Scalable and Distributed Methods for Entity Matching, Consolidation and Disambiguation over Linked Data Corpora

Aidan Hogan [a], Antoine Zimmermann [b], Jürgen Umbrich [a], Axel Polleres [c], Stefan Decker [a],

[a] *Digital Enterprise Research Institute, National University of Ireland, Galway*
[b] *INSA-Lyon, LIRIS, UMR5205, F-69621, France*
[c] *Siemens AG Österreich, Siemensstrasse 90, 1210 Vienna, Austria*

Abstract

With respect to large-scale, static, Linked Data corpora, in this paper we discuss scalable and distributed methods for entity consolidation (aka. smushing, entity resolution, object consolidation, etc.) to locate and process names that signify the same entity. We investigate (i) a baseline approach, which uses explicit `owl:sameAs` relations to perform consolidation; (ii) extended entity consolidation which additionally uses a subset of OWL 2 RL/RDF rules to derive novel `owl:sameAs` relations through the semantics of inverse-functional properties, functional-properties and (max-)cardinality restrictions with value one; (iii) deriving weighted concurrence measures between entities in the corpus based on shared inlinks/outlinks and attribute values using statistical analyses; (iv) disambiguating (initially) consolidated entities based on inconsistency detection using OWL 2 RL/RDF rules. Our methods are based upon distributed sorts and scans of the corpus, where we deliberately avoid the requirement for indexing all data. Throughout, we offer evaluation over a diverse Linked Data corpus consisting of 1.118 billion quadruples derived from a domain-agnostic, open crawl of 3.985 million RDF/XML Web documents, demonstrating the feasibility of our methods at that scale, and giving insights into the quality of the results for real-world data.

Key words: entity consolidation, web data, linked data, rdf

1. Introduction

Over a decade since the dawn of the Semantic Web, RDF publishing has begun to find some traction through adoption of Linked Data best practices as follows:

(i) use URIs as names for things (and not just documents);
(ii) make those URIs dereferencable via HTTP;
(iii) return useful and relevant RDF content upon lookup of those URIs;
(iv) include links to other datasets.

The Linked Open Data project has advocated the goal of providing dereferencable machine readable data in a common format (RDF), with emphasis on the re-use of URIs and inter-linkage between remote datasets—in so doing, the project has overseen exports from corporate entities (e.g., the BBC, BestBuy, Freebase), governmental bodies (e.g., the UK Government, the US government), existing structured datasets (e.g., DBPedia), social networking sites (e.g., flickr, Twitter, livejournal), academic communities (e.g., DBLP, UniProt), as well as esoteric exports (e.g., Linked Open Numbers, Poképédia). This burgeoning web of structured data has succinctly been dubbed the "Web of Data".

Considering the merge of these structured exports, at a conservative estimate there now exists somewhere in the order of thirty billion RDF triples published on

Email addresses: `aidan.hogan@deri.org` (Aidan Hogan), `antoine.zimmermann@insa-lyon.fr` (Antoine Zimmermann), `juergen.umbrich@deri.org` (Jürgen Umbrich), `axel.polleres@siemens.com` (Axel Polleres), `stefan.decker@deri.org` (Stefan Decker).

the Web as Linked Data.[1] However, in this respect, size isn't everything [73]. In particular, although the situation is improving, individual datasets are still not well-interlinked (cf. [72])—without sufficient linkage, the ideal of a "Web of Data" quickly disintegrates into the current reality of "Archipelagos of Datasets".

There have been numerous works that have looked at bridging the archipelagos. Some works aim at aligning a small number of related datasets (e.g., [48,58,49]), thus focusing more on theoretical considerations than scalability, usually combining symbolic (e.g., reasoning with consistency checking) methods and similarity measures. Some authors have looked at inter-linkage of domain specific RDF datasets at various degrees of scale (e.g., [61,59,38,54,46]). Further research has also looked at exploiting shared terminological data—as well as explicitly asserted links—to better integrate Linked Data collected from thousands or millions of sources (e.g., [30,50,15,35]); the work presented herein falls most closely into this category. One approach has tackled the problem from the publishers side, detailing a system for manually specifying some (possibly heuristic) criteria for creating links between two datasets [72]. We leave further detailed related work to Section 9.

In this paper, we look at methods to provide better linkage between resources, in particular focusing on finding *equivalent* entities in the data. Our notion of an *entity* is a representation of something being described by the data; e.g., a person, a place, a musician, a protein, etc. We say that two entities are equivalent if they are *coreferent*; e.g., refer to the same person, place, etc.[2] Given a collection of datasets that speak about the same referents using different identifiers, we wish to identify these coreferences and somehow merge the knowledge contribution provided by the distinct parties. We call this merge *consolidation*.

In particular, our work is inspired by the requirements of the Semantic Web Search Engine project [32], within which we aim to offer search and browsing over large, static, Linked Data corpora crawled from the Web.[3] The core operation of SWSE is to take user keyword queries as input, and to generate a ranked list of matching entities as results. After the core components of a crawler, index and user-interface, we saw a clear need for a component that consolidates—by means of identifying and canonicalising equivalent identifiers—the indexed corpus: there was an observable lack of URIs such that coreferent blank-nodes were prevalent [30] even within the same dataset, and thus we observed many duplicate results referring to the same thing, leading to poor integration of data from our source documents.

To take a brief example, consider a simple example query: "WHO DOES TIM BERNERS-LEE KNOW?". Knowing that Tim uses the URI `timblfoaf:i` to refer to himself in his personal FOAF profile document, and again knowing that the property `foaf:knows` relates people to their (reciprocated) acquaintances, we can formulate this request as the SPARQL query [53] as follows:

```
SELECT ?person
WHERE {
  timblfoaf:i foaf:knows ?person .
}
```

However, other publishers use different URIs to identify Tim, where to get more complete answers across these naming schemes, the SPARQL query must use disjunctive `UNION` clauses for each known URI; here we give an example using a *sample* of identifiers extracted from a real Linked Data corpus (introduced later):

```
SELECT ?person
WHERE {
  { timblfoaf:i foaf:knows ?person . }
  UNION { identicau:45563 foaf:knows ?person . }
  UNION { dbpedia:Berners-Lee foaf:knows ?person . }
  UNION { dbpedia:Dr._Tim_Berners-Lee foaf:knows ?person . }
  UNION { dbpedia:Tim-Berners_Lee foaf:knows ?person . }
  UNION { dbpedia:TimBL foaf:knows ?person . }
  UNION { dbpedia:Tim_Berners-Lee foaf:knows ?person . }
  UNION { dbpedia:Tim_berners-lee foaf:knows ?person . }
  UNION { dbpedia:Timbl foaf:knows ?person . }
  UNION { dbpedia:Timothy_Berners-Lee foaf:knows ?person . }
  UNION { yagor:Tim_Berners-Lee foaf:knows ?person . }
  UNION { fb:en.tim_berners-lee foaf:knows ?person . }
  UNION { swid:Tim_Berners-Lee foaf:knows ?person . }
  UNION { dblpperson:100007 foaf:knows ?person . }
  UNION { avtimbl:me foaf:knows ?person . }
  UNION { bmpersons:Tim+Berners-Lee foaf:knows ?person . }
  ...
}
```

We see disparate URIs not only across data publishers, but also within the same namespace. Clearly, the expanded query quickly becomes extremely cumbersome.

In this paper, we look at bespoke methods for identifying and processing coreference in a manner such that the resultant corpus can be consumed as if more complete agreement on URIs was present; in other words, using standard query-answering techniques, we want the enhanced corpus to return the same answers for the original simple query as for the latter expanded query.

Our core requirements for the consolidation component are as follows:

– the component *must* give **high precision** of consolidated results;

[1] http://www4.wiwiss.fu-berlin.de/lodcloud/state/

[2] Herein, we avoid philosophical discussion on the notion of identity; for interesting discussion thereon, see [26].

[3] By static, we mean that the system does not cater for updates; this omission allows for optimisations throughout the system. Instead, we aim at a cyclical indexing paradigm, where new indexes are bulk-loaded in the background on separate machines.

– the underlying algorithm(s) *must* be **scalable**;
– the approach *must* be **fully automatic**;
– the methods *must* be **domain agnostic**;

where a component with poor precision will lead to garbled final results merging unrelated entities, where scalability is required to apply the process over our corpora typically in the order of a billion statements (and which we feasibly hope to expand in future), where the scale of the corpora under analysis precludes any manual intervention, and where—for the purposes of research—the methods should not give preferential treatment to any domain or vocabulary of data (other than core RDF(S)/OWL terms). Alongside these *primary requirements*, we also identify the following *secondary criteria*:

– the analysis *should* demonstrate **high recall**;
– the underlying algorithm(s) *should* be **efficient**;

where the consolidation component should identify as many (correct) equivalences as possible, and where the algorithm should be applicable in reasonable time. Clearly the secondary requirements are also important, but they are superceded by those given earlier, where a certain trade-off exists: we prefer a system that gives a high percentage of correct results and leads to a clean consolidated corpus over an approach that gives a higher percentage of consolidated results but leads to a partially garbled corpus; similarly, we prefer a system that can handle more data (is more scalable), but may possibly have a lower throughput (is less efficient). [4]

Thus, herein we revisit methods for scalable, precise, automatic and domain-agnostic entity consolidation over large, static Linked Data corpora. In order to make our methods *scalable*, we avoid dynamic on-disk index structures and instead opt for algorithms that rely on sequential on-disk reads/writes of compressed flat files, using operations such as scans, external sorts, merge-joins, and only light-weight or non-critical in-memory indices. In order to make our methods *efficient*, we demonstrate distributed implementations of our methods over a cluster of shared-nothing commodity hardware, where our algorithms attempt to maximise the portion of time spent in embarrassingly parallel execution—i.e., parallel, independent computation without need for inter-machine coordination. In order to make our methods *domain-agnostic* and *fully-automatic*, we exploit the generic formal semantics of the data described in RDF(S)/OWL and also, generic statistics derivable from the corpus. In order to achieve *high recall*, we incorporate additional OWL features to find novel coreference through reasoning. Aiming at *high precision*, we introduce methods that again exploit the semantics and also the statistics of the data, but to conversely disambiguate entities: to defeat equivalences found in the previous step that are unlikely to be true according to some criteria.

As such, extending upon various reasoning and consolidation techniques described in previous works [30,31,33,34], we now give a self-contained treatment of our results in this area. In addition, we distribute the execution of all methods over a cluster of commodity hardware, we provide novel, detailed performance and quality evaluation over a large, real-world Linked Data corpus of one billion statements, and we also examine exploratory techniques for interlinking similar entities based on statistical analyses, as well as methods for disambiguating and repairing incorrect coreferences.

In summary, in this paper we:

– provide some necessary preliminaries and describe our distributed architecture (Section 2);
– characterise the 1 billion quadruple Linked Data corpus that will be used for later evaluation of our methods, particular focusing on the (re-)use of data-level identifiers in the corpus (Section 3);
– describe and evaluate our distributed base-line approach for consolidation, which leverages explicit `owl:sameAs` relations (Section 4);
– describe and evaluate a distributed approach that extends consolidation to consider a richer OWL semantics for consolidating (Section 5);
– present a distributed algorithm for determining weighted concurrence between entities using statistical analysis of predicates in the corpus (Section 6);
– present a distributed approach to disambiguate entities—i.e., detect likely erroneous consolidation—combining the semantics and statistics derivable from the corpus (Section 7);
– provide critical discussion (Section 8), render related work (Section 9) and conclude with discussion (Section 10).

2. Preliminaries

In this section, we provide some necessary preliminaries relating to (i) *RDF*: the structured format used in our corpora (Section 2.1); (ii) *RDFS/OWL*, which provides formal semantics to RDF data, including the se-

[4] Of course, entity consolidation has many practical applications outside of our referential use-case SWSE, and is useful in many generic query-answering secnarios—we see these requirements as being somewhat fundamental to a consolidation component, to varying degrees.

mantics of equality (Section 2.2); (iii) *rules*, which are used to interpret the semantics of RDF and apply reasoning (Sections 2.3, 2.4); (iv) *authoritative reasoning*, which critically examines the source of certain types of Linked Data to help ensure robustness of reasoning (Section 2.5); (v) and *OWL 2 RL/RDF rules*: a standard rule-based profile for supporting OWL semantics, a subset of which we use to deduce equality (Section 2.6). Throughout, we attempt to preserve notation and terminology as prevalent in the literature.

2.1. *RDF*

We briefly give some necessary notation relating to RDF constants and RDF triples; see [28].

RDF Constant Given the set of URI references $\mathbf{U}$, the set of blank nodes $\mathbf{B}$, and the set of literals $\mathbf{L}$, the set of *RDF constants* is denoted by $\mathbf{C} = \mathbf{U} \cup \mathbf{B} \cup \mathbf{L}$. As opposed to the RDF-mandated existential semantics for blank-nodes, we interpret blank-nodes as ground Skolem constants; note also that we rewrite blank-node labels to ensure uniqueness per document as per RDF merging [28].

RDF Triple A triple $t := (s, p, o) \in \mathbf{G} := (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times \mathbf{C}$ is called an *RDF triple*, where *s* is called subject, *p* predicate, and *o* object. We call a finite set of triples $G \subset \mathbf{G}$ a *graph*. This notion of a triple restricts the positions in which certain terms may appear. We use the phrase *generalised triple* where such restrictions are relaxed, and where literals and blank-nodes are allowed to appear in the subject/predicate positions.

RDF Triple in Context/Quadruple Given $c \in \mathbf{U}$, let $\texttt{http}(c)$ denote the possibly empty graph G_c given by retrieving the URI c through HTTP (directly returning `200 Okay`). An ordered pair (t, c) with an RDF triple t = (s, p, o), $c \in \mathbf{U}$ and $t \in \texttt{http}(c)$ is called a *triple in context* c. We may also refer to (s, p, o, c) as an *RDF quadruple* or quad q with context c.

2.2. *RDFS, OWL and* `owl:sameAs`

Conceptually, RDF data is composed of assertional data (i.e., instance data) and terminological data (i.e., schema data). *Assertional data* define relationships between *individuals*, provide literal-valued attributes of those individuals, and declare individuals as members of classes. Thereafter, *terminological data* describe those classes, relationships (object properties), and attributes (datatype properties) and declaratively assigns them a *semantics*.[5] With well-defined descriptions of classes and properties, *reasoning* can then allow for deriving new knowledge, including over assertional data.

On the Semantic Web, the RDFS [11] and OWL [42] standards are prominently used for making statements with well-defined meaning and enabling such reasoning. Although primarily concerned with describing terminological knowledge—such as subsumption or equivalence between classes and properties, etc.—RDFS and OWL also contain features that operate on an assertional level. The most interesting such feature for our purposes is `owl:sameAs`: a core OWL property that allows for defining equivalences between individuals (as well as classes and properties). Two individuals related by means of `owl:sameAs` are interpreted as referring to the same entity; i.e., they are *coreferent*.

Interestingly, OWL also contains other features (on a terminological level) that allow for inferring new `owl:sameAs` relations. Such features include:

- inverse-functional properties, which act as "key properties" for uniquely identifying subjects; e.g., an `:isbn` datatype-property whose values uniquely identifies books and other documents, or the object-property `:biologicalMotherOf`, which uniquely identifies the biological mother of a particular child;
- functional properties, which work in the inverse direction and uniquely identify objects; e.g., the property `:hasBiologicalMother`;
- cardinality and max-cardinality restrictions, which, when given a value of `1`, uniquely identify subjects of a given class; e.g., the value for the property `:spouse` uniquely identifies a member of the class `:Monogamist`.

Thus, we see that RDFS and OWL contain various features that allow for inferring and supporting coreference between individuals. In order to support the semantics of such features, we apply *inference rules*, introduced next.

2.3. *Inference Rules*

Herein, we formalise the notion of an *inference rule* as applicable over RDF graphs [33]. We begin by defining the preliminary concept of a *triple pattern*, of which rules are composed, and which may contain variables in any position.

[5] Terminological and assertional data are not by any means necessarily disjoint. Furthermore, this distinction is not always required, but is useful for our purposes herein.

Triple Pattern, Basic Graph Pattern A *triple pattern* is a generalised triple where variables from the set $\mathbf{V}$ are allowed; i.e.: $t^v := (s^v, p^v, o^v) \in \mathbf{G^V}$, $\mathbf{G^V} := (\mathbf{C} \cup \mathbf{V}) \times (\mathbf{C} \cup \mathbf{V}) \times (\mathbf{C} \cup \mathbf{V})$. We call a set (to be read as conjunction) of triple patterns $\mathcal{G}^\mathbf{V} \subset \mathbf{G^V}$ a *basic graph pattern*. We denote the set of variables in graph pattern $\mathcal{G}^\mathbf{V}$ by $\mathbf{V}(\mathcal{G}^\mathbf{V})$.

Intuitively, variables appearing in triple patterns can be bound by any RDF term in $\mathbf{C}$. Such a mapping from variables to constants is called a variable binding.

Variable Bindings Let Ω be the set of *variable binding* $\mathbf{V} \cup \mathbf{C} \to \mathbf{V} \cup \mathbf{C}$ that map every constant $c \in \mathbf{C}$ to itself and every variable $v \in \mathbf{V}$ to an element of the set $\mathbf{C} \cup \mathbf{V}$. A triple t is a *binding* of a triple pattern $t^v := (s^v, p^v, o^v)$ iff there exists $\mu \in \Omega$, such that $t = \mu(t^v) = (\mu(s^v), \mu(p^v), \mu(o^v))$. A graph $\mathcal{G}$ is a binding of a graph pattern $\mathcal{G}^\mathbf{V}$ iff there exists a mapping $\mu \in \Omega$ such that $\bigcup_{t^v \in \mathcal{G}^\mathbf{V}} \mu(t^v) = \mathcal{G}$; we use the shorthand $\mu(\mathcal{G}^\mathbf{V}) = \mathcal{G}$. We use $\Omega(\mathcal{G}^\mathbf{V}, \mathcal{G}) := \{\mu \mid \mu(\mathcal{G}^\mathbf{V}) \subseteq \mathcal{G}, \mu(v) = v \text{ if } v \notin \mathbf{V}(\mathcal{G}^\mathbf{V})\}$ to denote the set of variable binding mappings for graph pattern $\mathcal{G}^\mathbf{V}$ in graph $\mathcal{G}$ that map variables outside $\mathcal{G}^\mathbf{V}$ to themselves.

We can now give the notion of an inference rule, which is comprised of a pair of basic graph patterns that form an "`IF` $\Rightarrow$ `THEN`" logical structure.

Inference Rule We define an *inference rule* (or often just *rule*) as a pair $r := (\mathcal{A}nte_r, \mathcal{C}on_r)$, where the *antecedent* (or *body*) $\mathcal{A}nte_r \subset \mathbf{G^V}$ and the *consequent* (or *head*) $\mathcal{C}on_r \subset \mathbf{G^V}$ are basic graph patterns such that all variables in the consequent appear in the antecedent: $\mathbf{V}(\mathcal{C}on_r) \subseteq \mathbf{V}(\mathcal{A}nte_r)$. We write inference rules as $\mathcal{A}nte_r \Rightarrow \mathcal{C}on_r$.

Finally, rules allow for applying *inference* through *rule applications*, where the rule body is used to generate variable bindings against a given RDF graph, and where those bindings are applied on the head to generate new triples that that graph entails.

Rule Application and Standard Closure A *rule application* is the immediate consequences $T_r(\mathcal{G}) := \bigcup_{\mu \in \Omega(\mathcal{A}nte_r, \mathcal{G})} (\mu(\mathcal{C}on_r) \setminus \mu(\mathcal{A}nte_r))$ of a rule r on a graph $\mathcal{G}$; accordingly, for a ruleset $\mathcal{R}$, $T_\mathcal{R}(\mathcal{G}) := \bigcup_{r \in \mathcal{R}} T_r(\mathcal{G})$. Now, let $\mathcal{G}_{i+1} := \mathcal{G}_i \cup T_\mathcal{R}(\mathcal{G}_i)$ and $\mathcal{G}_0 := \mathcal{G}$; the *exhaustive application* of the $T_\mathcal{R}$ operator on a graph $\mathcal{G}$ is then the least fixpoint (the smallest value for n) such that $\mathcal{G}_n = T_\mathcal{R}(\mathcal{G}_n)$. We call $\mathcal{G}_n$ the *closure* of $\mathcal{G}$ wrt. ruleset $\mathcal{R}$, denoted as $Cl_\mathcal{R}(\mathcal{G})$, or succinctly $\overline{\mathcal{G}}$ where the ruleset is obvious.

The above closure takes a graph and a ruleset and recursively applies the rules over the union of the original graph and the inferences until a fixpoint.

2.4. *T-split Rules*

In [31,33], we formalised an optimisation based on a separation of terminological knowledge from assertional data when applying rules. Similar optimisations have also been used by other authors to enable large-scale rule-based inferencing [74,71,43]. This optimisation is based on the premise that, in Linked Data (and various other scenarios), terminological data speaking about classes and properties is much smaller than assertional data speaking about individuals. Previous observations indicate that for a Linked Data corpus in the order of a billion triples, such terminological data comprises of about 0.1% of the total data volume. Additionally, terminological data is very frequently accessed during reasoning. Hence, seperating and storing the terminological data in memory allows for more efficient execution of rules, albeit at the cost of possible incompleteness [74,33].

We now reintroduce some of the concepts relating to separating terminological information [33], which we will use later when applying rule-based reasoning to support the semantics of OWL equality. We begin by formalising some related concepts that help define our notion of terminological information.

Meta-class We consider a *meta-class* as a class whose members are themselves (always) classes or properties. Herein, we restrict our notion of meta-classes to the set defined in RDF(S) and OWL specifications, where examples include `rdf:Property`, `rdfs:Class`, `owl:DatatypeProperty`, `owl:FunctionalProperty`, etc. Note that, e.g., `owl:Thing` and `rdfs:Resource` are not meta-classes since not all of their members are classes or properties.

Meta-property A *meta-property* is one that has a meta-class as its domain; again, we restrict our notion of meta-properties to the set defined in RDF(S) and OWL specifications, where examples include `rdfs:domain`, `rdfs:range`, `rdfs:subClassOf`, `owl:hasKey`, `owl:inverseOf`, `owl:oneOf`, `owl:onProperty`, `owl:unionOf`, etc. Note that `rdf:type`, `owl:sameAs`, `rdfs:label`,

e.g., do *not* have a meta-class as domain and so are not considered as meta-properties.

Terminological Triple We define the set of *terminological triples* $\mathbf{T} \subset \mathbf{G}$ as the union of:

(i) triples with `rdf:type` as predicate and a meta-class as object;
(ii) triples with a meta-property as predicate;
(iii) triples forming a *valid* RDF list whose head is the object of a meta-property (e.g., a list used for `owl:unionOf`, etc.).

The above concepts cover what we mean by terminological data in the context of RDFS and OWL. Next, we define the notion of triple patterns that distinguish between these different categories of data.

Terminological/Assertional Pattern We refer to a *terminological -triple/-graph pattern* as one whose instance can only be a terminological triple or, resp., a set thereof. An *assertional pattern* is any pattern that is not terminological.

Given the above notions of terminological data/patterns, we can now define the notion of a T-split inference rule that distinguishes terminological from assertional information.

T-split inference rule Given a rule $r := (\mathcal{A}nte_r, \mathcal{C}on_r)$, we define a *$\mathcal{T}$-split rule* r^τ as the triple $(\mathcal{A}nte^{\mathcal{T}}_{r^\tau}, \mathcal{A}nte^{\mathcal{G}}_{r^\tau}, \mathcal{C}on)$ where $\mathcal{A}nte^{\mathcal{T}}_{r^\tau}$ is the set of terminological patterns in $\mathcal{A}nte_r$, and $\mathcal{A}nte^{\mathcal{G}}_{r^\tau} := \mathcal{A}nte_r \setminus \mathcal{A}nte^{\mathcal{T}}_{r^\tau}$.

