings into the same data source that only coincidentally have positive similarity scores. Inaccurate mappings often seemed to result from the noisiness of the BTC dataset, which includes a lot of metadata entities that all look the same. Note that we cannot compare our results with those of existing systems: To the best of our knowledge no existing joint matching system is able to process the same amount of data.

Distributed Web-Scale Experiment. We evaluated the MapReduce algorithm for LINDA on a web-scale dataset BTC++, which comprised all triples from the 2011 BTC data, DBpedia (version 3.6), Freebase, Yago, and Geonames.org (as of mid 2011), in total 566.2 million unique triples describing 115.5 million URIs. The initial queue comprises 154.5 million entity pairs, i.e., 4.07GB of binary data in HDFS. The BTC++ entity graph amounts to 39.17 GB in HDFS (including stored overlaps). The degrees of the graph vertices are very high, sometimes above 100,000.

We conducted experiments for a varying number of compute nodes n as well as varying acceptance rates K. We measured the time and the number of accepted links per iteration. Also, we kept track of the number of update messages and queue updates registered. For the number of messages sent across the cluster, we observed large numbers (dozens of millions) during early iterations. The message traffic levels out after 10-20 iterations (depending on n and K). As for the actual updates, we observed thousands of updates during early iterations, also leveling out later on.

Next, we compare output sizes with respect to runtime. Figure 2 shows the cumulative runtime (solid lines) and the number of links (dashed lines) computed by our distributed approach ($K = 100$). Clearly, given a specific iteration i as well as fixed n and K, the output size grows, since $i*n*K$ is its lower bound. However, as the acceptance of a single mapping induces the merge of the respective equivalence classes, the output size increases beyond this. Note that at some point (iteration 27 for $n = 60$), this increase is steeper than the runtime's increase – essentially leading to more links per time unit. For $n = 40$ and $n = 60$ ($K = 100$) we ran the

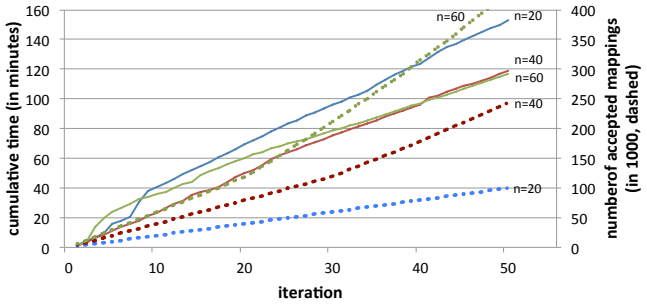

Figure 2: #Links vs. runtime over iterations

algorithm until the our output comprised five million links for the BTC++ data. This takes 500 or 325 iterations, respectively.

6. RELATED WORK

Record Linkage. Classical record linkage typically relies on similarity measures between strings and between records [8, 1]. Smart blocking techniques are often employed to reduce similarity comparisons [9, 11, 15]. More recently, research has shifted to reasoning on entire groups of interrelated records [4]. The purest approaches employ collective relational learning, based on probabilistic graphical models [12, 16]. These work well for highly structured data but are not yet geared towards web-scale datasets.

Semantic Web. A highly related task to our problem is ontology alignment (see, e.g., [2]), which focuses on aligning *classes* rather than large numbers of *instances* of classes. Systems typically rely on ontological constraints that are often unavailable for Linked Data. Existing systems for connecting Linked Data sources require humans to specify data-specific rules or training data and have not been applied jointly across a broad range of datasets [14, 3, 6, 13]. The *sameas.org* service provides `sameAs` links that have been manually collected. In contrast, our approach aims at automatic unsupervised discovery of links across the LOD cloud.

Distributed Entity Matching. MapReduce parallelization has been applied to standard blocking techniques [7]. A recent proposal for scaling up joint inference partitions the input into neighborhoods, and then parallel computations per neighborhood are periodically reconciled by message passing [10]. This line of work is viable for highly structured and clearly typed records but does not consider densely connected Linked Data triples with high heterogeneity at Web scale. Recently, MapReduce has been investigated for matching in the Linked Data world [14, 5]. However, these systems simply distribute independently made matching decisions. This is different to our joint model, where decisions in one partition can affect decisions in other partitions. To the best of our knowledge, there are no reports on joint mapping experiments at the scale of our input data with more than 110 million unique entity URIs.

7. CONCLUSION

We have presented the first fully automatic system for joint entity matching in the Web of Data that is able to scale beyond the size of the Billion Triples data. Unlike previous approaches, it is designed to operate on all LOD sources together, without source-specific customization. Our experiments demonstrate that the LINDA system successfully harnesses joint evidence and constraints. In our experiments, we did not use pre-existing `sameAs` links as additional evidence, but when deploying LINDA in practice it would be straightforward to exploit such knowledge for higher-quality output. We anticipate that joint linking approaches that consider the entire LOD cloud when making decisions will be crucial as the Web of Data keeps growing.

8. REFERENCES

[1] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19:1–16, 2007.

[2] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, 2007.

[3] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. J. Miller, and M. Wang. A framework for semantic link discovery over relational data. In *Proc. CIKM*, 2009.

[4] M. Herschel, F. Naumann, S. Szott, and M. Taubert. Scalable iterative graph duplicate detection. *TKDE*, 2011.

[5] A. Hogan *et al.* Scalable and distributed methods for entity matching, consolidation and disambiguation over linked data corpora. *J Web Semantics*, 10, 2012.

[6] W. Hu, J. Chen, and Y. Qu. A self-training approach for resolving object coreference on the semantic web. In *Proc. WWW*, 2011.

[7] L. Kolb, A. Thor, and E. Rahm. Block-based Load Balancing for Entity Resolution with MapReduce. In *Proc. CIKM*, 2011.

[8] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *DKE*, 69:197–210, 2010.

[9] G. Papadakis *et al.* Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data. In *Proc. WSDM*, 2012.

[10] V. Rastogi, N. Dalvi, and M. Garofalakis. Large-scale collective entity matching. *PVLDB*, 4:208–218, 2011.

[11] L. Shu, A. Chen, M. Xiong, and W. Meng. Efficient spectral neighborhood blocking for entity resolution. In *Proc. ICDE*, 2011.

[12] P. Singla and P. Domingos. Entity resolution with markov logic. In *Proc. ICDM*, 2006.

[13] F. M. Suchanek, S. Abiteboul, and P. Senellart. PARIS: Probabilistic alignment of relations, instances, and schema. *PVLDB*, 5(3):157–168, 2011.

[14] J. Volz *et al.* Discovering and maintaining links on the Web of Data. In *Proc. ISWC*, 2009.

[15] S. E. Whang and H. Garcia-Molina. Joint entity resolution. In *Proc. ICDE*, 2012.

[16] M. L. Wick, A. Culotta, K. Rohanimanesh, and A. McCallum. An entity based model for coreference resolution. In *Proc. SDM*, 2009.