The body of T-split rules are divided into a set of patterns that apply over terminological knowledge, and a set of patterns that apply over assertional (i.e., any) knowledge. Such rules enable an optimisation whereby terminological patterns are pre-bound to generate a new, larger set of purely assertional rules called T-ground rules. We illustrate this with an example.

Example: Let $R_{\mathsf{prp\text{-}ifp}}$ denote the following rule

(?p, a, owl:InverseFunctionalProperty),
(?x1, ?p, ?y),
(?x2, ?p, ?y)
⇒ (?x1, owl:sameAs, ?x2)

When writing T-split rules, we denote terminological patterns in the body by underlining. Also, we use 'a' as a convenient shortcut for `rdf:type`, which indicates class-membership. Now take the terminological triples:

(:isbn, a, owl:InverseFunctionalProperty)
(:mbox, a, owl:InverseFunctionalProperty)

From these two triples, we can generate two T-ground rules as follows:

(?x1, :isbn, ?y),
(?x2, :isbn, ?y)
⇒ (?x1, owl:sameAs, ?x2)

(?x1, :mbox, ?y),
(?x2, :mbox, ?y)
⇒ (?x1, owl:sameAs, ?x2)

These T-ground rules encode the terminological OWL knowledge given by the previous two triples, and can be applied directly over the assertional data, e.g., describing specific books with ISBN numbers. ◇

2.5. *Authoritative T-split Rules*

Caution must be exercised when applying reasoning over arbitrary data collected from the Web; e.g., Linked Data. In previous works, we have encountered various problems when naïvely performing rule-based reasoning over arbitrary Linked Data [31,33]. In particular, third-party claims made about popular classes and properties must be critically analysed to ensure their trustworthiness. As a single example of the type of claims that can cause issues with reasoning, we found one document that defines nine *properties* as the domain of `rdf:type`;[6] thereafter, the semantics of `rdfs:domain` mandate that everything which is a member of *any* class (i.e., almost every known resource) can be inferred to be a "member" of each of the nine properties. We found that naïve reasoning lead to ∼200× more inferences than would be expected when considering the definitions of classes and properties as defined in their "namespace documents". Thus, in the general case, performing rule-based materialisation with respect to OWL semantics over arbitrary Linked Data *requires* some critical analysis of the source of data, as per our authoritative reasoning algorithm. Please see [9] for a detailed analysis of the explosion in inferences that occurs when authoritative analysis is not applied.

Such observations prompted us to investigate more robust forms of reasoning. Along these lines, when reasoning over Linked Data we apply an algorithm called *authoritative reasoning* [14,31,33], which critically ex-

[6] viz. `http://www.eiao.net/rdf/1.0`

amines the source of terminological triples and conservatively rejects those that cannot be definitively trusted: i.e., those that redefine classes and/or properties outside of the namespace document.

Dereferencing and Authority We first give the function $\mathsf{http} : \mathbf{U} \to 2^{\mathbf{G}}$ that maps a URI to an RDF graph it returns upon a HTTP lookup that returns the response code `200 Okay`; in the case of failure, the function maps to an empty RDF graph. Next, we define the function $\mathsf{redir} : \mathbf{U} \to \mathbf{U}$ that follows a (finite) sequence of HTTP redirects given upon lookup of a URI, or that returns the URI itself in the absence of a redirect or in the case of failure; this function first strips the fragment identifier (given after the '#' character) from the original URI if present. Then we give the *dereferencing function* $\mathsf{deref} : \mathsf{http} \circ \mathsf{redir}$ as the mapping from a URI (a Web location) to an RDF graph it may provide by means of a given HTTP lookup that follows redirects. Finally, we can define the *authority function*—that maps the URI of a Web document to the set of RDF terms it is authoritative for—as follows:

$$\begin{aligned}\mathsf{auth} : \mathbf{U} &\to 2^{\mathbf{C}} \\ u &\mapsto \{c \in \mathbf{U} \mid \mathsf{redir}(c) = u\} \cup \mathbf{B}(\mathsf{http}(s))\end{aligned}$$

where $\mathbf{B}(G)$ denotes the set of blank-nodes appearing in a graph G. In other words, a Web document is authoritative for URIs that dereference to it and the blank nodes it contains.

We note that making RDF vocabulary URIs dereferenceable is encouraged in various best-practices [45,4]. To enforce authoritative reasoning, when applying rules with terminological *and* assertional patterns in the body we require that terminological triples are served by a document authoritative for terms that are bound by specific variable positions in the rule.

Authoritative T-split Rule Application Given a T-split rule and a rule application involving a mapping μ as before, for the rule application to be authoritative there must additionally exist a $\mu(v)$ such that $v \in \mathbf{V}(\mathcal{A}nte^{\mathcal{T}}) \cap \mathbf{V}(\mathcal{A}nte^{\mathcal{G}})$, $\mu(v) \in \mathsf{auth}(u)$, $\mu(\mathcal{A}nte^{\mathcal{T}}) \subseteq \mathsf{http}(u)$. We call the set of variables that appear in both the terminological and assertional segment of the rule body (i.e., $\mathbf{V}(\mathcal{A}nte^{\mathcal{T}}) \cap \mathbf{V}(\mathcal{A}nte^{\mathcal{G}})$) the set of authoritative variables for the rule. Where a rule does not have terminological or assertional patterns, we consider any rule-application as authoritative. The notation of an authoritative T-ground rule follows likewise.

Example: Let $R_{\mathsf{prp\text{-}ifp}}$ denote the same rule as used in the previous example, and let the following two triples be given by the dereferenced document of `v1:isbn`, but not `v2:mbox`—i.e., a document authoritative for the former term but not the latter.

```
(v1:isbn, a, owl:InverseFunctionalProperty)
(v2:mbox, a, owl:InverseFunctionalProperty)
```

The only authoritative variable appearing in both the terminological and assertional patterns of the body of $R_{\mathsf{prp\text{-}ifp}}$ is `?p`, bound by `v1:isbn` and `v2:mbox` above. Since the document serving the two triples is authoritative for `v1:isbn`, the first triple is considered authoritative, and the following T-ground rule will be generated:

```
(?x1, v1:isbn, ?y),
(?x2, v1:isbn, ?y)
 ⇒ (?x1, owl:sameAs, ?x2)
```

However, since the document is not authoritative for `v2:mbox`, the second T-ground rule will be considered non-authoritative and discarded:

```
(?x1, v2:mbox, ?y),
(?x2, v2:mbox, ?y)
 ⇒ (?x1, owl:sameAs, ?x2)
```

Where third-party documents re-define remote terms in a way that affects inferencing over instance data, we filter such non-authoritative definitions. ◇

2.6. *OWL 2 RL/RDF Rules*

Inference rules can be used to (partially) support the semantics of RDFS and OWL. Along these lines, the OWL 2 RL/RDF [24] ruleset is a partial-axiomatisation of the OWL 2 RDF-Based Semantics, which is applicable for arbitrary RDF graphs and constitutes an extension of the RDF Semantics [28]; in other words, the ruleset supports a standardised profile of reasoning that partially covers the complete OWL semantics. Interestingly for our scenario, this profile includes inference rules that support the semantics and inference of `owl:sameAs` as discussed.

First, in Table 1, we provide the set of OWL 2 RL/RDF rules that support the (positive) semantics of `owl:sameAs`, axiomatising the symmetry (rule **eq-sym**) and transitivity (rule **eq-trans**) of the relation. To take an example, rule **eq-sym** is as follows:

(?x, owl:sameAs, ?y)
⇒ (?y, owl:sameAs, ?x)

where if we know that (:a, owl:sameAs, :b), the rule will infer the symmetric relation (:b, owl:sameAs, :a). Rule **eq-trans** operates analogously; both together allow for computing the transitive, symmetric closure of the equivalence relation. Furthermore, Table 1 also contains the OWL 2 RL/RDF rules that support the semantics of replacement (rules **eq-rep-***), whereby data that holds for one entity must also hold for equivalent entities.

Note that we (optionally, and in the case of later evaluation) choose not to support:

(i) the reflexive semantics of owl:sameAs, since reflexive owl:sameAs statements will not lead to any consolidation or other non-reflexive equality relations and will produce a large bulk of materialisations;

(ii) equality on predicates or values for rdf:type, where we do not want possibly imprecise owl:sameAs data to affect terms in these positions.

OWL2RL	Antecedent	Consequent
	assertional	
eq-sym	*?x* owl:sameAs *?y* .	*?y* owl:sameAs *?x* .
eq-trans	*?x* owl:sameAs *?y* . *?y* owl:sameAs *?z* .	*?x* owl:sameAs *?z* .
eq-rep-s	*?s* owl:sameAs *?s′* . *?s ?p ?o* .	*?s′ ?p ?o* .
eq-rep-o	*?o* owl:sameAs *?o′* . *?s ?p ?o* .	*?s ?p ?o′* .

Table 1
Rules that support the positive semantics of owl:sameAs

Given that the semantics of equality is quadratic with respect to the A-Box, we apply a partial-materialisation approach that gives our notion of *consolidation*: instead of materialising (i.e., explicitly writing down) all possible inferences given by the semantics of replacement, we instead choose one canonical identifier to represent the set of equivalent terms, thus effectively compressing the data. We have used this approach in previous works [30–32], and it has also appeared in related works in the literature [70,6], as a common-sense optimisation for handling data-level equality. To take an example, in later evaluation (cf. Table 10) we find a valid equivalence class (set of equivalent entities) with 33,052 members; materialising all pairwise non-reflexive owl:sameAs statements would infer more than 1 billion owl:sameAs relations ($33{,}052^2$ - 33,052) = 1,092,434,704; further assuming that each entity appeared in on average, e.g., two quadruples, we would infer an additional ∼2 billion of massively duplicated data. By choosing a single canonical identifier, we would instead only materialise ∼100 thousand statements.

Note that although we only perform partial materialisation, we do not change the semantics of equality: our methods are sound with respect to OWL semantics. In addition, alongside the partially materialised data, we provide a set of consolidated owl:sameAs relations (containing all of the identifiers in each equivalence class) that can be used to "backward-chain" the full inferences possible through replacement (as required). Thus, we do not consider the canonical identifier as somehow 'definitive' or superceding the other identifiers, but merely consider it as *representing* the equivalence class. [7]

Finally, herein we do not consider consolidation of literals; one may consider useful applications, e.g., for canonicalising datatype literals, but such discussion is out of the current scope.

As previously mentioned, OWL 2 RL/RDF also contains rules that use terminological knowledge (alongside assertional knowledge) to directly infer owl:sameAs relations. We enumerate these rules in Table 2; note that we italicise the labels of rules supporting features new to OWL 2, and that we list authoritative variables in bold. [8]

Applying only these OWL 2 RL/RDF rules may miss inference of some owl:sameAs statements. For example, consider the example RDF data given in Turtle syntax [3] as follows:

```
# From the FOAF Vocabulary:
 foaf:homepage rdfs:subPropertyOf foaf:isPrimaryTopicOf .
 foaf:isPrimaryTopicOf owl:inverseOf foaf:primaryTopic .
 foaf:isPrimaryTopicOf a owl:InverseFunctionalProperty .

# From Example Document A:
 exA:axel foaf:homepage <http://polleres.net/> .

# From Example Document B:
 <http://polleres.net/> foaf:primaryTopic exB:apolleres .

# Inferred through prp-spo1:
 exA:axel foaf:isPrimaryTopicOf <http://polleres.net/> .

# Inferred through prp-inv:
 exB:apolleres foaf:isPrimaryTopicOf <http://polleres.net/> .

# Subsequently, inferred through prp-ifp:
 exA:axel owl:sameAs exB:apolleres .
 exB:apolleres owl:sameAs exA:axel .
```

The example uses properties from the prominent FOAF vocabulary, which is used for publishing personal profiles as RDF. Here, we additionally need OWL 2 RL/RDF rules **prp-inv** and **prp-spo1**—handling standard owl:inverseOf and rdfs:subPropertyOf inferencing respectively—to infer the owl:sameAs relation entailed by the data.

[7] We may optionally consider non-canonical blank-node identifiers as redundant and discard them.

[8] As discussed later, our current implementation requires at least one variable to appear in all assertional patterns in the body, and so we do not support **prp-key** and **cls-maxqc3**; however, these two features are not commonly used in Linked Data vocabularies [29].

OWL2RL	Antecedent		Consequent
	terminological	*assertional*	
prp-fp	***?p*** a owl:FunctionalProperty .	*?x ?p* $?y_1$, $?y_2$.	$?y_1$ owl:sameAs $?y_2$.
prp-ifp	***?p*** a owl:InverseFunctionalProperty .	$?x_1$ *?p ?y* . $?x_2$ *?p ?y* .	$?x_1$ owl:sameAs $?x_2$.
cls-maxc2	***?x*** owl:maxCardinality 1 . ***?x*** owl:onProperty ***?p*** .	*?u* a *?x* . *?u ?p* $?y_1$, $?y_2$.	$?y_1$ owl:sameAs $?y_2$.
cls-maxqc4	***?x*** owl:maxQualifiedCardinality 1 . ***?x*** owl:onProperty ***?p*** . ***?x*** owl:onClass owl:Thing .	*?u* a *?x* . *?u ?p* $?y_1$, $?y_2$.	$?y_1$ owl:sameAs $?y_2$

Table 2
OWL 2 RL/RDF rules that directly produce `owl:sameAs` relations; we denote authoritative variables with bold and italicise the labels of rules requiring new OWL 2 constructs

Along these lines, we support an extended set of OWL 2 RL/RDF rules that contain precisely one assertional pattern and for which we have demonstrated a scalable implementation called SAOR, designed for Linked Data reasoning [33]. These rules are listed in Table 3, and as per the previous example, can generate additional inferences that indirectly lead to the derivation of new `owl:sameAs` data. We will use these rules for entity consolidation in Section 5.

OWL2RL	Antecedent		Consequent
	terminological	*assertional*	
prp-dom	***?p*** rdfs:domain *?c* .	*?x ?p ?y* .	*?x* a *?c* .
prp-rng	***?p*** rdfs:range *?c* .	*?x ?p ?y* .	*?y* a *?c* .
prp-symp	***?p*** a owl:SymmetricProperty .	*?x ?p ?y* .	*?y ?p ?x* .
prp-spo1	***?p***$_1$ rdfs:subPropertyOf $?p_2$.	*?x* $?p_1$ *?y* .	*?x* $?p_2$ *?y* .
prp-eqp1	***?p***$_1$ owl:equivalentProperty $?p_2$.	*?x* $?p_1$ *?y* .	*?x* $?p_2$ *?y* .
prp-eqp2	$?p_1$ owl:equivalentProperty ***?p***$_2$.	*?x* $?p_2$ *?y* .	*?x* $?p_1$ *?y* .
prp-inv1	***?p***$_1$ owl:inverseOf $?p_2$.	*?x* $?p_1$ *?y* .	*?y* $?p_2$ *?x* .
prp-inv2	$?p_1$ owl:inverseOf ***?p***$_2$.	*?x* $?p_2$ *?y* .	*?y* $?p_1$ *?x* .
cls-int2	***?c*** owl:intersectionOf ($?c_1$ ... $?c_n$) .	*?x* a *?c* .	*?x* a $?c_1$...$?c_n$.
cls-uni	*?c* owl:unionOf ($?c_1$...***?c***$_i$...$?c_n$) .	*?x* a $?c_i$	*?x* a *?c* .
cls-svf2	*?x* owl:someValuesFrom owl:Thing ; owl:onProperty ***?p*** .	*?u ?p ?v* .	*?u* a *?x* .
cls-hv1	***?x*** owl:hasValue *?y* ; owl:onProperty *?p* .	*?u* a *?x* .	*?u ?p ?y* .
cls-hv2	*?x* owl:hasValue ***?y*** ; owl:onProperty ***?p*** .	*?u ?p ?y* .	*?u* a *?x* .
cax-sco	***?c***$_1$ rdfs:subClassOf $?c_2$.	*?x* a $?c_1$.	*?x* a $?c_2$.
cax-eqc1	***?c***$_1$ owl:equivalentClass $?c_2$.	*?x* a $?c_1$.	*?x* a $?c_2$.
cax-eqc2	$?c_1$ owl:equivalentClass ***?c***$_2$.	*?x* a $?c_2$.	*?x* a $?c_1$.

Table 3
OWL 2 RL/RDF rules containing precisely one assertional pattern in the body, with authoritative variables in bold (cf. [33])

Lastly, as we discuss later (particularly in Section 7) Linked Data is inherently noisy and prone to mistakes. Although our methods are sound (i.e., correct) with respect to formal OWL semantics, such noise in our input data may lead to unintended consequences when applying reasoning which we wish to minimise and/or repair. Relatedly, OWL contains features that allow for detecting formal contradictions—called *inconsistencies*—in RDF data. An example of an inconsistency would be where something is a member of two disjoint classes, such as `foaf:Person` and `foaf:Organization`; formally, the intersection of such disjoint classes should be empty and they should not share members. Along these lines, OWL 2 RL/RDF contains rules (with the special symbol `false` in the head) that allow for detecting inconsistency in an RDF graph. We use a subset of these consistency-checking OWL 2 RL/RDF rules later in Section 7 to try to automatically detect and repair unintended `owl:sameAs` inferences; the supported subset is listed in Table 4.

OWL2RL	Antecedent	
	terminological	*assertional*
eq-diff1	-	*?x* owl:sameAs *?y* . *?x* owl:differentFrom *?y* .
prp-irp	***?p*** a owl:IrreflexiveProperty .	*?x ?p ?x* .
prp-asyp	***?p*** a owl:AsymmetricProperty	*?x ?p ?y* . *?y ?p ?x* .
prp-pdw	***?p***$_1$ owl:propertyDisjointWith ***?p***$_2$.	*?x* $?p_1$ *?y* ; $?p_2$ *?y* .
prp-adp	*?x* a owl:AllDisjointProperties . *?x* owl:members (***?p***$_1$, ..., ***?p***$_n$) .	*?u* $?p_i$ *?y* ; $?p_j$ *?y* . ($i \neq j$)
cls-com	***?c***$_1$ owl:complementOf ***?c***$_2$.	*?x* a $?c_1$, $?c_2$.
cls-maxc1	***?x*** owl:maxCardinality 0 . ***?x*** owl:onProperty ***?p*** .	*?u* a *?x* ; *?p ?y* .
cls-maxqc2	***?x*** owl:maxQualifiedCardinality 0 . ***?x*** owl:onProperty ***?p*** . ***?x*** owl:onClass owl:Thing .	*?u* a *?x* ; *?p ?y* .
cax-dw	***?c***$_1$ owl:disjointWith ***?c***$_2$.	*?x* a $?c_1$, $?c_2$.
cax-adc	*?x* a owl:AllDisjointClasses . *?x* owl:members (***?c***$_1$, ..., ***?c***$_n$) .	*?z* a $?c_i$, $?c_j$. ($i \neq j$)

Table 4
OWL 2 RL/RDF rules used to detect inconsistency that we currently support; we denote authoritative variables with bold

2.7. *Distribution architecture*

To help meet our scalability requirements, our methods are implemented on a shared-nothing distributed architecture [64] over a cluster of commodity hardware. The distributed framework consists of a master machine that orchestrates the given tasks, and several slave machines that perform parts of the task in parallel.

The master machine can instigate the following distributed operations:

– **scatter:** partition on-disk data using some local *split* function, and send each chunk to individual slave machines for subsequent processing;

– **run:** request the parallel execution of a task by the slave machines—such a task either involves processing of some data local to the slave machine, or the **coordinate** method (described later) for reorganising the data under analysis;
– **gather:** gathers chunks of output data from the slave swarm and performs some local *merge* function over the data;
– **flood:** broadcast global knowledge required by all slave machines for a future task.

The master machine provides input data to the slave swarm, provides the control logic required by the distributed task (commencing tasks, coordinating timing, ending tasks), gathers and locally perform tasks on global knowledge that the slave machines would otherwise have to replicate in parallel, and transmits globally required knowledge.

The slave machines, as well as performing tasks in parallel, can perform the following distributed operation (at the behest of the master machine):
– **coordinate:** local data on each slave machine is partitioned according to some *split* function, with the chunks sent to individual machines in parallel; each slave machine also gathers the incoming chunks in parallel using some *merge* function.

The above operation allows slave machines to reorganise (split/send/gather) intermediary amongst themselves; the **coordinate** operation could be replaced by a pair of **gather/scatter** operations performed by the master machine, but we wish to avoid the channelling of all intermediary data through one machine.

Note that herein, we assume that the input corpus is evenly distributed and split across the slave machines, and that the slave machines have roughly even specifications: that is, we do not consider any special form of load balancing, but instead aim to have uniform machines processing comparable data-chunks.

We note that there is the practical issue of the master machine being idle waiting for the slaves, and, more critically, the potentially large cluster of slave machines waiting idle for the master machine. One could overcome idle times with mature task-scheduling (e.g., interleaving jobs) and load-balancing. From an algorithmic point of view, removing the central coordination on the master machine may enable better distributability. One possibility would be to allow the slave machines to duplicate the aggregation of global knowledge in parallel: although this would free up the master machine and would probably take roughly the same time, duplicating computation wastes resources that could otherwise be exploited by, e.g., interleaving jobs. A second possibility would be to avoid the requirement for global knowledge and to coordinate upon the larger corpus (e.g., a **coordinate** function hashing on the subject and object of the data, or perhaps an adaptation of the SPEEDDATE routing strategy [51]). Such decisions are heavily influenced by the scale of the task to perform, the percentage of knowledge that is globally required, how the input data are distributed, how the output data should be distributed, and the nature of the cluster over which it should be performed and the task-scheduling possible. The distributed implementation of our tasks are designed to exploit a relatively small percentage of global knowledge which is cheap to coordinate, and we choose to avoid—insofar as reasonable—duplicating computation.

2.8. *Experimental setup*

Our entire code-base is implemented on top of standard Java libraries. We instantiate the distributed architecture using Java RMI libraries, and using the lightweight open-source Java RMIIO package [9] for streaming data for the network.

All of our evaluation is based on nine machines connected by Gigabit ethernet, [10] each with uniform specifications, viz., 2.2GHz Opteron x86-64, 4GB main memory, 160GB SATA hard-disks, running Java 1.6.0_12 on Debian 5.0.4.

3. Experimental corpus

Later in this paper, we discuss the performance and results of applying our methods over a corpus of 1.118 billion quadruples derived from an open-domain RDF/XML crawl of 3.985 million web documents in mid-May 2010 (detailed in [32,29]). The crawl was conducted in a breadth-first manner, extracting URIs from all positions of the RDF data. Individual URI queues were assigned to different *pay-level-domains* (aka. *PLDs*: domains that require payment, e.g., `deri.ie`, `data.gov.uk`), where we enforced a politeness policy of accessing a maximum of two URIs per PLD per second. URIs with the highest inlink count per each PLD queue were polled first.

With regards the resulting corpus, of the 1.118 billion quads, 1.106 billion are unique, and 947 million are unique triples. The data contain 23 thousand unique predicates and 105 thousand unique class terms (terms

[9] `http://openhms.sourceforge.net/rmiio/`
[10] We observe, e.g., a max FTP transfer rate of 38MB/sec between machines.

in the object position of an rdf:type triple). In terms of diversity, the corpus consists of RDF from 783 pay-level-domains [32].

Note that further details about the parameters of the crawl and statistics about the corpus are available in [32,29].

Now we discuss the usage of terms in a *data-level position*, viz., terms in the subject position or object position of non-rdf:type triples.[11] Since we do not consider the consolidation of literals or schema-level concepts, we focus on characterising blank-node and URI re-use in such data-level positions, thus rendering a picture of the structure of the raw data.

We found 286.3 million unique terms, of which, 165.4 million (57.8%) were blank-nodes, 92.1 million (32.2%) were URIs, and 28.9 million (10%) were literals. With respect to literals, each had on average 9.473 data-level occurrences (by definition, all in the object position).

With respect to blank-nodes, each had on average 5.233 data-level occurrences. Each occurred on average 0.995 times in the object position of a non-rdf:type triple, with 3.1 million (1.87%) not occurring in the object position; conversely, each occurred on average 4.239 times in the subject position of a triple, with 69 thousand (0.04%) not occurring in the subject position. Thus, we summarise that almost all blank-nodes appear in both the subject position and object position, but occur most prevalently in the former. Importantly, note that in our input, blank-nodes cannot be re-used across sources.

With respect to URIs, each had on average 9.41 data-level occurrences (1.8× the average for blank-nodes), with 4.399 average appearances in the subject position and 5.01 appearances in the object position—19.85 million (21.55%) did not appear in an object position, whilst 57.91 million (62.88%) did not appear in a subject position.

With respect to re-use across sources, each URI had a data-level occurrence in, on average, 4.7 documents, and 1.008 PLDs—56.2 million (61.02%) of URIs appeared in only one document, and 91.3 million (99.13%) only appeared in one PLD. Also, re-use of URIs across documents was heavily weighted in favour of use in the object position: URIs appeared in the subject position in, on average, 1.061 documents and 0.346 PLDs; for the object position of non-rdf:type triples, URIs occurred in, on average, 3.996 documents and 0.727 PLDs.

The URI with the most data-level occurrences (1.66 million) was http://identi.ca/; the URI with the most re-use across documents (appearing in 179.3 thousand documents) was http://creativecommons.org/licenses/by/3.0/; the URI with the most re-use across PLDs (appearing in 80 different domains) was http://www.ldodds.com/foaf/foaf-a-matic. Although some URIs do enjoy widespread re-use across different documents and domains, in Figures 1 and 2 we give the distribution of re-use of URIs across documents and across PLDs, where a power-law relationship is roughly evident—again, the majority of URIs only appear in one document (61%) or in one PLD (99%).

From this analysis, we can conclude that with respect to data-level terms in our corpus:

– blank-nodes, which by their very nature cannot be re-used across documents, are 1.8× more prevalent than URIs;
– despite a smaller number of unique URIs, each one is used in (probably coincidentally) 1.8× more triples;
– unlike blank-nodes, URIs commonly only appear in either a subject position or an object position;
– each URI is re-used on average in 4.7 documents, but usually only within the same domain—most external re-use is in the object position of a triple;
– 99% of URIs appear in only one PLD.

We can conclude that within our corpus—itself a general crawl for RDF/XML on the Web—we find that there is only sparse re-use of data-level terms across sources, and particularly across domains.

Finally, for the purposes of demonstrating performance across varying numbers of machines, we extract a smaller corpus, comprising of 100 million quadruples, from the full evaluation corpus. We extract the sub-corpus from the head of the raw corpus; since the data are ordered by access time, polling statements from the head roughly emulates a smaller crawl of data, which should ensure, e.g., that all well-linked vocabularies are contained therein.

4. Base-line Consolidation

We now present the "base-line" algorithm for consolidation that consumes asserted owl:sameAs relations in the data. Linked Data best-practices encourage the provision of owl:sameAs links between exporters that coin different URIs for the same entities:

> *"It is common practice to use the owl:sameAs property for stating that another data source also provides information about a specific non-information resource."*

[11] Please see [32, Appendix A] for further statistics relating to this corpus.

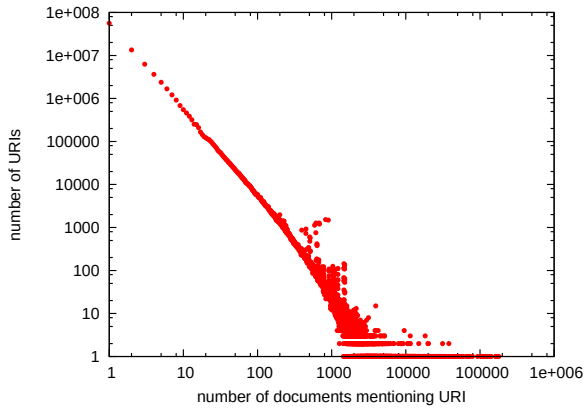


Fig. 1. Distribution of URIs and the number of documents they appear in (in a data-position)

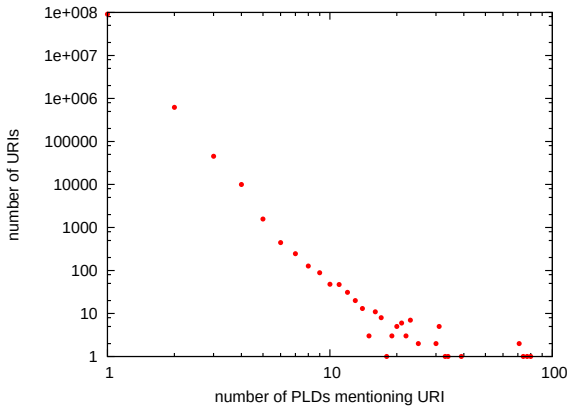


Fig. 2. Distribution of URIs and the number of PLDs they appear in (in a data-position)

—[7, § 6]

We would thus expect there to be a significant amount of explicit `owl:sameAs` data present in our corpus—provided directly by the Linked Data publishers themselves—that can be directly used for consolidation.

To perform consolidation over such data, the first step is to extract all explicit `owl:sameAs` data from the corpus. We must then compute the transitive and symmetric closure of the equivalence relation and build *equivalence classes*: sets of coreferent identifiers. Note that the set of equivalence classes forms a *partition* of coreferent identifiers where, due to the transitive and symmetric semantics of `owl:sameAs`, each identifier can only appear in one such class. Also note that we do not need to consider singleton equivalence classes that contain only one identifier: consolidation need not perform any action if an identifier is found only to be coreferent with itself. Once the closure of asserted `owl:sameAs` data has been computed, we then need to build an index that maps identifiers to the equivalence class in which it is contained. This index enables lookups of coreferent identifiers. Next, in the consolidated data, we would like to collapse the data mentioning identifiers in each equivalence class to instead use a single consistent, canonical identifier; for each equivalence class, we must thus choose a canonical identifier. Finally, we can scan the entire corpus, and using the equivalence class index, rewrite the original identifiers to their canonincal form. As an optional step, we can also add links between each canonical identifier and its coreferent forms using an artifical `owl:sameAs` relation in the output, thus persisting the original identifiers.

The distributed algorithm presented in this section is based on an in-memory equivalence class closure and index, and is the current method of consolidation employed by the SWSE system, described previously in [32]. Herein, we briefly reintroduce the approach from [32], where we also add new performance evaluation over varying numbers of machines in the distributed setup, present more discussion and analysis of the use of `owl:sameAs` in our Linked Data corpus, and manually evaluate the precision of the approach for an extended sample of 1,000 coreferent pairs. In particular, this section serves as a baseline for comparison against the extended consolidation approach that uses richer reasoning features, explored later in Section 5.

4.1. *High-level approach*

Based on the previous discussion, the approach is straight-forward:

(i) scan the corpus and separate out all asserted `owl:sameAs` relations from the main body of the corpus;
(ii) load these relations into an in-memory index that encodes the transitive and symmetric semantics of `owl:sameAs`;
(iii) for each equivalence class in the index, choose a canonical term;
(iv) scan the corpus again, canonicalising any term in the subject position or object position of an `rdf:type` triple.

Thus, we need only index a small subset of the corpus—`owl:sameAs` statements—and can apply consolidation by means of two scans.

The non-trivial aspects of the algorithm are given by the equality closure and index. To perform the in-

memory transitive, symmetric closure, we use a traditional union–find algorithm [66,40] for computing equivalence partitions, where (i) equivalent elements are stored in common sets such that each element is only contained in one set; (ii) a map provides a find function for looking up which set an element belongs to; and (iii) when new equivalent elements are found, the sets they belong to are unioned. The process is based on an in-memory map index, and is detailed in Algorithm 1, where:

(i) the eqc labels refer to *eq*uivalence *c*lasses, and s and o refer to RDF *s*ubject and *o*bject terms;
(ii) the function `map.get` refers to an identifier lookup on the in-memory index that should return the intermediary equivalence class associated with that identifier (i.e., find);
(iii) the function `map.put` associates an identifier with a new equivalence class.

The output of the algorithm is an in-memory map from identifiers to their respective equivalence class.

Next, we must choose a canonical term for each equivalence class: we prefer URIs over blank-nodes, thereafter choosing a term with the lowest alphabetical ordering; a canonical term is thus associated to each equivalent set. Once the equivalence index has been finalised, we re-scan the corpus and canonicalise the data using the in-memory index to service lookups.

Algorithm 1 Building equivalence map [32]

Require: *SAMEAS DATA*: $\mathcal{SA}$
map $\leftarrow \{\}$
for $(s, \texttt{owl:sameAs}, o) \in \mathcal{SA} \wedge s \neq o$ **do**
  $eqc_s \leftarrow$ map.get(s)
  $eqc_o \leftarrow$ map.get(o)
  if $eqc_s = \emptyset \wedge eqc_o = \emptyset$ **then**
    $eqc_{s \cup o} \leftarrow \{s, o\}$
    map.put($s, eqc_{s \cup o}$)
    map.put($o, eqc_{s \cup o}$)
  else if $eqc_s = \emptyset$ **then**
    add s to eqc_o
    map.put(s, eqc_o)
  else if $eqc_o = \emptyset$ **then**
    add o to eqc_s
    map.put(o, eqc_s)
  else if $eqc_s \neq eqc_o$ **then**
    if $|eqc_s| > |eqc_o|$ **then**
      add all eqc_o into eqc_s
      for $e_o \in eqc_o$ **do**
        map.put(e_o,eqc_s)
      end for
    else
      add all eqc_s into eqc_o
      for $e_s \in eqc_s$ **do**
        map.put(e_s,eqc_o)
      end for
    end if
  end if
end for

4.2. *Distributed approach*

Again, distribution of the approach is fairly intuitive, as follows:

(i) **run**: scan the distributed corpus (split over the slave machines) in parallel to extract `owl:sameAs` relations;
(ii) **gather**: gather all `owl:sameAs` relations onto the master machine, and build the in-memory equality index;
(iii) **flood/run**: send the equality index (in its entirety) to each slave machine, and apply the consolidation scan in parallel.

As we will see in the next section, the most expensive methods—involving the two scans of the main corpus—can be conducted in parallel.

4.3. *Performance Evaluation*

We applied the distributed base-line consolidation over our corpus with the aformentioned procedure and setup. The entire consolidation process took 63.3 min, with the bulk of time taken as follows: the first scan extracting `owl:sameAs` statements took 12.5 min, with an average idle time for the servers of 11 s (1.4%)—i.e., on average, the slave machines spent 1.4% of the time idly waiting for peers to finish. Transferring, aggregating and loading the `owl:sameAs` statements on the master machine took 8.4 min. The second scan rewriting the data according to the canonical identifiers took in total 42.3 min, with an average idle time of 64.7 s (2.5%) for each machine at the end of the round. The slower time for the second round is attributable to the extra overhead of re-writing the GZip-compressed data to disk, as opposed to just reading.

In the rightmost column of Table 5 (**full-8**), we give a breakdown of the timing for the tasks over the full corpus using eight slave machines and one master machine. Independent of the number of slaves, we note that the master machine required 8.5 min for coordinating globally-required `owl:sameAs` knowledge, and that the rest of the task time is spent in embarrassingly parallel execution (amenable to reduction by increasing the number of machines). For our setup, the slave ma-

	Category	1		2		4		8		full-8	
		min	%	min	%	min	%	min	%	min	%
	Total execution time	42.2	100.0	22.3	100.0	11.9	100.0	6.7	100.0	63.3	100.0
MASTER	**Executing**	0.5	1.1	0.5	2.2	0.5	4.0	0.5	7.5	8.5	13.4
MASTER	Aggregating `owl:sameAs`	0.4	1.1	0.5	2.2	0.5	4.0	0.5	7.5	8.4	13.3
MASTER	Miscellaneous	$\sim$	0.1	$\sim$	0.1	$\sim$	0.1	$\sim$	0.4	0.1	0.1
MASTER	**Idle (waiting for slaves)**	41.8	98.9	21.8	97.7	11.4	95.8	6.2	92.1	54.8	86.6
SLAVES	**Avg. Executing (total)**	41.8	98.9	21.8	97.7	11.4	95.8	6.2	92.1	53.5	84.6
SLAVES	Extract `owl:sameAs`	9.0	21.4	4.5	20.2	2.2	18.4	1.1	16.8	12.3	19.5
SLAVES	Consolidate	32.7	77.5	17.1	76.9	8.8	73.5	4.7	70.5	41.2	65.1
SLAVES	**Avg. Idle**	0.5	1.1	0.7	2.9	1.0	8.2	0.8	12.7	9.8	15.4
SLAVES	Waiting for peers	0.0	0.0	0.1	0.7	0.5	4.0	0.3	4.7	1.3	2.0
SLAVES	Waiting for master	0.5	1.1	0.5	2.3	0.5	4.2	0.5	7.9	8.5	13.4

Table 5
Breakdown of timing of distributed baseline consolidation

chines were kept busy for, on average, 84.6% of the total task time; of the idle time, 87% was spent waiting for the master to coordinate the `owl:sameAs` data, and 13% was spent waiting for peers to finish their task due to sub-optimal load balancing. The master machine spent 86.6% of the task idle waiting for the slaves to finish.

In addition, Table 5 also gives performance results for one master and varying numbers of slave machines (**1**,**2**,**4**,**8**) over the 100 million quadruple corpus. Note that in the table, we use the symbol $\sim$ to indicate a negligible value ($0 < \sim < 0.05$). We see that as the number of slave machines increases, so too does the *percentage* of time taken to aggregate global knowledge on the master machine (the *absolute* time stays roughly stable). This indicates that for a high number of machines, the aggregation of global knowledge will eventually become a bottleneck, and an alternative distribution strategy may be preferable, subject to further investigation. Overall, however, we see that the total execution times roughly halve when the number of slave machines are doubled: we observe a $0.528\times$, $0.536\times$ and $0.561\times$ reduction in total task execution time moving from **1** slave machine to **2**, **2** to **4**, and **4** to **8**, respectively (here, a value of $0.5\times$ would indicate linear scaling out).

4.4. *Results Evaluation*

We extracted 11.93 million raw `owl:sameAs` quadruples from the full corpus. Of these, however, there were only 3.77 million unique triples. After closure, the data formed 2.16 million equivalence classes mentioning 5.75 million terms (6.24% of URIs)—an average of 2.65 elements per equivalence class. Of the 5.75 million terms, only 4,156 were blank-nodes. Figure 3 presents the distribution of sizes of the equivalence classes, where the largest equivalence class contains 8,481 equivalent entities and 1.6 million (74.1%) equivalence classes contain the minimum two equivalent identifiers. Figure 4 shows a similar distribution, but for the number of PLDs extracted from URIs in each equivalence class. Interestingly, the majority (57.1%) of equivalence classes contained identifiers from more than one PLD, where the most diverse equivalence class contained URIs from 32 PLDs.

Table 6 shows the canonical URIs for the largest 5 equivalence classes; we manually inspected the results and show whether or not the results were verified as correct/incorrect. Indeed, results for class **1** and **2** were deemed incorrect due to over-use of `owl:sameAs` for linking drug-related entities in the DailyMed and LinkedCT exporters. Results **3** and **5** were verified as correct consolidation of prominent Semantic Web related authors, resp.: `Dieter Fensel` and `Rudi Studer`—authors are given many duplicate URIs by the RKBExplorer coreference index.[12] Result **4** contained URIs from various sites generally referring to the United States, mostly from DBPedia and LastFM. With respect to the DBPedia URIs, these (i) were equivalent but for capitilisation variations or stop-words, (ii) were variations of abbreviations or valid synonyms, (iii) were different language versions (e.g., `dbpedia:États_Unis`), (iv) were nicknames (e.g., `dbpedia:Yankee_land`), (v) were related but not equivalent (e.g., `dbpedia:American_Civilization`), (vi) were just noise (e.g., `dbpedia:LOL_Dean`).

Besides the largest equivalence classes, which we have seen are prone to errors perhaps due to the snowballing effect of the transitive and symmetric closure,

[12] For example, see the coreference results given by `http://www.rkbexplorer.com/sameAs/?uri=http://acm.rkbexplorer.com/id/person-53292-22877d02973d0d01e8f29c7113776e7e`, which at the time of writing correspond to 436 out of the 443 equivalent URIs found for `Dieter Fensel`.

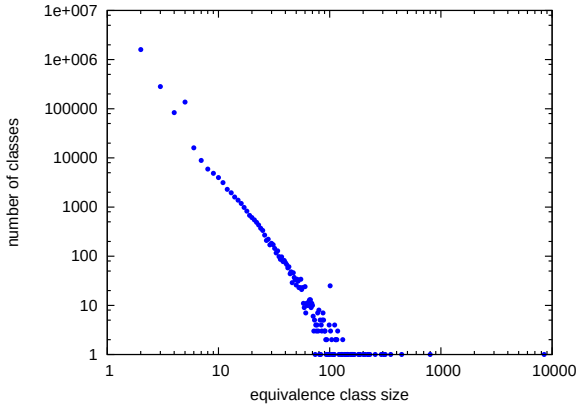


Fig. 3. Distribution of sizes of equivalence classes on log/log scale (from [32])

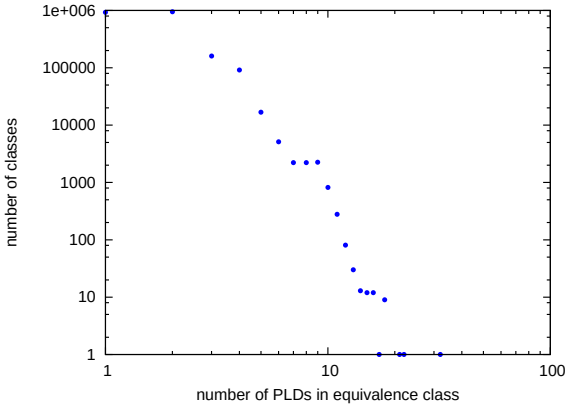


Fig. 4. Distribution of the number of PLDs per equivalence class on log/log scale

#	Canonical Term (Lexically First in Equivalence Class)	Size	Correct?
1	http://bio2rdf.org/dailymed_drugs:1000	8,481	✗
2	http://bio2rdf.org/dailymed_drugs:1042	800	✗
3	http://acm.rkbexplorer.com/id/person-53292-22877d02973d0d01e8f29c7113776e7e	443	✓
4	http://agame2teach.com/#ddb61cae0e083f705f65944cc3bb3968ce3f3ab59-ge_1	353	✓/✗
5	http://acm.rkbexplorer.com/id/person-236166-1b4ef5fdf4a5216256064c45a8923bc9	316	✓

Table 6
Largest 5 equivalence classes (from [32])

#	PLD	PLD	Co-occur
1	rdfize.com	uriburner.com	506,623
2	dbpedia.org	freebase.com	187,054
3	bio2rdf.org	purl.org	185,392
4	loc.gov	info:lc/authorities [13]	166,650
5	l3s.de	rkbexplorer.com	125,842
6	bibsonomy.org	l3s.de	125,832
7	bibsonomy.org	rkbexplorer.com	125,830
8	dbpedia.org	mpii.de	99,827
9	freebase.com	mpii.de	89,610
10	dbpedia.org	umbel.org	65,565

Table 7
Top 10 PLD pairs co-occurring in the equivalence classes, with number of equivalence classes they co-occur in

we also manually evaluated a random sample of results. For the sampling, we take the closed equivalence classes extracted from the full corpus, pick an identifier (possibly non-canonical) at random and then randomly pick another coreferent identifier from its equivalence class. We then retrieve the data associated for both identifiers and manually inspect them to see if they are, in fact, equivalent. For each pair, we then select one of four options: SAME, DIFFERENT, UNCLEAR, TRIVIALLY SAME. We used the UNCLEAR option sparingly and only where there was not enough information to make an informed decision, or for difficult subjective choices; note that where there was not enough information in our corpus, we tried to dereference more information to minimise use of UNCLEAR; in terms of difficult, subjective choices, one example pair we marked unclear was `dbpedia:Folkcore` and `dbpedia:Experimental_folk`. We used the TRIVIALLY SAME option to indicate that an equivalence is purely "syntactic", where one identifier has no information other than being the target of a `owl:sameAs` link; although the identifiers are still coreferent, we distinguish this case since the equivalence is not so "meaningful". For the 1,000 pairs, we choose option TRIVIALLY SAME 661 times (66.1%), SAME 301 times (30.1%), DIFFERENT 28 times (2.8%) and UNCLEAR 10 times (1%). In summary, our manual verification of the baseline results puts the precision at ∼97.2%, albeit with many syntatic equivalences found. Of those found to be incorrect, many were closely-related DBpedia resources that were linked to by the same resource on the Freebase exporter; an example of this was `dbpedia:Rock_Hudson` and `dbpedia:Rock_Hudson_filmography` having `owl:sameAs` links from the same Freebase concept `fb:rock_hudson`.

Moving on, in Table 7 we give the most frequently co-occurring PLD-pairs in our equivalence classes, where datasets resident on these domains are "heavily" interlinked with `owl:sameAs` relations. We italicise the indexes of inter-domain links that were observed to often be purely syntactic.

With respect to consolidation, identifiers in 78.6 mil-

lion subject positions (7% of subject positions) and 23.2 million non-rdf:type-object positions (2.6%) were rewritten, giving a total of 101.9 million positions rewritten (5.1% of total rewritable positions). The average number of documents mentioning each URI rose slightly from 4.691 to 4.719 (a 0.6% increase) due to consolidation, and the average number of PLDs also rose slightly from 1.005 to 1.007 (a 0.2% increase).

5. Extended Reasoning Consolidation

We now look at extending the baseline approach to include more expressive reasoning capabilities, in particular using OWL 2 RL/RDF rules (introduced previously in Section 2.6) to infer novel owl:sameAs relations that can then be used for consolidation alongside the explicit relations asserted by publishers.

As described in Section 2.2, OWL provides various features—including functional properties, inverse-functional properties, and certain cardinality restrictions—that allow for inferring novel owl:sameAs data. Such features could ease the burden on publishers of providing explicit owl:sameAs mappings to coreferent identifiers in remote naming schemes. For example, inverse-functional properties can be used in conjunction with legacy identification schemes—such as ISBNs for books, EAN·UCC-13 or MPN for products, MAC addresses for network-enabled devices, etc.—to bootstrap identity on the Web of Data within certain domains; such identification values can be encoded as simple datatype strings, thus bypassing the requirement for bespoke agreement or mappings between URIs. Also, "information resources" with indigenous URIs can be used for *indirectly* identifying related resources to which they have a functional mapping, where examples include personal email-addresses, personal homepages, WIKIPEDIA articles, etc.

Although Linked Data literature has not explicitly endorsed or encouraged such usage, prominent grassroots efforts publishing RDF on the Web rely (or have relied) on inverse-functional properties for maintaining consistent identity. For example, in the Friend Of A Friend (FOAF) community, a technique called smushing was proposed to leverage such properties for identity, serving as an early precursor to methods described herein. [14]

Versus the baseline consolidation approach, we must first apply some reasoning techniques to infer novel owl:sameAs relations. Once we have the inferred owl:sameAs data materialised and merged with the asserted owl:sameAs relations, we can then apply the same consolidation approach as per the baseline: (i) apply the transitive symmetric closure and generate the equivalence classes, (ii) pick canonical identifiers for each class and (iii) rewrite the corpus. However, in contrast to the baseline, we expect the extended volume of owl:sameAs data to make in-memory storage infeasible. [15] Thus, in this section, we avoid the need to store equivalence information in-memory, where we instead investigate batch-processing methods for applying an on-disk closure of the equivalence relations, and likewise on-disk methods for rewriting data.

Early versions of the local batch-processing [31] techniques and some of the distributed reasoning techniques [33] are borrowed from previous works. Herein, we reformulate and combine these techniques into a complete solution for applying enhanced consolidation in a distributed, scalable manner over large-scale Linked Data corpora. We provide new evaluation over our 1.118 billion quadruple evaluation corpus, presenting new performance results over the full corpus and for a varying number of slave machines. We also analyse, in depth, the results of applying the techniques over our evaluation corpus, analysing the applicability of our methods in such scenarios. We contrast the results given by the baseline and extended consolidation approaches, and again manually evaluate the results of the extended consolidation approach for 1,000 pairs found to be coreferent.

5.1. *High-level approach*

In Table 2, we provide the pertinent rules for inferring new owl:sameAs relations from the data. However, after analysis of the data, we observed that no documents used the owl:maxQualifiedCardinality construct required for the **cls-maxqc*** rules, and that only one document defined one owl:hasKey axiom [16] involving properties with less than 5 occurrences in the data—hence, we leave implementation of these rules for future work and note that these new OWL 2 constructs have probably not yet had time to find proper traction on the Web. Thus, on top of inferencing involving explicit owl:sameAs, we are left with rule **prp-fp**, which supports the semantics of properties typed owl:Funct-

[14] cf. http://wiki.foaf-project.org/w/Smushing; retr. 2011/01/22.

[15] Further note that since all machines must have all owl:sameAs information, adding more machines does not increase the capacity for handling more such relations: the scalability of the baseline approach is limited by the machine with the least in-memory capacity.

[16] http://huemer.lstadler.net/role/rh.rdf

ionalProperty; and rule **prp-ifp**, which supports the semantics of properties typed owl:InverseFunctional-Property; and rule **cls-maxc1**, which supports the semantics of classes with a specified cardinality of 1 for some defined property (a class restricted version of the functional-property inferencing). [17]

Thus, we look at using OWL 2 RL/RDF rules **prp-fp**, **prp-ifp** and **cls-maxc2** for inferring new owl:sameAs relations between individuals—we also support an additional rule that gives an exact cardinality version of **cls-maxc2**. [18] Herein, we refer to these rules as *consolidation rules*.

We also investigate pre-applying more general OWL 2 RL/RDF reasoning over the corpus to derive more complete results, where the ruleset is available in Table 3 and is restricted to OWL 2 RL/RDF rules with one assertional pattern [33]; unlike the consolidation rules, these *general rules* do not require the computation of assertional joins, and thus are amenable to execution by means of a single-scan of the corpus. We have demonstrated this profile of reasoning to have good competency with respect to the features of RDFS and OWL used in Linked Data [29]. These general rules may indirectly lead to the inference of additional owl:sameAs relations, particularly when combined with the consolidation rules of Table 2.

Once we have used reasoning to derive novel owl:sameAs data, we then apply an on-disk batch processing technique to close the equivalence relation and to ultimately consolidate the corpus.

Thus, our high-level approach is as follows:

(i) extract relevant terminological data from the corpus;
(ii) bind the terminological patterns in the rules from this data, thus creating a larger set of general rules with only one assertional pattern and identifying assertional patterns that are useful for consolidation;
(iii) apply general-rules over the corpus, and buffer any input/inferred statements relevant for consolidation to a new file;
(iv) derive the closure of owl:sameAs statements from the consolidation-relevant dataset;
(v) apply consolidation over the main corpus with respect to the closed owl:sameAs data.

[17] Note that we have presented non-distributed, batch-processing execution of these rules at a smaller scale (147 million quadruples) in [31].

[18] Exact cardinalities are disallowed in OWL 2 RL due to their effect on the formal proposition of completeness underlying the profile, but such considerations are moot in our scenario.

In Step (i), we extract terminological data—required for application of our rules—from the main corpus. As discussed in Section 2.5, to help ensure robustness, we apply authoritative reasoning; in other words, at this stage we discard any third-party terminological statements that affect inferencing over instance data for remote classes and properties.

In Step (ii), we then compile the authoritative terminological statements into the rules to generate a set of T-ground (purely assertional) rules, which can then be applied over the entirety of the corpus [33]. As mentioned in Section 2.6, the T-ground general rules only contain one assertional pattern, and thus are amenable to execution by a single scan; e.g., given the statement:

```
:homepage rdfs:subPropertyOf :isPrimaryTopicOf .
```

we would generate a T-ground rule:

```
?x :homepage ?y .
⇒ ?x :isPrimaryTopicOf ?y .
```

which does not require the computation of joins. We also bind the terminological patterns of the consolidation rules in Table 2; however, these rules cannot be performed by means of a single scan over the data since they contain multiple assertional patterns in the body. Thus, we instead extract triple patterns, such as:

```
?x :isPrimaryTopicOf ?y .
```

which may indicate data useful for consolidation (note that here, :isPrimaryTopicOf is assumed to be inverse-functional).

In Step (iii), we then apply the T-ground general rules over the corpus with a single scan. Any input or inferred statements matching a consolidation-relevant pattern—including any owl:sameAs statements found—are buffered to a file for later join computation.

Subsequently, in Step (iv), we must now compute the on-disk *canonicalised closure* of the owl:sameAs statements. In particular, we mainly employ the following three on-disk primitives:

(i) **sequential scans** of flat files containing line-delimited tuples; [19]
(ii) **external-sorts** where batches of statements are sorted in memory, the sorted batches written to disk, and the sorted batches merged to the final output; and
(iii) **merge-joins** where multiple sets of data are sorted according to their required join position, and subsequently scanned in an interleaving manner that aligns on the join position and where an

[19] These files are G-Zip compressed flat files of N-Triple-like syntax encoding arbitrary length tuples of RDF constants.

in-memory join is applied for each individual join element.

Using these primitives to compute the owl:sameAs closure minimises the amount of main memory required, where we have presented similar approaches in [30,31].

First, assertional memberships of functional properties, inverse-functional properties and cardinality restrictions (both properties and classes) are written to separate on-disk files. For functional-property and cardinality reasoning, a consistent join variable for the assertional patterns is given by the subject position; for inverse-functional-property reasoning, a join variable is given by the object position. [20] Thus, we can sort the former sets of data according to subject and perform a merge-join by means of a linear scan thereafter; the same procedure applies to the latter set, sorting and merge-joining on the object position. Applying merge-join scans, we produce new owl:sameAs statements.

Both the originally asserted and newly inferred owl:sameAs relations are similarly written to an on-disk file, over which we now wish to perform the canonicalised symmetric/transitive closure. We apply a similar method again, leveraging external sorts and merge-joins to perform the computation (herein, we sketch and point the interested reader to [31, Section 4.6]). In the following, we use $>$ and $<$ to denote lexical ordering, SA as a shortcut for owl:sameAs, a, b, c, etc., to denote members of $\mathbf{U} \cup \mathbf{B}$ such that $a < b < c$, and define URIs to be lexically lower than blank nodes ($\forall u \in \mathbf{U}, \forall v \in \mathbf{B} : u < v$). The process is as follows:

(i) we only materialise symmetric equality relations that involve a (possibly intermediary) canonical term chosen by a lexical ordering: given b SA a, we materialise a SA b; given a SA b SA c, we materialise the relations a SA b, a SA c, and their inverses, but do not materialise b SA c or its inverse;

(ii) transitivity is supported by iterative merge-join scans:
   - in the scan, if we find c SA a (sorted by object) and c SA d (sorted naturally), we infer a SA d and drop the non-canonical c SA d (and d SA c);
   - at the end of the scan:
     - newly inferred triples are marked and merge-joined into the main equality data;
     - any triples echoing an earlier inference are ignored;
     - dropped non-canonical statements are removed;
   - the process is then iterative: in the next scan, if we find d SA a and d SA e, we infer a SA e and e SA a;
   - inferences will only occur if they involve a statement added in the previous iteration, ensuring that inference steps are not re-computed and that the computation will terminate;

(iii) the above iterations stop when a fixpoint is reached and nothing new is inferred;

(iv) the process of reaching a fixpoint is accelerated using available main-memory to store a cache of partial equality chains.

The above steps follow well-known methods for transtive closure computation, modified to support the symmetry of owl:sameAs and to use adaptive canonicalisation in order to avoid quadratic output. With respect to the last item, we use Algorithm 1 to derive "batches" of in-memory equivalences, and when in-memory capacity is achieved, we write these batches to disk and proceed with on-disk computation: this is particularly useful for computing the small number of long equality chains, which would otherwise require sorts and merge-joins over all of the canonical owl:sameAs data currently derived, and where the number of iterations would otherwise be the length of the longest chain. The result of this process is a set of canonicalised equality relations representing the symmetric/transitive closure.

Next, we briefly describe the process of canonicalising data with respect to this on-disk equality closure, where we again use external-sorts and merge-joins. First, we prune the owl:sameAs index to only maintain relations s_1 SA s_2 such that $s_1 > s_2$; thus, given s_1 SA s_2, we know that s_2 is the canonical identifier, and s_1 is to be rewritten. We then sort the data according to the position that we wish to rewrite, and perform a merge-join over both sets of data, buffering the canonicalised data to an output file. If we want to rewrite multiple positions of a file of tuples (e.g., subject and object), we must rewrite one position, sort the results by the second position, and then rewrite the second position. [21]

Finally, note that in the derivation of owl:sameAs from the consolidation rules **prp-fp**, **prp-ifp**, **cax-maxc2**, the overall process may be iterative. For example, consider:

[20] Although a predicate-position join is also available, we prefer object-position joins that provide smaller batches of data for the in-memory join.

[21] One could instead build an on-disk map for equivalence classes and pivot elements and follow a consolidation method similar to the previous section over the unordered data: however, we would expect such an on-disk index to have a low cache hit-rate given the nature of the data, which would lead to many (slow) disk seeks. An alternative approach might be to hash-partition the corpus and equality index over different machines: however, this would require a non-trivial minimum amount of memory on the cluster.

```
dblp:Axel foaf:isPrimaryTopicOf <http://polleres.net/> .
axel:me foaf:isPrimaryTopicOf <http://axel.deri.ie/> .
<http://polleres.net/> owl:sameAs <http://axel.deri.ie/> .
```

from which the conclusion that `dblp:Axel` is the same as `axel:me` holds. We see that new `owl:sameAs` relations (either asserted or derived from the consolidation rules) may in turn "align" terms in the join position of the consolidation rules, leading to new equivalences. Thus, for deriving the final `owl:sameAs`, we require a higher-level iterative process as follows:

(i) initially apply the consolidation rules, and append the results to a file alongside the asserted `owl:sameAs` statements found;
(ii) apply the initial closure of the `owl:sameAs` data;
(iii) then, iteratively until no new `owl:sameAs` inferences are found:
  – rewrite the join positions of the on-disk files containing the data for each consolidation rule according to the current `owl:sameAs` data;
  – derive new `owl:sameAs` inferences possible through the previous rewriting for each consolidation rule;
  – re-derive the closure of the `owl:sameAs` data including the new inferences.

The final closed file of `owl:sameAs` data can then be re-used to rewrite the main corpus in two sorts and merge-join scans over subject and object.

5.2. *Distributed approach*

The distributed approach follows quite naturally from the previous discussion. As before, we assume that the input data are evenly pre-distributed over the slave machines (in any arbitrary ordering), where we can then apply the following process:

(i) **run**: scan the distributed corpus (split over the slave machines) in parallel to extract relevant terminological knowledge;
(ii) **gather**: gather terminological data onto the master machine and thereafter bind the terminological patterns of the general/consolidation rules;
(iii) **flood**: flood the rules for reasoning and the consolidation-relevant patterns to all slave machines;
(iv) **run**: apply reasoning and extract consolidation-relevant statements from the input and inferred data;
(v) **gather**: gather all consolidation statements onto the master machine, then in parallel:
  – **local**: compute the closure of the consolidation rules and the `owl:sameAs` data on the master machine;
  – **run**: each slave machine sorts its fragment of the main corpus according to natural order (s, p, o, c);
(vi) **flood**: send the closed `owl:sameAs` data to the slave machines once the distributed sort has been completed;
(vii) **run**: each slave machine then rewrites the subjects of their segment of the corpus, subsequently sorts the rewritten data by object, and then rewrites the objects (of non-`rdf:type` triples) with respect to the closed `owl:sameAs` data.

5.3. *Performance Evaluation*

Applying the above process to our 1.118 billion quadruple corpus took 12.34 h. Extracting the terminological data took 1.14 h with an average idle time of 19 min (27.7%) (one machine took ∼18 min longer than the rest due to processing a large ontology containing ∼2 million quadruples [32]). Merging and aggregating the terminological data took roughly ∼1 min. Applying the reasoning and extracting the consolidation relevant statements took 2.34 h, with an average idle time of 2.5 min (1.8%). Aggregating and merging the consolidation relevant statements took 29.9 min. Thereafter, locally computing the closure of the consolidation rules and the equality data took 3.52 h, with the computation requiring two iterations overall (the minimum possible—the second iteration did not produce any new results). Concurrent to the previous step, the parallel sort of remote data by natural order took 2.33 h with an average idle time of 6 min (4.3%). Subsequent parallel consolidation of the data took 4.8 h with 10 min (3.5%) average idle time—of this, ∼19% of the time was spent consolidating the pre-sorted subjects, ∼60% of the time was spent sorting the rewritten data by object, and ∼21% of the time was spent consolidating the objects of the data.

As before, Table 8 summarises the timing of the task. Focusing on the performance for the entire corpus, the master machine took 4.06 h to coordinate global knowledge, constituting the lower bound on time possible for the task to execute with respect to increasing machines in our setup—in future it may be worthwhile to investigate distributed strategies for computing the `owl:sameAs` closure (which takes 28.5% of the total computation time), but for the moment we mitigate the cost by concurrently running a sort on the slave machines, thus keeping the slaves busy for 63.4% of the time taken for this local aggregation step. The slave machines were,

	Category	1		2		4		8		full-8	
		min	**%**	**min**	**%**	**min**	**%**	**min**	**%**	**min**	**%**
	Total execution time	454.3	100.0	230.2	100.0	127.5	100.0	81.3	100.0	740.4	100.0
MASTER	**Executing**	29.9	6.6	30.3	13.1	32.5	25.5	30.1	37.0	243.6	32.9
MASTER	Aggregate Consolidation Relevant Data	4.3	0.9	4.5	2.0	4.7	3.7	4.6	5.7	29.9	4.0
MASTER	Closing `owl:sameAs`†	24.4	5.4	24.5	10.6	25.4	19.9	24.4	30.0	211.2	28.5
MASTER	Miscellaneous	1.2	0.3	1.3	0.5	2.4	1.9	1.1	1.3	2.5	0.3
	Idle (waiting for slaves)	424.5	93.4	199.9	86.9	95.0	74.5	51.2	63.0	496.8	67.1
	Avg. Executing (total)	448.9	98.8	221.2	96.1	111.6	87.5	56.5	69.4	599.1	80.9
SLAVE	Extract Terminology	44.4	9.8	22.7	9.9	11.8	9.2	6.9	8.4	49.4	6.7
SLAVE	Extract Consolidation Relevant Data	95.5	21.0	48.3	21.0	24.3	19.1	13.2	16.2	137.9	18.6
SLAVE	Initial Sort (by subject)†	98.0	21.6	48.3	21.0	23.7	18.6	11.6	14.2	133.8	18.1
SLAVE	Consolidation	211.0	46.4	101.9	44.3	51.9	40.7	24.9	30.6	278.0	37.5
	Avg. Idle	5.5	1.2	8.9	3.9	37.9	29.7	23.6	29.0	141.3	19.1
	Waiting for peers	0.0	0.0	3.2	1.4	7.1	5.6	6.3	7.7	37.5	5.1
	Waiting for master	5.5	1.2	5.8	2.5	30.8	24.1	17.3	21.2	103.8	14.0

Table 8
Breakdown of timing of distributed extended consolidation w/reasoning where † identifies the master/slave tasks that run concurrently

on average, busy for 80.9% of the total task time; of the idle time, 73.3% was spent waiting for the master machine to aggregate the consolidation relevant data and to finish the closure of `owl:sameAs` data, and the balance (26.7%) was spent waiting for peers to finish (mostly during the extraction of terminological data).

With regards performance evaluation over the 100 million quadruple corpus for a varying number of slave machines, perhaps most notably, the average idle time of the slave machines increases quite rapidly as the number of machines increases. In particular, by **4** machines, the slaves can perform the distributed sort-by-subject task faster than the master machine can perform the local `owl:sameAs` closure. The significant workload of the master machine will eventually become a bottleneck as the number of slave machines increases further. This is also noticeable in terms of overall task time: the respective time savings as the number of slave machines doubles (moving from **1** slave machine to **8**) are 0.507×, 0.554× and 0.638× respectively.

Briefly, we also ran the consolidation without the general reasoning rules (Table 3) motivated earlier. With respect to performance, the main variations were given by (i) the extraction of consolidation relevant statements—this time directly extracted from explicit statements as opposed to explicit and inferred statements—which took 15.4 min (11% of the time taken including the general reasoning) with an average idle time of less than one minute (6% average idle time); (ii) local aggregation of the consolidation relevant statements took 17 min (56.9% of the time taken previously); (iii) local closure of the `owl:sameAs` data took 3.18 h (90.4% of the time taken previously). The total time saved equated to 2.8 h (22.7%), where 33.3 min were saved from coordination on the master machine, and 2.25 h were saved from parallel execution on the slave machines.

5.4. *Results Evaluation*

In this section, we present the results of the consolidation, which included the general reasoning step in the extraction of the consolidation-relevant statements. In fact, we found that the only major variation between the two approaches was in the amount of consolidation-relevant statements collected (discussed presently), where the changes in the total amounts of coreferent identifiers, equivalence classes, and canonicalised terms were in fact negligible (<0.1%). Thus, for our corpus, extracting only asserted consolidation-relevant statements offered a very close approximation of the extended reasoning approach. [22]

Extracting the terminological data, we found authoritative declarations of 434 functional properties, 57 inverse-functional properties, and 109 cardinality restrictions with a value of `1`. We discarded some third-party, non-authoritative declarations of inverse-functional or functional properties, many of which were simply "echoing" the authoritative semantics of those properties; e.g., many people copy the FOAF vocabulary definitions into their local documents. We also found some other documents on the `bio2rdf.org` domain that define a number of popular third-party properties as

[22] At least in terms of pure *quantity*. However, we do not give an indication of the *quality* or *importance* of those few equivalences we miss with this approximation, which may well be application specific.

being functional, including dc:title, dc:identifier, foaf:name, etc.[23] However, these particular axioms would only affect consolidation for literals; thus, we can say that if applying only the consolidation rules, also considering non-authoritative definitions would not affect the owl:sameAs inferences for our corpus. Again, non-authoritative reasoning over general rules for a corpus such as ours quickly becomes infeasible [9].

We (again) gathered 3.77 million owl:sameAs triples, as well as 52.93 million memberships of inverse-functional properties, 11.09 million memberships of functional properties, and 2.56 million cardinality-relevant triples. Of these, respectively 22.14 million (41.8%), 1.17 million (10.6%) and 533 thousand (20.8%) were asserted—however, in the resulting closed owl:sameAs data derived with and without the extra reasoned triples, we detected a variation of less than 12 thousand terms (0.08%), where only 129 were URIs, and where other variations in statistics were less than 0.1% (e.g., there were 67 less equivalence classes when the reasoned triples were included).

From previous experiments for older RDF Web data [30], we were aware of certain values for inverse-functional properties and functional properties that are erroneously published by exporters, and which cause massive incorrect consolidation. We again blacklist statements featuring such values from our consolidation processing, where we give the top 10 such values encountered for our corpus in Table 9—this blacklist is the result of trial and error, manually inspecting large equivalence classes and the most common values for (inverse-)functional properties. Empty literals are commonly exported (with and without language tags) as values for inverse-functional-properties (particularly FOAF "chat-ID properties"). The literal "08445a31a78661b5c746feff39a9db6e4e2cc5cf" is the SHA-1 hash of the string 'mailto:', commonly assigned as a foaf:mbox_sha1sum value to users who don't specify their email in some input form. The remaining URIs are mainly user-specified values for foaf:homepage, or values automatically assigned for users that don't specify such.[24]

During the computation of the owl:sameAs closure, we found zero inferences through cardinality rules, 106.8 thousand raw owl:sameAs inferences through functional-property reasoning, and 8.7 million raw owl:sameAs inferences through inverse-functional-property reasoning. The final canonicalised, closed, and non-symmetric owl:sameAs index (such that s_1 SA s_2, $s_1 > s_2$, and s_2 is a canonical identifier) contained 12.03 million statements.

#	Blacklisted Term	Occurrences
1	empty literals	584,735
2	<http://null>	414,088
3	<http://www.vox.com/gone/>	150,402
4	"08445a31a78661b5c746feff39a9db6e4e2cc5cf"	58,338
5	<http://www.facebook.com>	6,988
6	<http://facebook.com>	5,462
7	<http://www.google.com>	2,234
8	<http://www.facebook.com/>	1,254
9	<http://google.com>	1,108
10	<http://null.com>	542

Table 9
Top ten most frequently occurring blacklisted values

From this data, we generated 2.82 million equivalence classes (an increase of 1.31× from baseline consolidation) mentioning a total of 14.86 million terms (an increase of 2.58× from baseline—5.77% of all URIs and blank-nodes), of which 9.03 million were blank-nodes (an increase of 2173× from baseline—5.46% of all blank-nodes) and 5.83 million were URIs (an increase of 1.014× from baseline—6.33% of all URIs). Thus, we see a large expansion in the amount of blank-nodes consolidated, but only minimal expansion in the set of URIs referenced in the equivalence classes. With respect to the canonical identifiers, 641 thousand (22.7%) were blank-nodes and 2.18 million (77.3%) were URIs.

Figure 5 contrasts the equivalence class sizes for the baseline approach (seen previously in Figure 3), and for the extended reasoning approach. Overall, there is an observable increase in equivalence class sizes, where we see the average equivalence class size grow to 5.26 entities (1.98× baseline), the largest equivalence class size grow to 33,052 (3.9× baseline) and the percentage of equivalence classes with the minimum size 2 drop to 63.1% (from 74.1% in baseline).

In Table 10, we update the five largest equivalence classes. Result **2** carries over from the baseline consolidation. The rest of the results are largely intra-PLD equivalences, where the entity is described using thousands of blank-nodes, with a consistent (inverse-)functional property value attached. Result **1** refers to a meta-user—labelled Team Vox—commonly appearing in user-FOAF exports on the Vox blogging platform.[25] Result **3** refers to a person identified using blank-nodes (and once by URI) in thousands of RDF documents resident on the same server. Result **4** refers to the Im-

[23] cf. http://bio2rdf.org/ns/bio2rdf:Topic; *offline as of* 2011/06/20
[24] Our full blacklist contains forty-one such values, and can be found at http://aidanhogan.com/swse/blacklist.txt.
[25] This site shut down on 2010/09/30.

age Bioinformatics Research Group in the University of Oxford—labelled IBRG—where again it is identified in thousands of documents using different blank-nodes, but a consistent foaf:homepage. Result **5** is similar to result **1**, but for a Japanese version of the Vox user.

We performed an analogous manual inspection of 1,000 coreferent pairs as per the sampling used previously in Section 4.4 for the baseline consolidation results. This time, we selected SAME 823× (82.3%), TRIVIALLY SAME 145× (14.5%), DIFFERENT 23× (2.3%) and UNCLEAR 9× (0.9%). This gives an observed precision for the sample of ∼97.7% (vs. ∼97.2% for baseline). After applying our blacklisting, the extended consolidation results are, in fact, slightly more precise (on average) than the baseline equivalences: this is due in part to a high number of blank-nodes being consolidated correctly from very uniform exporters of FOAF data that "dilute" the incorrect results. [26]

Figure 6 presents a similar analysis to Figure 5, this time looking at identifiers on a PLD-level granularity. Interestingly, the difference between the two approaches is not so pronounced, initially indicating that many of the additional equivalences found through the consolidation rules are "intra-PLD". In the baseline consolidation approach, we determined that 57% of equivalence classes were inter-PLD (contain identifiers from more than one PLD), with the plurality of equivalence classes containing identifiers from precisely two PLDs (951 thousand, 44.1%); this indicates that explicit owl:sameAs relations are commonly asserted between PLDs. In the extended consolidation approach (which of course subsumes the above results), we determined that the percentage of inter-PLD equivalence classes dropped *to* 43.6%, with the majority of equivalence classes containing identifiers from only one PLD (1.59 million, 56.4%). The entity with the most diverse identifiers (the observable outlier on the x-axis in Figure 6) was the person "Dan Brickley"—one of the founders and leading contributors of the FOAF project—with 138 identifiers (67 URIs and 71 blank-nodes) minted in 47 PLDs; various other prominent community members and some country identifiers also featured high on the list.

In Table 11, we compare the consolidation of the top five ranked identifiers in the SWSE system (see [32]). The results refer respectively to (i) the (co-)founder of the Web "Tim Berners-Lee"; (ii) "Dan Brickley" as aforementioned; (iii) a meta-user for the microblogging platform StatusNet, which exports RDF; (iv) the "FOAF-a-matic" FOAF profile generator (linked from many diverse domains hosting FOAF profiles it created); and (v) "Evan Prodromou", founder of the identi.ca/StatusNet micro-blogging service and platform. We see a significant increase in equivalent identifiers found for these results; however, we also noted that after reasoning consolidation, Dan Brickley was conflated with a second person. [27]

Note that the most frequently co-occurring PLDs in our equivalence classes remained unchanged from Table 7.

During the rewrite of the main corpus, terms in 151.77 million subject positions (13.58% of all subjects) and 32.16 million object positions (3.53% of non-rdf:type objects) were rewritten, giving a total of 183.93 million positions rewritten (1.8× the baseline consolidation approach). In Figure 7, we compare the re-use of terms across PLDs before consolidation, after baseline consolidation, and after the extended reasoning consolidation. Again, although there is an increase in re-use of identifiers across PLDs, we note that: (i) the vast majority of identifiers (about 99%) still only appear in one PLD; (ii) the difference between the baseline and extended reasoning approach is not so pronounced. The most widely referenced consolidated entity—in terms of unique PLDs—was "Evan Prodromou" as aforementioned, referenced with six equivalent URIs in 101 distinct PLDs.

In summary, we conclude that applying the consolidation rules directly (without more general reasoning) is currently a good approximation for Linked Data, and that in comparison to the baseline consolidation over explicit owl:sameAs, (i) the additional consolidation rules generate a large bulk of intra-PLD equivalences for blank-nodes; [28] (ii) relatedly, there is only a minor expansion (1.014×) in the number of URIs involved in the consolidation; (iii) with blacklisting, the overall precision remains roughly stable.

6. Statistical Concurrence Analysis

Having looked extensively at the subject of consolidating Linked Data entities, in this section, we now introduce methods for deriving a weighted concurrence score between entities in the Linked Data corpus: we

[26] We make these and later results available for review at http://swse.deri.org/entity/.

[27] Domenico Gendarmi with three URIs—one document assigns one of Dan's foaf:mbox_sha1sum values (for danbri@w3.org) to Domenico: http://foafbuilder.qdos.com/people/myriamleggieri.wordpress.com/foaf.rdf

[28] We believe this to be due to FOAF publishing practices whereby a given exporter uses consistent inverse-functional property values instead of URIs to uniquely identify entities across local documents.

#	Canonical Term (Lexically Lowest in Equivalence Class)	Size	Correct?
1	bnode37@http://a12iggymom.vox.com/profile/foaf.rdf	33,052	✓
2	http://bio2rdf.org/dailymed_drugs:1000	8,481	×
3	http://ajft.org/rdf/foaf.rdf#_me	8,140	✓
4	bnode4@http://174.129.12.140:8080/tcm/data/association/100	4,395	✓
5	bnode1@http://aaa977.vox.com/profile/foaf.rdf	1,977	✓

Table 10
Largest 5 equivalence classes after extended consolidation

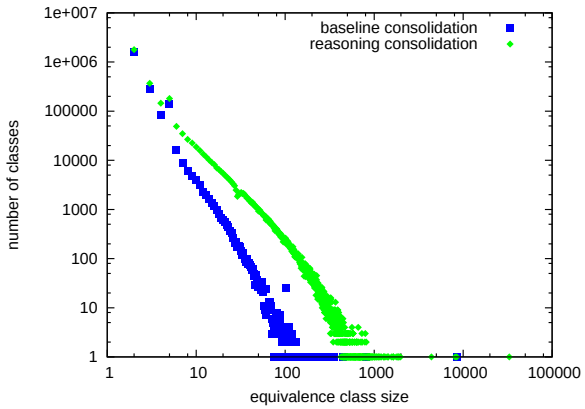


Fig. 5. Distribution of the number of identifiers per equivalence classes for baseline consolidation and extended reasoning consolidation (log/log)

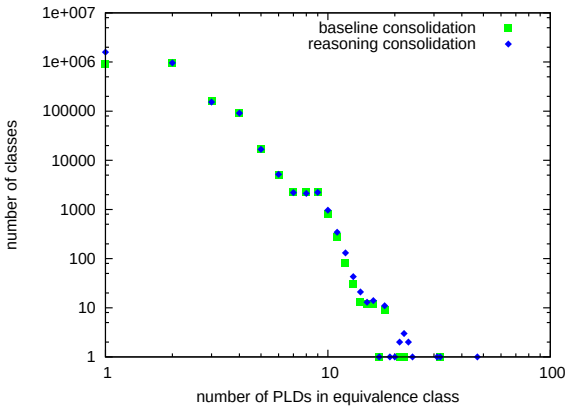


Fig. 6. Distribution of the number of PLDs per equivalence class for baseline consolidation and extended reasoning consolidation (log/log)

#	Canonical Identifier	BL#	R#
1	<http://dblp.l3s.de/.../Tim_Berners-Lee>	26	50
2	<genid:danbri>	10	138
3	<http://update.status.net/>	0	0
4	<http://www.ldodds.com/foaf/foaf-a-matic>	0	0
5	<http://update.status.net/user/1#acct>	0	6

Table 11
Number of equivalent identifiers found for the top-five ranked entities in SWSE with respect to baseline consolidation (**BL#**) and reasoning consolidation (R#)

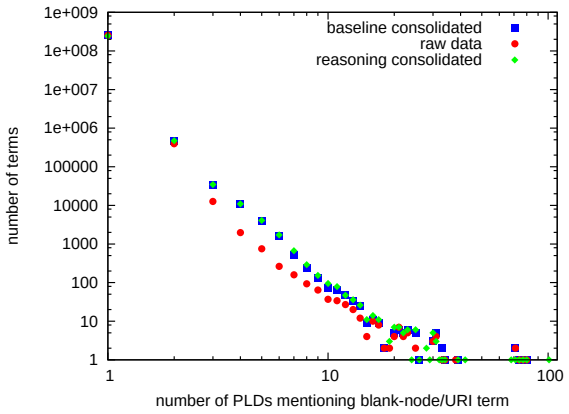


Fig. 7. Distribution of number of PLDs the terms are referenced by, for the raw, baseline consolidated, and reasoning consolidated data (log/log)

define *entity concurrence* as the sharing of outlinks, inlinks and attribute values, denoting a specific form of similarity. Conceptually, concurrence generates scores between two entities based on their shared inlinks, outlinks, or attribute values. More "discriminating" shared characteristics lend a higher concurrence score; for example, two entities based in the same village are more concurrent than two entities based in the same country. How discriminating a certain characteristic is is determined through statistical selectivity analysis. The more characteristics two entities share, (typically) the higher their concurrence score will be.

We use these concurrence measures to materialise new links between related entities, thus increasing the interconnectedness of the underlying corpus. We also leverage these concurrence measures in Section 7 for disambiguating entities, where, after identifying that an equivalence class is likely erroneous and causing inconsistency, we use concurrence scores to realign the most similar entities.

In fact, we initially investigated concurrence as a speculative means of increasing the recall of consolidation for Linked Data; the core premise of this investigation was that very similar entities—with higher concurrence scores—are likely to be coreferent. Along these lines, the methods described herein are based on preliminary works [34], where we:

– investigated domain-agnostic statistical methods for performing consolidation and identifying equivalent entities;
– formulated an initial small-scale (5.6 million triples) evaluation corpus for the statistical consolidation using reasoning consolidation as a best-effort "gold-standard".

Our evaluation gave mixed results where we found some correlation between the reasoning consolidation and the statistical methods, but we also found that our methods gave incorrect results at high degrees of confidence for entities that were clearly not equivalent, but shared many links and attribute values in common. This highlights a crucial fallacy in our speculative approach: in almost all cases, even the highest degree of similarity/-concurrence does not necessarily indicate equivalence or co-reference (cf. [27, Section 4.4]). Similar philosophical issues arise with respect to handling transitivity for the weighted "equivalences" derived [16,39].

However, deriving weighted concurrence measures has applications other than approximative consolidation: in particular, we can materialise named relationships between entities that share a lot in common, thus increasing the level of inter-linkage between entities in the corpus. Again, as we will see later, we can leverage the concurrence metrics to repair equivalence classes found to be erroneous during the disambiguation step of Section 7. Thus, we present a modified version of the statistical analysis presented in [34], describe a (novel) scalable and distributed implementation thereof, and finally evaluate the approach with respect to finding highly-concurring entities in our 1 billion triple Linked Data corpus.

6.1. *High-level approach*

Our statistical concurrence analysis inherits similar *primary requirements* to that imposed for consolidation: the approach should be **scalable**, **fully automatic**, and **domain agnostic** to be applicable in our scenario. Similarly, with respect to *secondary criteria*, the approach should be **efficient to compute**, should give **high precision**, and should give **high recall**. Compared to consolidation, high precision is not as critical for our statistical use-case: in SWSE, we aim to use concurrence measures as a means of *suggesting* additional navigation steps for users browsing the entities—if the suggestion is uninteresting, it can be ignored, whereas incorrect consolidation will often lead to conspicuously garbled results, aggregating data on multiple disparate entities.

Thus, our requirements (particularly for scale) preclude the possibility of complex analyses or any form of pair-wise comparison, etc. Instead, we aim to design lightweight methods implementable by means of distributed sorts and scans over the corpus. Our methods are designed around the following intuitions and assumptions:

(i) the concurrence of entities is measured as a function of their *shared pairs*, be they predicate-subject (loosely, inlinks), or predicate-object pairs (loosely, outlinks or attribute values);
(ii) the concurrence measure should give a *higher weight to exclusive shared-pairs*—pairs that are typically shared by few entities, for edges (predicates) that typically have a low in-degree/out-degree;
(iii) with the possible exception of correlated pairs, *each additional shared pair should increase the concurrence* of the entities—a shared pair cannot reduce the measured concurrence of the sharing entities;
(iv) *strongly exclusive property-pairs should be more influential than a large set of weakly exclusive pairs*;
(v) *correlation* may exist between shared pairs—e.g., two entities may share an inlink and an inverse-outlink to the same node (e.g., `foaf:depiction`, `foaf:depicts`), or may share a large number of shared pairs for a given property (e.g., two entities co-authoring one paper are more likely to co-author subsequent papers)—where we wish to dampen the cumulative effect of correlation in the concurrence analysis;
(vi) the *relative value* of the concurrence measure is important; the absolute value is unimportant.

In fact, the concurrence analysis follows a similar principle to that for consolidation, where instead of considering discrete functional and inverse-functional properties as given by the semantics of the data, we attempt to identify properties that are quasi-functional, quasi-inverse-functional, or what we more generally term *exclusive*: we determine the degree to which the values of properties (here abstracting directionality) are unique to an entity or set of entities. The concurrence between two entities then becomes an aggregation of the weights for the property-value pairs they share in common.

Consider the following running-example:

```
dblp:AliceB10 foaf:maker ex:Alice .
dblp:AliceB10 foaf:maker ex:Bob .

ex:Alice foaf:gender "female" .
ex:Alice foaf:workplaceHomepage <http://wonderland.com> .

ex:Bob foaf:gender "male" .
ex:Bob foaf:workplaceHomepage <http://wonderland.com> .

ex:Claire foaf:gender "female" .
ex:Claire foaf:workplaceHomepage <http://wonderland.com> .
```

where we want to determine the level of (relative) concurrence between three colleagues: `ex:Alice`, `ex:Bob` and `ex:Claire`: i.e., how much do they coincide/concur with respect to exclusive shared pairs.

6.1.1. *Quantifying concurrence*

First, we want to characterise the uniqueness of properties; thus, we analyse their *observed* cardinality and inverse-cardinality as found in the corpus (in contrast to their defined cardinality as possibly given by the formal semantics):

Definition 1 (Observed Cardinality) *Let G be an RDF graph, p be a property used as a predicate in G and s be a subject in G. The* observed cardinality *(or henceforth in this section, simply* cardinality*)* of p wrt s in G*, denoted* $\mathrm{Card}_G(p, s)$*, is the cardinality of the set* $\{o \in \mathbf{C} \mid (s, p, o) \in G\}$.

Definition 2 (Observed Inverse-Cardinality) *Let G and p be as before, and let o be an object in G. The* observed inverse-cardinality *(or henceforth in this section, simply* inverse-cardinality*)* of p wrt o in G*, denoted* $\mathrm{ICard}_G(p, o)$*, is the cardinality of the set* $\{s \in \mathbf{U} \cup \mathbf{B} \mid (s, p, o) \in G\}$.

Thus, loosely, the observed cardinality of a property-subject pair is the number of unique objects it appears within the graph (or unique triples it appears in); letting G_{ex} denote our example graph, then, e.g., $\mathrm{Card}_{G_{ex}}(\texttt{foaf:maker}, \texttt{dblp:AliceB10}) = 2$. We see this value as a good indicator of how *exclusive* (or *selective*) a given property-subject pair is, where sets of entities appearing in the object position of low-cardinality pairs are considered to concur more than those appearing with high-cardinality pairs. The observed inverse-cardinality of a property-object pair is the number of unique subjects it appears with in the graph—e.g., $\mathrm{ICard}_{G_{ex}}(\texttt{foaf:gender}, \texttt{"female"}) = 2$. Both directions are considered analogous for deriving concurrence scores—note however that we do not consider concurrence for literals (i.e., we do not derive concurrence for literals that share a given predicate-subject pair; we do of course consider concurrence for subjects with the same literal value for a given predicate).

To avoid unnecessary duplication, we henceforth focus on describing only the inverse-cardinality statistics of a property, where the analogous metrics for plain-cardinality can be derived by switching subject and object (that is, switching directionality)—we choose the inverse direction as perhaps being more intuitive, indicating concurrence of entities in the subject position based on the predicate-object pairs they share.

Definition 3 (Average Inverse-Cardinality) *Let G be an RDF graph, and p be a property used as a predicate in G. The* average inverse-cardinality (AIC) of p with respect to G*, written* $\mathrm{AIC}_G(p)$*, is the average of the non-zero inverse-cardinalities of p in the graph G. Formally:*

$$\mathrm{AIC}_G(p) = \frac{|\{(s, o) \mid (s, p, o) \in G\}|}{|\{o \mid \exists s : (s, p, o) \in G\}|} \ .$$

The average cardinality $\mathrm{AC}_G(p)$ of a property p is defined analogously as for $\mathrm{AIC}_G(p)$. Note that the (inverse-)cardinality value of any term appearing as a predicate in the graph is necessarily greater-than or equal-to one: the numerator is by definition greater-than or equal-to the denominator. Taking an example, $\mathrm{AIC}_{G_{ex}}(\texttt{foaf:gender}) = 1.5$, which can be viewed equivalently as the average non-zero cardinalities of `foaf:gender` (1 for `"male"` and 2 for `"female"`), or the number of triples with predicate `foaf:gender` divided by the number of unique values appearing in the object position of such triples ($\frac{3}{2}$).

We call a property p for which we observe $\mathrm{AIC}_G(p) \approx 1$, a *quasi-inverse-functional property* with respect to the graph G, and analogously properties for which we observe $\mathrm{AC}_G(p) \approx 1$ as *quasi-functional properties*. We see the values of such properties—in their respective directions—as being very exceptional: very rarely shared by entities. Thus, we would expect a property such as `foaf:gender` to have a high $\mathrm{AIC}_G(p)$ since there are only two object-values (`"male"`, `"female"`) shared by a large number of entities, whereas we would expect a property such as `foaf:workplaceHomepage` to have a lower $\mathrm{AIC}_G(p)$ since there are arbitrarily many values to be shared amongst the entities; given such an observation, we then surmise that a shared `foaf:gender` value represents a weaker "indicator" of concurrence than a shared value for `foaf:workplaceHomepage`.

Given that we deal with incomplete information under the Open World Assumption underlying RDF(S)/OWL, we also wish to weigh the average (inverse) cardinality values for properties with a low number of observations towards a global mean—consider a

fictional property ex:maritalStatus for which we only encounter a few predicate-usages in a given graph, and consider two entities given the value "married": given sparse inverse-cardinality observations, we may naïvely over-estimate the significance of this property-object pair as an indicator for concurrence. Thus, we use a credibility formula as follows to weight properties with few observations towards a global mean:

Definition 4 (Adjusted Average Inverse-Cardinality) *Let p be a property appearing as a predicate in the graph G. The* adjusted average inverse-cardinality (AAIC) *of p with respect to G, written $\mathrm{AAIC}_G(p)$, is then*

$$\mathrm{AAIC}_G(p) = \frac{\mathrm{AIC}_G(p) \times |G^{\overrightarrow{p}}| + \overline{\mathrm{AIC}_G} \times |G^{\rightarrow}|}{|G^{\overrightarrow{p}}| + |G^{\rightarrow}|} \tag{1}$$

where $|G^{\overrightarrow{p}}|$ is the number of distinct objects that appear in a triple with p as a predicate (the denominator of Definition 3), $\overline{\mathrm{AIC}_G}$ is the average inverse-cardinality for all predicate-object pairs (formally, $\overline{\mathrm{AIC}_G} = \frac{|G|}{|\{(p,o)|\exists s:(s,p,o)\in G\}|}$), and $|G^{\rightarrow}|$ is the average number of distinct objects for all predicates in the graph (formally, $|G^{\rightarrow}| = \frac{|\{(p,o)|\exists s:(s,p,o)\in G\}|}{|\{p|\exists s,\exists o:(s,p,o)\in G\}|}$).

Again, the adjusted average cardinality $\mathrm{AAC}_G(p)$ of a property p is defined analogously as for $\mathrm{AAIC}_G(p)$. Some reduction of Equation 1 is possible if one considers that $\mathrm{AIC}_G(p) \times |G^{\overrightarrow{p}}| = |\{(s,o) \mid (s,p,o) \in G\}|$ also denotes the number of triples for which p appears as a predicate in graph G, and that $\overline{\mathrm{AIC}_G} \times |G^{\rightarrow}| = \frac{|G|}{|\{p|\exists s,\exists o:(s,p,o)\in G\}|}$ also denotes the average number of triples per predicate. We maintain Equation 1 in the given unreduced form as it more clearly corresponds to the structure of a standard credibility formula: the reading ($\mathrm{AIC}_G(p)$) is dampened towards a mean ($\overline{\mathrm{AIC}_G}$) by a factor determined by the size of the sample used to derive the reading ($|G^{\overrightarrow{p}}|$) relative to the average sample size ($|G^{\rightarrow}|$).

Now, we move towards combining these metrics to determine the concurrence of entities who share a given non-empty set of property-value pairs. To do so, we combine the adjusted average (inverse) cardinality values, which apply generically to properties, and the (inverse) cardinality values, which apply to a given property-value pair. For example, take the property foaf:workplaceHomepage: entities that share a value referential to a large company—e.g., http://google.com/—should not gain as much concurrence as entities that share a value referential to a smaller company—e.g., http://deri.ie/. Conversely, consider a fictional property ex:citizenOf, which relates a citizen to its country, and for which we find many observations in our corpus, returning a high AAIC value, and consider that only two entities share the value ex:Vanuatu for this property: given that our data are incomplete, we can use the high AAIC value of ex:citizenOf to determine that the property is usually not exclusive, and that it is generally not a good indicator of concurrence. [29]

We start by assigning a coefficient to each pair (p, o) and each pair (p, s) that occur in the dataset, where the coefficient is an indicator of how exclusive that pair is:

Definition 5 (Concurrence Coefficients) *The* concurrence-coefficient of a predicate-subject pair (p, s) *with respect to a graph G is given as:*

$$\mathrm{C}_G(p,s) = \frac{1}{\mathrm{Card}_G(p,s) \times \mathrm{AAC}_G(p)}$$

and the concurrence-coefficient of a predicate-object pair (p, o) *with respect to a graph G is analogously given as:*

$$\mathrm{IC}_G(p,o) = \frac{1}{\mathrm{ICard}_G(p,o) \times \mathrm{AAIC}_G(p)} \ .$$

Again, please note that these coefficients fall into the interval $]0, 1]$ since the denominator, by definition, is necessarily greater than one.

To take an example, let p_{wh} = foaf:workplaceHomepage and say that we compute $\mathrm{AAIC}_G(p_{wh})$ = 7 from a large number of observations, indicating that each workplace homepage in the graph G is linked to by, on average, seven employees. Further, let o_g = http://google.com/ and assume that o_g occurs 2,000 times as a value for p_{wh}: $\mathrm{ICard}_G(p_{wh}, o_g)$ = 2,000; now, $\mathrm{IC}_G(p_{wh}, o_g) = \frac{1}{2{,}000\times 7} = 0.00007$. Also, let o_d = http://deri.ie/ such that $\mathrm{ICard}_G(p_{wh}, o_d)$ = 100; now, $\mathrm{IC}_G(p_{wh}, o_d) = \frac{1}{10\times 7} \approx 0.00143$. Here, sharing DERI as a workplace will indicate a higher level of concurrence than analogously sharing Google.

Finally, we require some means of aggregating the coefficients of the set of pairs that two entities share to derive the final concurrence measure.

Definition 6 (Aggregated Concurrence Score) *Let $Z = (z_1, \ldots z_n)$ be a tuple such that for each*

[29] Here, we try to distinguish between property-value pairs that are exclusive in reality (i.e., on the level of what's signified) and those that are exclusive in the given graph. Admittedly, one could think of counter-examples where not including the general statistics of the property may yield a better indication of weighted concurrence, particularly for generic properties that can be applied in many contexts; for example, consider the exclusive predicate-object pair (skos:subject, category:Koenigsegg_vehicles) given for a non-exclusive property.

$i = 1, \ldots, n$, $z_i \in]0, 1]$. *The aggregated concurrence value* ACS_n *is computed iteratively: starting with* $\text{ACS}_0 = 0$*, then for each* $k = 1 \ldots n$, $\text{ACS}_k = z_k + \text{ACS}_{k-1} - z_k * \text{ACS}_{k-1}$.

The computation of the ACS value is the same process as determining the probability of two independent events occurring—$P(A \vee B) = P(A) + P(B) - P(A * B)$—which is by definition commutative and associative, and thus computation is independent of the order of the elements in the tuple. It may be more accurate to view the coefficients as *fuzzy values*, and the aggregation function as a disjunctive combination in some extensions of fuzzy logic [75].

However, the underlying coefficients may not be derived from strictly independent phenomena: there may indeed be correlation between the property-value pairs that two entities share. To illustrate, we reintroduce a relevant example from [34] shown in Figure 8, where we see two researchers that have co-authored many papers together, have the same affiliation, and are based in the same country.

This example illustrates three categories of concurrence correlation:

(i) *same-value correlation* where two entities may be linked to the same value by multiple predicates in either direction (e.g., `foaf:made`, `dc:creator`, `swrc:author`, `foaf:maker`);

(ii) *intra-property correlation* where two entities that share a given property-value pair are likely to share further values for the same property (e.g., co-authors sharing one value for `foaf:made` are more likely to share further values);

(iii) *inter-property correlation* where two entities sharing a given property-value pair are likely to share further distinct but related property-value pairs (e.g., having the same value for `swrc:affiliation` and `foaf:based_near`).

Ideally, we would like to reflect such correlation in the computation of the concurrence between the two entities.

Regarding *same-value correlation*, for a value with multiple edges shared between two entities, we choose the shared predicate edge with the lowest AA[I]C value and disregard the other edges: i.e., we only consider the most exclusive property used by both entities to link to the given value and prune the other edges.

Regarding *intra-property correlation*, we apply a lower-level aggregation for each predicate in the set of shared predicate-value pairs. Instead of aggregating a single tuple of coefficients, we generate a bag of tuples $\mathbf{Z} = \{Z_{p_1}, \ldots, Z_{p_n}\}$, where each element Z_{p_i} represents the tuple of (non-pruned) coefficients generated for the predicate p_i.[30] We then aggregate this bag as follows:

$$\text{ACS}(\mathbf{Z}) = \text{ACS}(\text{ACS}(Z_{p_i} * \text{AA[I]C}(p_i))_{Z_{p_i} \in \mathbf{Z}})$$

where AA[I]C is either AAC or AAIC, dependant on the directionality of the predicate-value pair observed. Thus, the total contribution possible through a given predicate (e.g., `foaf:made`) has an upper-bound set as its AA[I]C value, where each successive shared value for that predicate (e.g., each successive co-authored paper) contributes positively (but increasingly less) to the overall concurrence measure.

Detecting and counteracting *inter-property correlation* is perhaps more difficult, and we leave this as an open question.

6.1.2. *Implementing entity-concurrence analysis*

We aim to implement the above methods using sorts and scans, and we wish to avoid any form of complex indexing, or pair-wise comparison. First, we wish to extract the statistics relating to the (inverse-)cardinalities of the predicates in the data. Given that there are 23 thousand unique predicates found in the input corpus, we assume that we can fit the list of predicates and their associated statistics in memory—if such were not the case, one could consider an on-disk map, where we would expect a high cache hit-rate based on the distribution of property occurrences in the data (cf. [32]).

Moving forward, we can calculate the necessary predicate-level statistics by first sorting the data according to natural order (s, p, o, c), and then scanning the data, computing the cardinality (number of distinct objects) for each (s, p) pair, and maintaining the average cardinality for each p found. For inverse-cardinality scores, we apply the same process, sorting instead by (o, p, s, c) order, counting the number of distinct subjects for each (p, o) pair, and maintaining the average inverse-cardinality scores for each p. After each scan, the statistics of the properties are adjusted according to the credibility formula in Equation 4.

We then apply a second scan of the sorted corpus; first we scan the data sorted in natural order, and for each (s, p) pair, for each set of unique objects O_{ps} found thereon, and for each pair in

$$\{(o_i, o_j) \in \mathbf{U} \cup \mathbf{B} \times \mathbf{U} \cup \mathbf{B} \mid o_i, o_j \in O_{ps}, o_i < o_j\}$$

where $<$ denotes lexicographical order, we output the following sextuple to an on-disk file:

$$(o_i, o_j, \text{C}(p, s), p, s, -)$$

[30] For brevity, we omit the graph subscript.

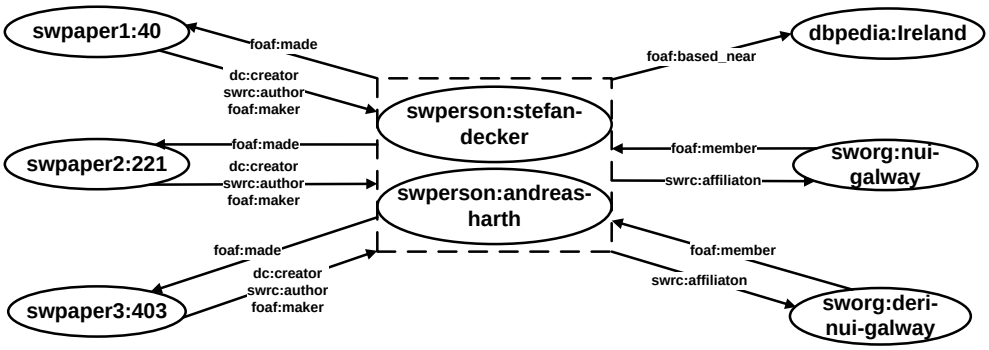


Fig. 8. Example of same-value, inter-property and intra-property correlation, where the two entities under comparison are highlighted in the dashed box, and where the labels of inward-edges (with respect to the principal entities) are italicised and underlined (from [34])

where $\mathrm{C}(p, s) = \frac{1}{|O_{ps}| \times \mathrm{AAC}(p)}$. We apply the same process for the other direction: for each (p, o) pair, for each set of unique subjects S_{po}, and for each pair in

$$\{(s_i, s_j) \in \mathbf{U} \cup \mathbf{B} \times \mathbf{U} \cup \mathbf{B} \mid s_i, s_j \in S_{po}, s_i < s_j\}$$

we output analogous sextuples of the form:

$$(s_i, s_j, \mathrm{IC}(p, o), p, o, +)$$

We call the sets O_{ps} and their analogues S_{po} *concurrence classes*, denoting sets of entities that share the given predicate-subject/predicate-object pair respectively. Here, note that the '$+$' and '$-$' elements simply mark and track the directionality from which the tuple was generated, required for the final aggregation of the co-efficient scores. Similarly, we do not immediately materialise the symmetric concurrence scores, where we instead do so at the end so as to forego duplication of intermediary processing.

Once generated, we can sort the two files of tuples by their natural order, and perform a merge-join on the first two elements—generalising the directional o_i/s_i to simply e_i, each (e_i, e_j) pair denotes two entities that share some predicate-value pairs in common, where we can scan the sorted data and aggregate the final concurrence measure for each (e_i, e_j) pair using the information encoded in the respective tuples. We can thus generate (trivially sorted) tuples of the form (e_i, e_j, s), where s denotes the final aggregated concurrence score computed for the two entities; optionally, we can also write the symmetric concurrence tuples (e_j, e_i, s), which can be sorted separately as required.

Note that the number of tuples generated is quadratic with respect to the size of the respective concurrence class, which becomes a major impediment for scalability given the presence of large such sets—for example, consider a corpus containing 1 million persons sharing the value `"female"` for the property `foaf:gender`, where we would have to generate $\frac{10^{6\times2}-10^6}{2} \approx 500$ billion non-reflexive, non-symmetric concurrence tuples. However, we can leverage the fact that such sets can only invoke a minor influence on the final concurrence of their elements, given that the magnitude of the set—e.g., $|S_{po}|$—is a factor in the denominator of the computed $\mathrm{C}(p, o)$ score, such that $\mathrm{C}(p, o) \propto \frac{1}{|S_{op}|}$. Thus, in practice, we implement a maximum-size threshold for the S_{po} and O_{ps} concurrence classes: this threshold is selected based on a practical upper limit for raw similarity tuples to be generated, where the appropriate maximum class size can trivially be determined alongside the derivation of the predicate statistics. For the purpose of evaluation, we choose to keep the number of raw tuples generated at around $\sim$1 billion, and so set the maximum concurrence class size at 38—we will see more in Section 6.4.

6.2. *Distributed implementation*

Given the previous discussion, our distributed implementation is fairly straight-forward as follows:

(i) **coordinate**: the slave machines split their segment of the corpus according to a modulo-hash function on the subject position of the data, sort the segments, and send the split segments to the peer determined by the hash-function; the slaves simultaneously gather incoming sorted segments, and subsequently perform a merge-sort of the segments;

(ii) **coordinate**: the slave machines apply the same operation, this time hashing on object—triples with `rdf:type` as predicate are not included in the object-hashing; subsequently the slaves merge-sort the segments ordered by object;

(iii) **run**: the slave machines then extract predicate-level statistics, and statistics relating to the

concurrence-class-size distribution, which are used to decide upon the class size threshold;

(iv) **gather/flood/run**: the master machine gathers and aggregates the high-level statistics generated by the slave machines in the previous step and sends a copy of the global statistics back to each machine; the slaves subsequently generate the raw concurrence-encoding sextuples as described before from a scan of the data in both orders;

(v) **coordinate**: the slave machines coordinate the locally generated sextuples according to the first element (join position) as before;

(vi) **run**: the slave machines aggregate the sextuples coordinated in the previous step, and produce the final non-symmetric concurrence tuples;

(vii) **run**: the slave machines produce the symmetric version of the concurrence tuples, and coordinate and sort on the first element.

Here, we make heavy use of the **coordinate** function to align data according to the join position required for the subsequent processing step—in particular, aligning the raw data by subject and object, and then the concurrence tuples analogously.

Note that we do not hash on the object position of `rdf:type` triples: our raw corpus contains 206.8 million such triples, and given the distribution of class memberships, we assume that hashing these values will lead to uneven distribution of data, and subsequently uneven load balancing—e.g., 79.2% of all class memberships are for `foaf:Person`, hashing on which would send 163.7 million triples to one machine, which alone is greater than the average number of triples we would expect per machine (139.8 million). In any case, given that our corpus contains 105 thousand unique values for `rdf:type`, we would expect the average-inverse-cardinality to be approximately 1,970—even for classes with two members, the potential effect on concurrence is negligible.

6.3. *Performance evaluation*

We apply our concurrence analysis over the consolidated corpus derived in Section 5. The total time taken was 13.9 h. Sorting, splitting and scattering the data according to subject on the slave machines took 3.06 h, with an average idle time of 7.7 min (4.2%). Subsequently, merge-sorting the sorted segments took 1.13 h, with an average idle time of 5.4 min (8%). Analogously sorting, splitting and scattering the non-`rdf:type` statements by object took 2.93 h, with an average idle time of 11.8 min (6.7%). Merge sorting the data by object took 0.99 h, with an average idle time of 3.8 min (6.3%). Extracting the predicate statistics and threshold information from data sorted in both orders took 29 min, with an average idle time of 0.6 min (2.1%). Generating the raw, unsorted similarity tuples took 69.8 min with an average idle time of 2.1 min (3%). Sorting and coordinating the raw similarity tuples across the machines took 180.1 min, with an average idle time of 14.1 min (7.8%). Aggregating the final similarity took 67.8 min, with an average idle time of 1.2 min (1.8%).

Table 12 presents a breakdown of the timing of the task. First, with regards application over the full corpus, although this task requires some aggregation of global-knowledge by the master machine, the volume of data involved is minimal: a total of 2.1 minutes is spent on the master machine performing various minor tasks (initialisation, remote calls, logging, aggregation and broadcast of statistics). Thus, 99.7% of the task is performed in parallel on the slave machine. Although there is less time spent waiting for the master machine compared to the previous two tasks, deriving the concurrence measures involves three expensive sort/**coordinate**/merge-sort operations to redistribute and sort the data over the slave swarm. The slave machines were idle for, on average, 5.8% of the total task time; most of this idle time (99.6%) was spent waiting for peers. The master machine was idle for almost the entire task, with 99.7% waiting for the slave machines to finish their tasks—again, interleaving a job for another task would have practical benefits.

Second, with regards the timing of tasks when varying the number of slave machines for 100 million quadruples, we see that the percentage of time spent idle on the slave machines increases from 0.4% (**1**) to 9% (**8**). However, for each incremental doubling of the number of slave machines, the total overall task times are reduced by 0.509×, 0.525× and 0.517× respectively.

6.4. *Results Evaluation*

With respect to data distribution, after hashing on subject we observed an average absolute deviation (average distance from the mean) of 176 thousand triples across the slave machines, representing an average 0.13% deviation from the mean: near-optimal data distribution. After hashing on the object of non-`rdf:type` triples, we observed an average absolute deviation of 1.29 million triples across the machines, representing an average 1.1% deviation from the mean; in particular, we note that one machine was assigned 3.7 million triples above the mean (an additional 3.3% above the mean). Although not optimal, the percentage of data deviation

	Category	1		2		4		8		full-8	
		min	**%**	**min**	**%**	**min**	**%**	**min**	**%**	**min**	**%**
	Total execution time	516.2	100.0	262.7	100.0	137.8	100.0	73.0	100.0	835.4	100.0
MASTER	**Executing**	2.2	0.4	2.3	0.9	3.1	2.3	3.4	4.6	2.1	0.3
MASTER	Miscellaneous	2.2	0.4	2.3	0.9	3.1	2.3	3.4	4.6	2.1	0.3
MASTER	**Idle (waiting for slaves)**	514.0	99.6	260.4	99.1	134.6	97.7	69.6	95.4	833.3	99.7
SLAVE	**Avg. Executing (total exc. idle)**	514.0	99.6	257.8	98.1	131.1	95.1	66.4	91.0	786.6	94.2
SLAVE	Split/sort/scatter (subject)	119.6	23.2	59.3	22.6	29.9	21.7	14.9	20.4	175.9	21.1
SLAVE	Merge-sort (subject)	42.1	8.2	20.7	7.9	11.2	8.1	5.7	7.9	62.4	7.5
SLAVE	Split/sort/scatter (object)	115.4	22.4	56.5	21.5	28.8	20.9	14.3	19.6	164.0	19.6
SLAVE	Merge-sort (object)	37.2	7.2	18.7	7.1	9.6	7.0	5.1	7.0	55.6	6.6
SLAVE	Extract High-level Statistics	20.8	4.0	10.1	3.8	5.1	3.7	2.7	3.7	28.4	3.3
SLAVE	Generate Raw Concurrence Tuples	45.4	8.8	23.3	8.9	11.6	8.4	5.9	8.0	67.7	8.1
SLAVE	Cooordinate/Sort Concurrence Tuples	97.6	18.9	50.1	19.1	25.0	18.1	12.5	17.2	166.0	19.9
SLAVE	Merge-sort/Aggregate Similarity	36.0	7.0	19.2	7.3	10.0	7.2	5.3	7.2	66.6	8.0
SLAVE	**Avg. Idle**	2.2	0.4	4.9	1.9	6.7	4.9	6.5	9.0	48.8	5.8
SLAVE	Waiting for peers	0.0	0.0	2.6	1.0	3.6	2.6	3.2	4.3	46.7	5.6
SLAVE	Waiting for master	2.2	0.4	2.3	0.9	3.1	2.3	3.4	4.6	2.1	0.3

Table 12
Breakdown of timing of distributed concurrence analysis

given by hashing on object is still within the natural variation in run-times we have seen for the slave machines during most parallel tasks.

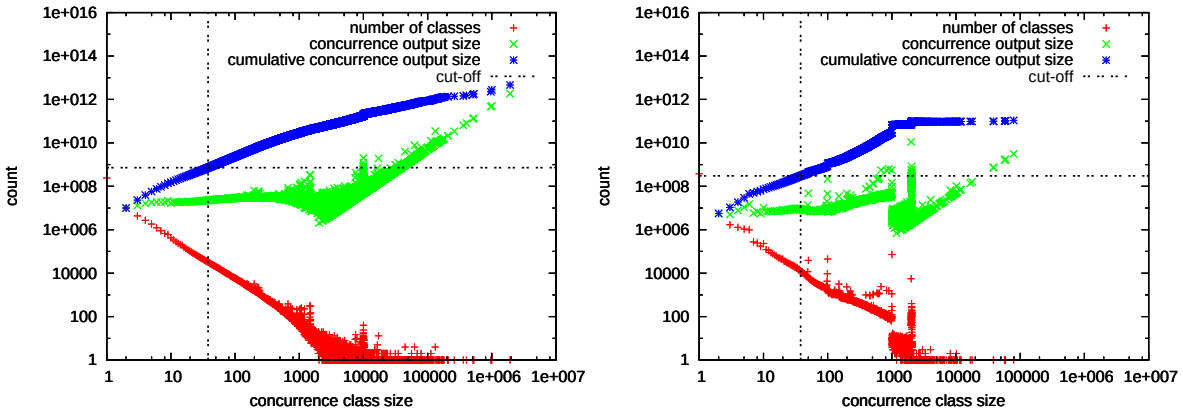
In Figures 9(a) and 9(b), we illustrate the effect of including increasingly large concurrence classes on the number of raw concurrence tuples generated. For the predicate-object pairs, we observe a power-law(-esque) relationship between the size of the concurrence class and the number of such classes observed. Second, we observe that the number of concurrences generated for each increasing class size initially remains fairly static—i.e., larger class sizes give quadratically more concurrences, but occur polynomially less often—until the point where the largest classes that generally only have one occurrence is reached, and the number of concurrences begins to increase quadratically. Also shown is the cumulative count of concurrence tuples generated for increasing class sizes, where we initially see a power-law(-esque) correlation, which subsequently begins to flatten as the larger concurrence classes become more sparse (although more massive).

For the predicate-subject pairs, the same roughly holds true, although we see fewer of the very largest concurrence classes: the largest concurrence class given by a predicate-subject pair was 79 thousand, versus 1.9 million for the largest predicate-object pair, respectively given by the pairs (`kwa:map`, `macs:manual_rameau_lcsh`) and (`opiumfield:rating`, `""`). Also, we observe some "noise" where for milestone concurrence class sizes (esp., at 50, 100, 1,000, 2,000) we observe an unusual amount of classes. For example, there were 72 thousand concurrence classes of precisely size 1,000 (versus 88 concurrence classes at size 996)—the 1,000 limit was due to a FOAF exporter from the `hi5.com` domain, which seemingly enforces that limit on the total "friends count" of users, translating into many users with precisely 1,000 values for `foaf:knows`. [31] Also for example, there were 5.5 thousand classes of size 2,000 (versus 6 classes of size 1,999)—almost all of these were due to an exporter from the `bio2rdf.org` domain, which puts this limit on values for the `b2r:linkedToFrom` property. [32] We also encountered unusually large numbers of classes approximating these milestones, such as 73 at 2,001. Such phenomena explain the staggered "spikes"and "discontinuities" in Figure 9(b), which can be observed to correlate with such milestone values (in fact, similar but less noticeable spikes are also present in Figure 9(a)).

These figures allow us to choose a threshold of concurrence-class size given an upper bound on raw concurrence tuples to generate. For the purposes of evaluation, we choose to keep the number of materialised concurrence tuples at around 1 billion, which limits our maximum concurrence class size to 38 (from which we produce 1.023 billion tuples: 721 million through shared (p, o) pairs and 303 million through (p, s) pairs.

With respect to the statistics of predicates, for the predicate-subject pairs, each predicate had an average of 25,229 unique objects for 37,953 total triples, giving an average cardinality of $\sim$1.5. We give the five predi-

[31] cf. `http://api.hi5.com/rest/profile/foaf/100614697`
[32] cf. `http://bio2rdf.org/mesh:D000123Q000235`



(a) Predicate-object pairs

(b) Predicate-subject pairs

Fig. 9. Breakdown of potential concurrence classes given with respect to predicate-object pairs and predicate-subject pairs respectively, where for each class size on the x-axis (s_i) we show the number of classes (c_i); the raw non-reflexive, non-symmetric concurrence tuples generated ($sp_i = c_i \times \frac{s_i^2+s_i}{2}$); a cumulative count of tuples generated for increasing class sizes ($\widehat{sp_i} = \sum_{j \leq i} sp_j$); and our cut-off ($s_i = 38$) that we've chosen to keep the total number of tuples at ∼1 billion (log/log)

#	Predicate	Objects	Triples	AAC
1	foaf:nick	150,433,035	150,437,864	1.000
2	lldpubmed:journal	6,790,426	6,795,285	1.003
3	rdf:value	2,802,998	2,803,021	1.005
4	eurostat:geo	2,642,034	2,642,034	1.005
5	eurostat:time	2,641,532	2,641,532	1.005

Table 13
Top five predicates with respect to lowest adjusted average cardinality (AAC)

#	Predicate	Subjects	Triples	AAIC
1	lldpubmed:meshHeading	2,121,384	2,121,384	1.009
2	opiumfield:recommendation	1,920,992	1,920,992	1.010
3	fb:type.object.key	1,108,187	1,108,187	1.017
4	foaf:page	1,702,704	1,712,970	1.017
5	skipinions:hasFeature	1,010,604	1,010,604	1.019

Table 14
Top five predicates with respect to lowest adjusted average inverse-cardinality (AAIC)

cates observed to have the lowest adjusted average cardinality in Table 13. These predicates are judged by the algorithm to be the most selective for identifying their subject with a given object value (i.e., quasi-inverse-functional); note that the bottom two predicates will not generate any concurrence scores since they are perfectly unique to a given object (i.e., the number of triples equals the number of objects such that no two entities can share a value for this predicate). For the predicate-object pairs, there was an average of 11,572 subjects for 20,532 triples, giving an average inverse-cardinality of ∼2.64; We analogously give the five predicates observed to have the lowest adjusted average inverse cardinality in Table 14. These predicates are judged to be the most selective for identifying their object with a given subject value (i.e., quasi-functional); however, four of these predicates will not generate any concurrences since they are perfectly unique to a given subject (i.e., those where the number of triples equals the number of subjects).

Aggregation produced a final total of 636.9 million weighted concurrence pairs, with a mean concurrence weight of ∼0.0159. Of these pairs, 19.5 million involved a pair of identifiers from different PLDs (3.1%), whereas 617.4 million involved identifiers from the same PLD; however, the average concurrence value for an intra-PLD pair was 0.446, versus 0.002 for inter-PLD pairs—although fewer intra-PLD concurrences are found, they typically have higher concurrences. [33]

In Table 15, we give the labels of top five most concurrent entities, including the number of pairs they share—the concurrence score for each of these pairs was $>$ 0.9999999. We note that they are all locations, where particularly on WIKIPEDIA (and thus filtering through to DBpedia), properties with location values are typically duplicated (e.g., dbp:deathPlace, dbp:birthPlace, dbp:headquarters—properties that are *quasi-functional*); for example, New York City and New York State are both the dbp:deathPlace of dbpedia:Isacc_Asimov, etc.

[33] Note that we apply this analysis over the consolidated data, and thus this is an approximative reading for the purposes of illustration: we extract the PLDs from canonical identifiers, which are choosen based on arbitrary lexical ordering.

#	Entity Label 1	Entity Label 2	Concur
1	New York City	New York State	791
2	London	England	894
3	Tokyo	Japan	900
4	Toronto	Ontario	418
5	Philadelphia	Pennsylvania	217

Table 15
Top five concurrent entities and the number of pairs they share

#	Ranked Entity	#Con.	"Closest" Entity	Val.
1	Tim Berners-Lee	908	Lalana Kagal	0.83
2	Dan Brickley	2,552	Libby Miller	0.94
3	update.status.net	11	socialnetwork.ro	0.45
4	FOAF-a-matic	21	foaf.me	0.23
5	Evan Prodromou	3,367	Stav Prodromou	0.89

Table 16
Breakdown of concurrences for top five ranked entities in SWSE, ordered by rank, with, respectively, entity label, number of concurrent entities found, the label of the concurrent entity with the largest degree, and finally the degree value

In Table 16, we give a description of the concurrent entities found for the top-five ranked entities—for brevity, again we show entity labels. In particular, we note that a large amount of concurrent entities are identified for the highly-ranked persons. With respect to the strongest concurrences: (i) Tim and his former student Lalana share twelve primarily academic links, coauthoring six papers; (ii) Dan and Libby, co-founders of the FOAF project, share 87 links, primarily 73 `foaf:knows` relations to and from the same people, as well as a co-authored paper, occupying the same professional positions, etc.;[34] (iii) `update.status.net` and `socialnetwork.ro` share a single `foaf:accountServiceHomepage` link from a common user; (iv) similarly, the FOAF-a-matic and `foaf.me` services share a single `mvcb:generatorAgent` inlink; (v) finally, Evan and Stav share 69 `foaf:knows` inlinks and outlinks exported from the `identi.ca` service.

7. Entity Disambiguation and Repair

We have already seen that—even by only exploiting the formal logical consequences of the data through reasoning—consolidation may already be imprecise due to various forms of noise inherent in Linked Data. In this section, we look at trying to detect erroneous coreferences as produced by the extended consolidation approach introduced in Section 5. Ideally, we would like to automatically detect, revise, and repair such cases to improve the effective precision of the consolidation process. As discussed in Section 2.6, OWL 2 RL/RDF contains rules for automatically detecting inconsistencies in RDF data, representing formal contradictions according to OWL semantics. Where such contradictions are created due to consolidation, we believe this to be a good indicator of erroneous consolidation.

Once erroneous equivalences have been detected, we would like to subsequently diagnose and repair the coreferences involved. One option would be to completely disband the entire equivalence class; however, there may often be only one problematic identifier that, e.g., causes inconsistency in a large equivalence class—breaking up all equivalences would be a coarse solution. Instead, herein we propose a more fine-grained method for repairing equivalence classes that incrementally rebuilds coreferences in the set in a manner that preserves consistency and that is based on the original evidences for the equivalences found, as well as concurrence scores (discussed in the previous section) to indicate how similar the original entities are. Once the erroneous equivalence classes have been repaired, we can then revise the consolidated corpus to reflect the changes.

7.1. *High-level approach*

The high-level approach is to see if the consolidation of any entities conducted in Section 5 lead to any *novel* inconsistencies, and subsequently *recant* the equivalences involved; thus, it is important to note that our aim is not to repair inconsistencies in the data, but instead to repair incorrect consolidation symptomised by inconsistency.[35] In this section, we (i) describe what forms of inconsistency we detect and how we detect them; (ii) characterise how inconsistencies can be caused by consolidation using examples from our corpus where possible; (iii) discuss the repair of equivalence classes that have been determined to cause inconsistency.

First, in order to track which data are consolidated and which not, in the previous consolidation step we output sextuples of the form:

$$(s, p, o, c, s', o')$$

where s, p, o, c, denote the consolidated quadruple containing canonical identifiers in the subject/object posi-

[34] Notably, Leigh Dodds (creator of the FOAF-a-matic service) is linked by the property `quaffing:drankBeerWith` to both.

[35] We instead refer the interested reader to [9] for some previous works on the topic of general inconsistency repair for Linked Data.

tion as appropriate, and s' and o' denote the input identifiers prior to consolidation. [36]

To detect inconsistencies in the consolidated corpus, we use the OWL 2 RL/RDF rules with the `false` consequent [24] as listed in Table 4. A quick check of our corpus revealed that one document provides 8 `owl:AsymmetricProperty` and 10 `owl:IrreflexiveProperty` axioms [37], and one directory gives 9 `owl:AllDisjointClasses` axioms [38], and where we found no other OWL 2 axioms relevant to the rules in Table 4.

We also consider an additional rule, whose semantics are indirectly axiomatised by the OWL 2 RL/RDF rules (through **prp-fp**, **dt-diff** and **eq-diff1**), but which we must support directly since we do not consider consolidation of literals:

`?p a owl:FunctionalProperty .`
`?x ?p` $?l_1$ `, ` $?l_2$ `.`
$?l_1$ *owl:differentFrom* $?l_2$ `.`
$\Rightarrow$ `false`

where we underline the terminological pattern. To illustrate, we take an example from our corpus:

```
# Terminological [http://dbpedia.org/data3/length.rdf]
 dpo:length rdf:type owl:FunctionalProperty .
# Assertional [http://dbpedia.org/data/Fiat_Nuova_500.xml]
 dbpedia:Fiat_Nuova_500′ dpo:length "3.546"^^xsd:double .
# Assertional [http://dbpedia.org/data/Fiat_500.xml]
 dbpedia:Fiat_Nuova′ dpo:length "2.97"^^xsd:double .
```

where we use the prime symbol [′] to denote identifiers considered coreferent by consolidation. Here we see two very closely related models of cars consolidated in the previous step, but we now identify that they have two different values for `dpo:length`—a functional-property—and thus the consolidation raises an inconsistency.

Note that we do not expect the *owl:differentFrom* assertion to be materialised, but instead intend a rather more relaxed semantics based on a heurisitic comparison: given two (distinct) literal bindings for $?l_1$ and $?l_2$, we flag an inconsistency iff (i) the data values of the two bindings are not equal (standard OWL semantics); *and* (ii) their lower-case string value (minus language-tags and datatypes) are lexically unequal. In particular, the relaxation is inspired by the definition of the FOAF (datatype) functional properties `foaf:age`, `foaf:gender`, and `foaf:birthday`, where the range of these properties is rather loosely defined: a generic range of `rdfs:Literal` is formally defined for these properties, with informal recommendations to use `male`/`female` as gender values, and `MM-DD` syntax for birthdays, but not giving recommendations for datatype or language-tags. The latter relaxation means that we would not flag an inconsistency in the following data:

```
# Terminological [http://xmlns.com/foaf/spec/index.rdf]
 foaf:Person owl:disjointWith foaf:Document .
# Assertional [fictional]
 ex:Ted foaf:age 25 .
 ex:Ted foaf:age "25" .
 ex:Ted foaf:gender "male" .
 ex:Ted foaf:gender "Male"@en .
 ex:Ted foaf:birthday "25-05"^^xsd:gMonthDay .
 ex:Ted foaf:birthday "25-05" .
```

With respect to these consistency checking rules, we consider the terminological data to be sound. [39] We again only consider terminological axioms that are authoritatively served by their source; for example, the following statement:

`sioc:User owl:disjointWith foaf:Person .`

would have to be served by a document that either `sioc:User` or `foaf:Person` dereferences to (*either* the FOAF or SIOC vocabulary since the axiom applies over a *combination* of FOAF and SIOC assertional data).

Given a grounding for such an inconsistency-detection rule, we wish to analyse the constants bound by variables in join positions to determine whether or not the contradiction is caused by consolidation; we are thus only interested in join variables that appear at least once in a consolidatable position (thus, we do not support **dt-not-type**) and where the join variable is "intra-assertional" (exists twice in the assertional patterns). Other forms of inconsistency must otherwise be present in the raw data.

Further note that `owl:sameAs` patterns—particularly in rule **eq-diff1**—are implicit in the consolidated data; e.g., consider:

```
# Assertional [http://www.wikier.org/foaf.rdf]
 wikier:wikier′ owl:differentFrom eswc2006p:sergio-fernandez′ .
```

where an inconsistency is implicitly given by the `owl:sameAs` relation that holds between the consolidated identifiers `wikier:wikier′` and `eswc2006p:sergio-fernandez′`. In this example, there are two Semantic Web researchers, respectively

[36] We use syntactic shortcuts in our file to denote when $s = s'$ and/or $o = o'$. Maintaining the additional rewrite information during the consolidation process is trivial, where the output of consolidating subjects gives quintuples (s, p, o, c, s'), which are then sorted and consolidated by o to produce the given sextuples.

[37] `http://models.okkam.org/ENS-core-vocabulary#country_of_residence`

[38] `http://ontologydesignpatterns.org/cp/owl/fsdas/`

[39] In any case, we always source terminological data from the raw unconsolidated corpus.

named "Sergio Fernández" [40] and "Sergio Fernández Anzuola" [41] who both participated in the ESWC 2006 conference, and who were subsequently conflated in the "DogFood" export. [42] The former Sergio subsequently added a counter-claim in his FOAF file, asserting the above `owl:differentFrom` statement.

Other inconsistencies do not involve explicit `owl:sameAs` patterns, a subset of which may require "positive" reasoning to be detected; e.g.:

```
# Terminological [http://xmlns.com/foaf/spec]
 foaf:Person owl:disjointWith foaf:Organization .
 foaf:knows rdfs:domain foaf:Person .

# Assertional [http://identi.ca/w3c/foaf]
 identica:48404′ foaf:knows identica:45563 .

# Assertional [inferred by prp-dom]
 identica:48404′ a foaf:Person .

# Assertional [http://data.semanticweb.org/organization/w3c/rdf]
 semweborg:w3c′ a foaf:Organization .
```

where the two entities are initially consolidated due to sharing the value `http://www.w3.org/` for the inverse-functional property `foaf:homepage`; the W3C is stated to be a `foaf:Organization` in one document, and is inferred to be a person from its `identi.ca` profile through rule **prp-dom**; finally, the W3C is a member of two disjoint classes, forming an inconsistency detectable by rule **cax-dw**. [43]

Once the inconsistencies caused by consolidation have been identified, we need to perform a repair of the equivalence class involved. In order to resolve inconsistencies, we make three simplifying assumptions:

(i) the steps involved in the consolidation can be rederived with knowledge of direct inlinks and outlinks of the consolidated entity, or reasoned knowledge derived from there;
(ii) inconsistencies are caused by pairs of consolidated identifiers;
(iii) we repair individual equivalence classes and do not consider the case where repairing one such class may indirectly repair another.

With respect to the first item, our current implementation will be performing a repair of the equivalence class based on knowledge of direct inlinks and outlinks, available through a simple merge-join as used in the previous section; this thus precludes repair of consolidation found through rule **cls-maxqc2**, which also requires knowledge about the class memberships of the outlinked node. With respect to the second item, we say that inconsistencies are caused by pairs of identifiers—what we term *incompatible identifiers*—such that we do not consider inconsistencies caused with respect to a single identifier (inconsistencies not caused by consolidation) and do not consider the case where the alignment of more than two identifiers are required to cause a single inconsistency (not possible in our rules) where such a case would again lead to a disjunction of repair strategies. With respect to the third item, it is possible to resolve a set of inconsistent equivalence classes by repairing one; for example, consider rules with multiple "intra-assertional" join-variables (**prp-irp**, **prp-asyp**) that can have explanations involving multiple consolidated identifiers as follows:

```
# Terminological [fictional]
 :made owl:propertyDisjointWith :maker .

# Assertional [fictional]
 ex:AZ″ :maker ex:entcons′ .
 dblp:Antoine_Zimmermann″ :made dblp:HoganZUPD15′ .
```

where both equivalences together constitute an inconsistency. Repairing one equivalence class would repair the inconsistency detected for both: we give no special treatment to such a case, and resolve each equivalence class independently. In any case, we find no such incidences in our corpus: these inconsistencies require (i) axioms new in OWL 2 (rules **prp-irp**, **prp-asyp**, **prp-pdw** and **prp-adp**); (ii) alignment of two consolidated sets of identifiers in the subject/object positions. Note that such cases can also occur given the recursive nature of our consolidation, whereby consolidating one set of identifiers may lead to alignments in the join positions of the consolidation rules in the next iteration; however, we did not encounter such recursion during the consolidation phase for our data.

The high-level approach to repairing inconsistent consolidation is as follows:

(i) rederive and build a non-transitive, symmetric graph of equivalences between the identifiers in the equivalence class, based on the inlinks and outlinks of the consolidated entity;
(ii) discover identifiers that together cause inconsistency and must be separated, generating a new seed equivalence class for each, and breaking the direct links between them;
(iii) assign the remaining identifiers into one of the seed equivalence classes based on:
  (a) minimum distance in the non-transitive

[40] `http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/f/Fern=aacute=ndez:Sergio.html`
[41] `http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/a/Anzuola:Sergio_Fern=aacute=ndez.html`
[42] `http://data.semanticweb.org/dumps/conferences/eswc-2006-complete.rdf`
[43] Note that this also could be viewed as a counter-example for using inconsistencies to recant consolidation, where arguably the two entities are coreferent from a practical perspective, even if "incompatible" from a symbolic perspective.

equivalence class;
(b) if tied, use a concurrence score.

Given the simplifying assumptions, we can formalise the problem thus: we denote the graph of non-transitive equivalences for a given equivalence class as a weighted graph $G = (V, E, \omega)$ such that $V \subset \mathbf{B} \cup \mathbf{U}$ is the set of vertices, $E \subset \mathbf{B} \cup \mathbf{U} \times \mathbf{B} \cup \mathbf{U}$ is the set of edges, and $\omega : E \mapsto \mathbb{N} \times \mathbb{R}$ is a weighting function for the edges. Our edge weights are pairs (d, c) where d is the number of sets of *input* triples in the corpus that allow to directly derive the given equivalence relation by means of a direct `owl:sameAs` assertion (in either direction), or a shared inverse-functional object, or functional subject—loosely, the independent evidences for the relation given by the input graph, excluding transitive `owl:sameAs` semantics; c is the concurrence score derivable between the unconsolidated entities and is used to resolve ties (we would expect many strongly connected equivalence graphs where, e.g., the entire equivalence class is given by a single shared value for a given inverse-functional property, and we thus require the additional granularity of concurrence for repairing the data in a non-trivial manner). We define a total lexicographical order over these pairs.

Given an equivalence class $Eq \subset \mathbf{U} \cup \mathbf{B}$ that we perceive to cause a *novel* inconsistency—i.e., an inconsistency derivable by the alignment of incompatible identifiers—we first derive a collection of sets $\mathbf{C} = \{C_1, \ldots, C_n\}$, $\mathbf{C} \subset 2^{\mathbf{U} \cup \mathbf{B}}$, such that each $C_i \in \mathbf{C}$, $C_i \subseteq Eq$ denotes an unordered pair of incompatible identifiers.

We then apply a straightforward, greedy *consistent clustering* of the equivalence class, loosely following the notion of a minimal cutting (see, e.g., [63]). For Eq, we create an initial set of singleton sets $\mathbf{Eq}_0$, each containing an individual identifier in the equivalence class. Now let $\Phi(E_i, E_j)$ denote the aggregated weight of the edge considering the merge of the nodes of E_i and the nodes of E_j in the graph: the pair (d, c) such that d denotes the unique evidences for equivalence relations between all nodes in E_i and all nodes in E_j and such that c denotes the concurrence score considering the merge of entities in E_i and E_j—intuitively, the same weight as before, but applied as if the identifiers in E_i and E_j were consolidated in the graph. We can apply the following clustering:

- for each pair of sets $E_i, E_j \in \mathbf{Eq}_n$ such that $\nexists\{a, b\} \in \mathbf{C} : a \in Eq_i, b \in Eq_j$ (i.e., consistently mergeable subsets) identify the weights of $\Phi(E_i, E_j)$ and order the pairings;
- in descending order with respect to the above weights, merge E_i, E_j pairs—such that neither E_i or E_j have already been merged in this iteration—producing E_{n+1} at iteration's end;
- iterate over n until fixpoint.

Thus, we determine the pairs of incompatible identifiers that must necessarily be in different repaired equivalence classes, deconstruct the equivalence class, and then begin reconstructing the repaired equivalence class by iteratively merging the most strongly linked intermediary equivalence classes that will not contain incompatible identifers. [44]

7.2. *Implementing disambiguation*

The implementation of the above disambiguation process can be viewed on two levels: the *macro* level, which identifies and collates the information about individual equivalence classes and their respectively consolidated inlinks/outlinks, and the *micro* level, which repairs individual equivalence classes.

On the macro level, the task assumes input data sorted by both subject (s, p, o, c, s', o') and object (o, p, s, c, o', s'), again such that s, o represent canonical identifiers and s', o' represent the original identifiers as before. Note that we also require the asserted `owl:sameAs` relations encoded likewise. Given that all the required information about the equivalence classes (their inlinks, outlinks, derivable equivalences and original identifiers) are gathered under the canonical identifiers, we can apply a straight-forward merge-join on s-o over the sorted stream of data and batch consolidated segments of data.

On a micro level, we buffer each individual consolidated segment into an in-memory index; currently, these segments fit in memory, where for the largest equivalence classes we note that inlinks/outlinks are commonly duplicated—if this were not the case, one could consider using an on-disk index, which should be feasible given that only small batches of the corpus are under analysis at each given time. We assume access to the relevant terminological knowledge required for reasoning, and the predicate-level statistics derived during from the concurrence analysis. We apply scan-reasoning and inconsistency detection over each batch, and for efficiency, skip over batches that are not symptomised by incompatible identifiers.

For equivalence classes containing incompatible identifiers, we first determine the full set of such pairs through application of the inconsistency detection rules:

[44] We note the possibility of a dual correspondence between our "bottom-up" approach to repair and the "top-down" minimal hitting set techniques introduced by Reiter [55].

usually, each detection gives a single pair, where we ignore pairs containing the same identifier (i.e., detections that would equally apply over the unconsolidated data). We check the pairs for a trivial solution: if all identifiers in the equivalence class appear in some pair, we check whether the graph formed by the pairs is strongly connected, in which case, the equivalence class must necessarily be completely disbanded.

For non-trivial repairs, we extract the explicit `owl:sameAs` relations (which we view as directionless) and reinfer `owl:sameAs` relations from the consolidation rules, encoding the subsequent graph. We label edges in the graph with a set of hashes denoting the input triples required for their derivation, such that the cardinality of the hashset corresponds to the primary edge weight. We subsequently use a priority-queue to order the edge-weights, and only materialise concurrence scores in the case of a tie. Nodes in the equivalence graph are merged by combining unique edges and merging the hashsets for overlapping edges. Using these operations, we can apply the aforementioned process to derive the final repaired equivalence classes.

In the final step, we encode the repaired equivalence classes in memory, and perform a final scan of the corpus (in natural sorted order), revising identifiers according to their repaired canonical term.

7.3. *Distributed implementation*

Distribution of the task becomes straight-forward, assuming that the slave machines have knowledge of terminological data, predicate-level statistics, and already have the consolidation encoding sextuples sorted and coordinated by hash on s and o. Note that all of these data are present on the slave machines from previous tasks; for the concurrence analysis, we in fact maintain sextuples during the data preparation phase (although not required by the analysis).

Thus, we are left with two steps:

- **run**: each slave machine performs the above process on its segment of the corpus, applying a merge-join over the data sorted by $(s, p, o, c, s'o')$ and (o, p, s, c, o', s') to derive batches of consolidated data, which are subsequently analysed, diagnosed, and a repair derived in memory;
- **gather/run**: the master machine gathers all repair information from all slave machines, and floods the merged repairs to the slave machines; the slave machines subsequently perform the final repair of the corpus.

7.4. *Performance Evaluation*

We ran the inconsistency detection and disambiguation over the corpora produced by the extended consolidation approach. For the full corpus, the total time taken was 2.35 h. The inconsistency extraction and equivalence class repair analysis took 1.2 h, with a significant average idle time of 7.5 min (9.3%): in particular, certain large batches of consolidated data took significant amounts of time to process, particularly to reason over. The additional expense is due to the relaxation of duplicate detection: we cannot consider duplicates on a triple level, but must consider uniqueness based on the entire sextuple to derive the information required for repair; we must apply many duplicate inferencing steps. On the other hand, we can skip certain reasoning paths that cannot lead to inconsistency. Repairing the corpus took 0.98 h, with an average idle time of 2.7 min.

In Table 17, we again give a detailed breakdown of the timings for the task. With regards the full corpus, note that the aggregation of the repair information took a negligible amount of time, and where only a total of one minute is spent on the slave machine. Most notably, load-balancing is somewhat of an issue, causing slave machines to be idle for, on average, 7.2% of the total task time, mostly waiting for peers. This percentage—and the associated load-balancing issues—would likely be aggrevated further by more machines or a higher scale of data.

With regards the timings of tasks when varying the number of slave machines, as the number of slave machines doubles, the total execution times decrease by factors of 0.534×, 0.599× and 0.649× respectively. Unlike for previous tasks where the aggregation of global knowledge on the master machine poses a bottleneck, this time load-balancing for the inconsistency detection and repair selection task sees overall performance start to converge when increasing machines.

7.5. *Results Evaluation*

It seems that our discussion of inconsistency repair has been somewhat academic: from the total of 2.82 million consolidated batches to check, we found 280 equivalence classes (0.01%)—containing 3,985 identifiers—causing novel inconsistency. Of these 280 inconsistent sets, 185 were detected through disjoint-class constraints, 94 were detected through distinct literal values for inverse-functional properties, and one was detected through `owl:differentFrom` assertions. We list the top five functional-properties given non-distinct lit-

	Category	1		2		4		8		full-8	
		min	**%**	**min**	**%**	**min**	**%**	**min**	**%**	**min**	**%**
	Total execution time	114.7	100.0	61.3	100.0	36.7	100.0	23.8	100.0	141.1	100.0
MASTER	**Executing**	∼	∼	∼	∼	∼	0.1	∼	0.1	∼	∼
MASTER	Miscellaneous	∼	∼	∼	∼	∼	0.1	∼	0.1	∼	∼
MASTER	**Idle (waiting for slaves)**	114.7	100.0	61.2	100.0	36.6	99.9	23.8	99.9	141.1	100.0
SLAVE	**Avg. Executing (total exc. idle)**	114.7	100.0	59.0	96.3	33.6	91.5	22.3	93.7	130.9	92.8
SLAVE	Identify inconsistencies and repairs	74.7	65.1	38.5	62.8	23.2	63.4	17.2	72.2	72.4	51.3
SLAVE	Repair Corpus	40.0	34.8	20.5	33.5	10.3	28.1	5.1	21.5	58.5	41.5
SLAVE	**Avg. Idle**	∼	∼	2.3	3.7	3.1	8.5	1.5	6.3	10.2	7.2
SLAVE	Waiting for peers	0.0	0.0	2.2	3.7	3.1	8.4	1.5	6.2	10.1	7.2
SLAVE	Waiting for master	∼	∼	∼	∼	∼	0.1	∼	0.1	∼	∼

Table 17
Breakdown of timing of distributed disambiguation and repair

#	Functional Property	Detections
1	`foaf:gender`	60
2	`foaf:age`	32
3	`dbo:height`, `dbo:length`	4
4	`dbo:wheelbase`, `dbo:width`	3
5	`atomowl:body`	1
6	`loc:address`, `loc:name`, `loc:phone`	1

Table 18
Breakdown of inconsistency detections for functional-properties, where properties in the same row gave identical detections

#	Disjoint Class 1	Disjoint Class 2	Detections
1	`foaf:Document`	`foaf:Person`	163
2	`foaf:Document`	`foaf:Agent`	153
3	`foaf:Organization`	`foaf:Person`	17
4	`foaf:Person`	`foaf:Project`	3
5	`foaf:Organization`	`foaf:Project`	1

Table 19
Breakdown of inconsistency detections for disjoint-classes

eral values in Table 18 and the top five disjoint classes in Table 19; note that some inconsistent equivalent classes had multiple detections, where, e.g., the class `foaf:Person` is a subclass of `foaf:Agent` and thus an identical detection is given for each. Notably, almost all of the detections involve FOAF classes or properties. (Further, we note that between the time of the crawl and the time of writing, the FOAF vocabulary has removed disjointness constraints between the `foaf:Document` and `foaf:Person`/`foaf:Agent` classes.)

In terms of repairing these 280 equivalence classes, 29 (10.4%) had to be completely disbanded since all identifiers were pairwise incompatible. A total of 905 partitions were created during the repair, with each of the 280 classes being broken up into an average of 3.23 repaired, consistent partitions. Figure 10 gives the distribution of the number of repaired partitions created for each equivalence class; 230 classes were broken into the minimum of two partitions, and one class was broken into 96 partitions. Figure 11 gives the distribution of the number of identfiers in each repaired partition; each partition contained an average of 4.4 equivalent identifiers, with 577 identifiers in singleton partitions, and one partition containing 182 identifiers.

Finally, although the repaired partitions no longer cause inconsistency, this does not necessarily mean that they are *correct*. From the raw equivalence classes identified to cause inconsistency, we applied the same sampling technique as before: we extracted 503 identifiers and for each, randomly sampled an identifier originally thought to be equivalent. We then manually checked whether or not we considered the identifiers to refer to the same entity or not. In the (blind) manual evaluation, we selected option UNCLEAR 19 times, option SAME 232 times (of which, 49 were TRIVIALLY SAME) and option DIFFERENT 249 times. [45] We checked our manually annotated results against the partitions produced by our repair to see if they corresponded. The results are enumerated in Table 20. Of the 481 (clear) manually-inspected pairs, 361 (72.1%) remained equivalent after the repair, and 123 (25.4%) were separated. Of the 223 pairs manually annotated as SAME, 205 (91.9%) also remained equivalent in the repair, whereas 18 (18.1%) were deemed different; here we see that the repair has a 91.9% recall when reesatablishing equivalence for resources that are the same. Of the 261 pairs verified as DIFFERENT, 105 (40.2%) were also deemed different in the repair, whereas 156 (59.7%) were deemed to be equivalent.

Overall, for identifying which resources were the same and should be re-aligned during the repair, the precision was 0.568, with a recall of 0.919, and F_1-measure of 0.702. Conversely, the precision for identi-

[45] We were particularly careful to distinguish information resources—such as WIKIPEDIA articles—and non-information resources as being DIFFERENT.

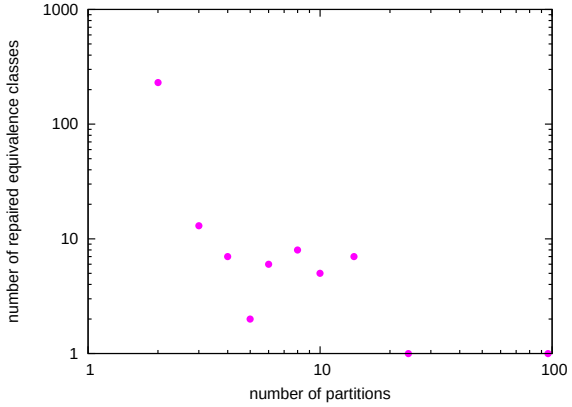


Fig. 10. Distribution of the number of partitions the inconsistent equivalence classes are broken into (log/log)

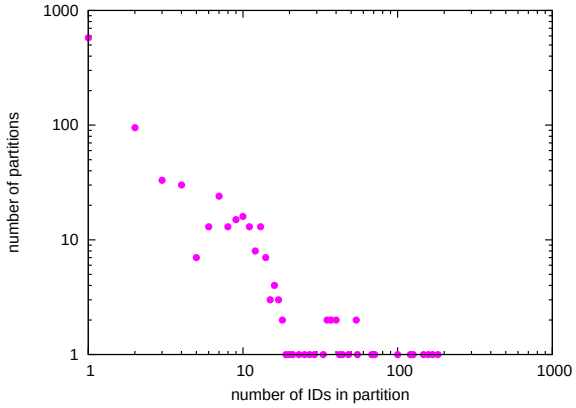


Fig. 11. Distribution of the number of identifiers per repaired partition (log/log)

manual	auto	count	%
SAME	*	223	44.3
DIFFERENT	*	261	51.9
~~UNCLEAR~~	–	~~19~~	~~3.8~~
*	SAME	361	74.6
*	DIFFERENT	123	25.4
SAME	SAME	205	91.9
SAME	DIFFERENT	18	8.1
DIFFERENT	DIFFERENT	105	40.2
DIFFERENT	SAME	156	59.7

Table 20
Results of manual repair inspection

fying which resources were different and should be kept separate during the repair, the precision was 0.854, with a recall of 0.402, and F_1-measure of 0.546.

Interpreting and inspecting the results, we found that the inconsistency-based repair process works well for correctly fixing equivalence classes with one or two "bad apples". However, for equivalence classes which contain many broken resources, we found that the repair process tends towards re-aligning too many identifiers: although the output partitions are consistent, they are not correct. For example, we found that 30 of the pairs which were annotated as different but were re-consolidated by the repair were from the `opera.com` domain, where users had specified various nonsense values which were exported as values for inverse-functional chat-ID properties in FOAF (and were missed by our black-list). This lead to groups of users (the largest containing 205 users) being initially identified as being co-referent through a web of sharing different such values. In these groups, inconsistency was caused by different values for gender (a functional property), and so they were passed into the repair process. However, the repair process simply split the groups into male and female partitions, which although now consistent, were still incorrect. This illustrates a core weakness of the proposed repair process: consistency does not imply correctness.

In summary, for an inconsistency-based detection and repair process to work well over Linked Data, vocabulary publishers would need to provide more, sensible axioms which indicate when instance data are inconsistent. These can then be used to detect more examples of incorrect equivalence (amongst other use-cases [9]). To help repair such cases, in particular, the widespread use and adoption of datatype-properties which are both inverse-functional *and* functional (i.e., one-to-one properties such as `isbn`) would greatly help to automatically identify not only which resources are the same, but to identify which resources are different in a granular manner. [46] Functional properties (e.g., `gender`) or disjoint classes (e.g., `Person`, `Document`) which have wide catchments are useful to detect inconsistencies in the first place, but are not ideal for performing granular repairs.

8. Critical Discussion

In this section, we provide critical discussion of our approach, following the dimensions of the requirements listed at the outset.

With respect to **scale**, on a high level, our primary means of organising the bulk of the corpus is external-sorts, characterised by the linearithmic time complexity $O(n*\log(n))$; external-sorts do not have a critical main-memory requirement, and are efficiently distributable. Our primary means of accessing the data is via linear scans. With respect to the individual tasks:

[46] Of course, in OWL (2) DL, datatype-properties cannot be inverse-functional, but Linked Data vocabularies often break this restriction [29].

– our current baseline consolidation approach relies on an in-memory owl:sameAs index: however we demonstrate an on-disk variant in the extended consolidation approach;
– the extended consolidation currently loads terminological data that is required by all machines into memory: if necessary, we claim that an on-disk terminological index would offer good performance given the distribution of class and property memberships, where we believe that a high cache-hit rate would be enjoyed;
– for the entity concurrence analysis, the predicate level statistics required by all machines is small in volume—for the moment, we do not see this as a serious factor in scaling-up;
– for the inconsistency detection, we identify the same potential issues with respect to terminological data; also, given large equivalence classes with a high number of inlinks and outlinks, we would encounter main-memory problems, where we believe that an on-disk index could be applied assuming a reasonable upper limit on batch sizes.

With respect to **efficiency**:

– the on-disk aggregation of owl:sameAs data for the extended consolidation has proven to be a bottleneck—for efficient processing at higher levels of scale, distribution of this task would be a priority, which should be feasible given that again, the primitive operations involved are external sorts and scans, with non-critical in-memory indices to accelerate reaching the fixpoint;
– although we typically observe terminological data to constitute a small percentage of Linked Data corpora (0.1% in our corpus; cf. [31,33]) at higher scales, aggregating the terminological data for all machines may become a bottleneck, and distributed approaches to perform such would need to be investigated; similarly, as we have seen, large terminological documents can cause load-balancing issues; [47]
– for the concurrence analysis and inconsistency detection, data are distributed according to a modulo-hash function on the subject and object position, where we do not hash on the objects of rdf:type triples; although we demonstrated even data distribution by this approach for our current corpus, this may not hold in the general case;
– as we have already seen for our corpus and machine count, the complexity of repairing consolidated batches may become an issue given large equivalence class sizes;
– there is some notable idle time for our machines, where the total cost of running the pipeline could be reduced by interleaving jobs.

With the exception of our manually derived blacklist for values of (inverse-)functional-properties, the methods presented herein have been entirely **domain-agnostic** and **fully automatic**.

One major open issue is the question of **precision** and **recall**. Given the nature of the tasks—particularly the scale and diversity of the datasets—we believe that deriving an appropriate gold standard is currently infeasible:

– the scale of the corpus precludes manual or semi-automatic processes;
– any automatic process for deriving the gold standard would make redundant the approach to test;
– results derived from application of the methods on subsets of manually verified data would not be equatable to the results derived from the whole corpus;
– even assuming a manual approach were feasible, oftentimes there is no objective criteria for determining what precisely signifies what—the publisher's original intent is often ambiguous.

Thus, we prefer symbolic approaches to consolidation and disambiguation that are predicated on the formal semantics of the data, where we can appeal to the fact that incorrect consolidation is due to erroneous data, not an erroneous approach. Without a formal means of sufficiently evaluating the results, we employ statistical methods for applications where precision is not a primary requirement. We also use formal inconsistency as an indicator of imprecise consolidation, although we have shown that this method does not currently yield many detections. In general, we believe that for the corpora we target, such research can only find it's real litmus test when integrated into a system with a critical user-base.

Finally, we have only briefly discussed issues relating to **web-tolerance**: e.g., spamming or conflicting data. With respect to such consideration, we currently (i) derive and use a blacklist for common void values; (ii) consider authority for terminological data [31,33]; and (iii) try to detect erroneous consolidation through consistency verification. One might question an approach which trusts all equivalences asserted or derived from the data. Along these lines, we track the original pre-consolidation identifiers (in the form of sextuples), which can be used to revert erroneous consolidation. In

[47] We reduce terminological statements on a document-by-document basis according to unaligned blank-node positions: for example, we prune RDF collections identified by blank-nodes that do not join with, e.g., an owl:unionOf axiom.

fact, similar considerations can be applied more generally to the re-use of identifiers across sources: giving special consideration to the consolidation of third party data about an entity is somewhat fallacious without also considering the third party contribution of data using a consistent identifier. In both cases, we track the context of (consolidated) statements, which at least can be used to verify or post-process sources. [48] Currently, the corpus we use probably does not exhibit any significant *deliberate* spamming, but rather *indeliberate* noise. We leave more mature means of handling spamming for future work (as required).

9. Related work

Related Fields

Work relating to entity consolidation has been researched in the area of databases for a number of years, aiming to identify and process co-referent signifiers, with works under the titles of record linkage, record or data fusion, merge-purge, instance fusion, and duplicate identification, and (ironically) a plethora of variations thereupon; see [47,44,13,5,1,2], etc., and surveys at [19,8]. Unlike our approach, which leverages the declarative semantics of the data in order to be domain agnostic, such systems usually operate given closed schemas—similarly, they typically focus on string-similarity measures and statistical analysis. Haas et al. note that "*in relational systems where the data model does not provide primitives for making same-as assertions <...> there is a value-based notion of identity*" [25]. However, we note that some works have focused on leveraging semantics for such tasks in relation databases; e.g., Fan et al. [21] leverage domain knowledge to match entities, where interestingly they state "*real life data is typically dirty...* <thus> *it is often necessary to hinge on the semantics of the data*".

Some other works—more related to Information Retrieval and Natural Language Processing—focus on extracting coreferent entity names from unstructured text, tying in with aligning the results of Named Entity Recognition where for example, Singh et al. [60] present an approach to identify coreferences from a corpus of 3 million natural language "mentions" of persons, where they build compound "entities" out of the individual mentions.

Instance Matching

With respect to RDF, one area of research also goes by the name *instance matching*: for example, in 2009, the Ontology Alignment Evaluation Initiative [49] introduced a new test track on instance matching [50]. We refer the interested reader to the results of OAEI 2010 [20] for further information, where we now discuss some recent instance matching systems. It is worth noting that in contrast to our scenario, many instance matching systems take as input a small number of instance sets—i.e., consistently named datasets—across which similarities and coreferences are computed. Thus, the instance matching systems mentioned in this section are not specifically tailored for processing Linked Data (and most do not present experiments along these lines).

The LN2R [56] system incorporates two methods for performing instance matching: (i) L2R applies deductive techniques where rules are used to model the semantics of the respective ontology—particularly functional properties, inverse-functional properties and disjointness constraints—which are then used to infer consistent coreference matches; (ii) N2R is an inductive, numeric approach which uses the results of L2R to perform unsupervised matching, where a system of linear-equations representing similarities is seeded using text-based analysis and then used to compute instance similarity by iterative resolution. Scalability is not directly addressed.

The MOMA [68] engine matches instances from two distinct datasets; to help ensure high precision, only members of the same class are matched (which can be determined, perhaps, by an ontology-matching phase). A suite of matching tools is provided by MOMA, along with the ability to pipeline matchers, and to select a variety of operators for merging, composing and selecting (thresholding) results. Along these lines, the system is designed to facilitate a human expert in instance-matching, focusing on a different scenario to ours presented herein.

The RiMoM [41] engine is designed for ontology matching, implementing a wide range of strategies which can be composed and combined in a graphical user interface; strategies can also be selected by the engine itself based on the nature of the input. Matching results from different strategies can be composed using linear interpolation. Instance matching techniques mainly rely on text-based analysis of resources using Vector Space Models and IR-style measures of similarity and relevance. Large scale instance matching is

[48] Although it must be said, we currently do not track the steps used to derive the equivalences involved in consolidation, which would be expensive to materialise and maintain.

[49] OAEI. http://oaei.ontologymatching.org/

[50] Instance data matching. http://www.instancematching.org/

enabled by an inverted-index over text terms, similar in principle to candidate reduction techniques discussed later in this section. They currently do not support symbolic techniques for matching (although they do allow disambiguation of instances from different classes).

Nikolov et al. [48] present the KnoFuss architecture for aligning data on an assertional level; they identify a three phase process involving coreferencing (finding equivalent individuals), conflict detection (finding inconsistencies caused by the integration), and inconsistency resolution. For the coreferencing, the authors introduce and discuss approaches incorporating string similarity measures and class-based machine learning techniques. Although the high-level process is similar to our own, the authors do not address scalability concerns.

In motivating the design of their RDF-AI system, Scharffe et al. [58] identify four steps in aligning datasets: *align*, *interlink*, *fuse* and *post-process*. The align process identifies equivalences between entities in the two datasets, the interlink process materialises `owl:sameAs` relations between the two datasets, the aligning step merges the two datasets (on both a terminological and assertional level, possibly using domain-specific rules), and the *post-processing* phase subsequently checks the consistency of the output data. Although parts of this conceptual process echoes our own, the RDF-AI system itself differs greatly from our work. First, the authors focus on the task of identifying coreference *across* two distinct datasets, whereas we aim to identify coreference in a large "bag of instances". Second, the authors do not emphasise scalability, where the RDF datasets are loaded into memory and processed using the popular Jena framework; similarly, the system proposes performing using pair-wise comparison for, e.g., using string matching techniques (which we do not support). Third, although inconsistency detection is mentioned in the conceptual process, the authors do not discuss implementation details for the RDF-AI system itself.

Noessner et al. [49] present an approach for aligning two A-Boxes described using the same T-Box; in particular they leverage similarity measures introduced by Stuckenschmidt [65], and define an optimisation problem to identify the alignment that generates the highest weighted similarity between the two A-Boxes under analysis: they use Integer Linear Programming to generate the optimal alignment, encoding linear constraints to enforce *valid* (i.e., consistency preserving), one-to-one, functional mappings. Although they give performance results, they do not directly address scalability. Their method for comparing entities is similar in practice to ours: they measure the "overlapping knowledge" between two entities, counting how many assertions are true about both. The goal is to match entities such that: the resulting consolidation is consistent; the measure of overlap is maximal.

Like us, Castano et al. [12] approach instance matching from two distinct perspectives: (i) determine coreferent identifiers; (ii) detect similar individuals based on the data they share. Much of their work is similar in principle to ours: in particular, they use reasoning for identifying equivalences and use a statistical approach for identifying properties "with high identification power". They do not consider use of inconsistency detection for disambiguating entities, and perhaps more critically, only evaluate with respect to a dataset containing $\sim$15 thousand entities.

Cudré-Mauroux et al. [17] present the idMesh system, which leverages user-defined associations and probabalistic methods to derive entity-level relationships, including resolution of conflicts; they also delineate entities based on "temporal discrimination", whereby coreferent entities may predate or postdate one another, capturing a description thereof at a particular point in time. The idMesh system itself is designed over a peer-to-peer network with centralised coordination. However, evaluation is over synthetic data, where they only demonstrate a maximum scale involving 8,000 entities and 24,000 links, over 400 machines: the evaluation of performance focuses on network traffic and message exchange as opposed to time.

Domain-Specific Consolidation

Various authors have looked at applying consolidation over domain-specific RDF corpora: e.g., Sleeman and Finin look at using machine learning techniques to consolidate FOAF personal profile information [61]; Shi et al. similarly look at FOAF-specific alignment techniques [59] using inverse-functional properties and fuzzy string matching; Jentzsch et al. examine alignment of published drug data [38];[51] Raimond et al. look at interlinking RDF from the music-domain [54]; Monaghan and O' Sullivan apply consolidation to photo annotations expressed in RDF [46].

Salvadores et al. [57] present the LinksB2N system, which aims to perform scalable integration of RDF data, particularly focusing on evaluation over corpora from the marketing domain; however, their methods are not specific to this domain. They do not leverage the se-

[51] In fact, we believe that this work generates the incorrect results observable in Table 6; cf. `http://groups.google.com/group/pedantic-web/browse_thread/thread/ad740f7052cc3a2d`.

mantics of the data for performing consolidation, instead using similarity measures based on the idea that "the unique combination of RDF predicates associated with RDF resources is what defines their existence as unique" [57]. This is a similar intuition to that behind our concurrence analysis, but we again question the validity of such an assumption for consolidation, particularly given incomplete data and the Open World Assumption underlying RDF(S)/OWL—we view an RDF resource as a description of something signified, and would wish to avoid conflating unique signifiers, even if they match *precisely* with respect to their description.

Large-Scale/Web-Data Consolidation

Like us, various authors have investigated consolidation techniques tailored to the challenges—esp. scalability and noise—of resolving coreference in heterogeneous RDF Web data.

On a larger scale and following a similar approach to us, Hu et al. [36,35] have recently investigated a consolidation system—called ObjectCoRef—for application over ∼500 million triples of Web data. They too define a baseline consolidation (which they call the *kernel*) built by applying reasoning on `owl:sameAs`, `owl:InverseFunctionalProperty`, `owl:FunctionalProperty` and `owl:cardinality`. No other OWL constructs or reasoning is used when building the kernel. Then, starting from the coreferents in the kernel, they use a learning technique to find the most discriminative properties. Further, they reuse the notion of average (inverse) cardinality [34] to find frequent property combination pairs. A small scale experiment is conducted that shows good results. At large scale, although they manage to build the kernel for the ∼500 million triples dataset, they only apply and evaluate their statistical method on a very small subset, confessing that the approach would not be feasible for the complete set.

The Sindice system has historically used inverse-functional properties to find equivalent identifiers, also investigating some bespoke "schema-level" reasoning to identify a wider range of such properties [50]. However, Sindice no longer uses such OWL features for determining coreference of entities. [52] The related Sig.ma search system [69] internally uses IR-style string-based measures (e.g., TF-IDF scores) to mashup results in their engine; compared to our system, they do not prioritise precision, where mashups are designed to have a high recall of related data, and to be disambiguated manually by the user using the interface. This can be verified anecdotally by searching for various ambiguous terms in their public interface. [53]

Bishop et al. [6] apply reasoning over 1.2 billion Linked Data triples using the BigOWLIM reasoner; however, this dataset is manually selected as a merge of a number of smaller, known datasets as opposed to an arbitrary corpus. They discuss well-known optimisations similar to our canonicalisation as a necessary means of avoiding the quadratic nature of traditional replacement semantics for `owl:sameAs`.

Large-Scale/Distributed Consolidation

With respect to distributed consolidation, Urbani et al. [70] propose the WebPie system, which uses MapReduce to perform pD* reasoning [67] using a cluster of commodity hardware similar to ourselves. The pD* ruleset contains rules for handling `owl:sameAs` replacement, inverse-functional properties and functional-properties, but not for cardinalities (which, in any case we demonstrated to be ineffective over out corpus). The authors also discuss a canonicalisation approach for handling equivalent identifiers. They demonstrate their methods over 100 billion triples of synthetic LUBM data over 64 machines; however, they do not present evaluation over Linked Data, do not perform any form of similarity or concurrence measures, do not consider inconsistency detection (not given by the pD* ruleset, or by their corpora) and generally have a somewhat different focus: scalable distributed rule-based materialisation.

Candidate Reduction

For performing large-scale consolidation, one approach is to group together sets of candidates within which there are likely to be coreferent identifiers. Such candidate reduction then by-passes the need for complete pair-wise comparison and can enable the use of more expensive matching methods in closed regions. This is particularly relevant for text-based analyses which we have not tackled in this paper. However, the principle of candidate reduction generalises to any form of matching that cannot be performed in a complete pair-wise manner.

Song and Heflin [62] investigate scalable methods to prune the candidate-space for the instance matching task, focusing on scenarios such as our own. First, discriminating properties—i.e., those for which a typical value is shared by few instances—are used to group initial candidate sets. Second, the authors propose building

[52] From personal communication with the Sindice team

[53] `http://sig.ma/search?q=paris`

textual descriptions of instances from surrounding text and using an IR-style inverted index to query and group lexically similar instances. Evaluation demonstrates feasibility for matching up-to one million instances, although the authors claim that the approach can scale further.

Ioannou et al. [37] also propose a system, called RDFSim, to cluster similar candidates based on associated textual information. Virtual prose documents are constructed for resources from surrounding text, which are then encoded in a special index which using Locality Sensitive Hashing (LSH). The core idea of LSH is to hash similar virtual documents into regions of space, where distance measures can be used to identify similar resources. A set of hash functions can be employed for these purposes, and "close" documents are then subject to other similarity measures; the RDFSim system currently uses the Jaccard coefficient to compute textual similarities for these purposes.

Naming/Linking Resources on the Web

A number of works have investigated the fundamentals of resource naming and linking on the Web, proposing schemes or methods to bootstrap agreement between remote publishers on common resource URIs, or to enable publishers to better link their contribution with other external datasets.

With respect to URI naming on the Web, Bouquet et al. [10] propose OKKAM: a centralised naming architecture for minting URI signifiers for the Web. The OKKAM system thus supports entity lookups for publishers, where a search providing a description of the entity in question returns a set of possible candidates. Publishers can then re-use the URI for the candidate which most closely matches their intention in their local data, with the intuition that other publishers have done the same; thus, all users of the system will have their naming schemes aligned. However, we see such a centralised "'naming authority" as going against the ad-hoc, decentralised, scale-free nature of the Web. More concretely, for example, having one URI per resource goes against current Linked Data principles which encourage use of dereferenceable URIs: one URI can only dereference to one document, so which document should that be, and who should decide?

Online systems RKBExplorer [23,22][54], <sameAs>[55] and ObjectCoref [15][56] offer on-demand querying for `owl:sameAs` relations found for a given input URI, which they internally compute and store; the former focus on publishing `owl:sameAs` relations for authors and papers in the area of scientific publishing, with the latter two systems offering more general `owl:sameAs` relationships between Linked Data identifiers. In fact, many of the `owl:sameAs` relations we consume are published as Linked Data by the RKBExplorer system.

Volz et al. [72] present the Silk framework for creating and maintaining inter-linkage between domain-specific RDF datasets; in particular, this framework provides publishers with a means of discovering and creating `owl:sameAs` links between data sources using domain-specific rules and parameters. Thereafter, publishers can integrate discovered links into their exports, enabling better linkage of the data and subsequent consolidation by data consumers: this framework goes hand-in-hand with our approach, producing the `owl:sameAs` relations which we consume.

Popitsch and Haslhofer present discussion on the problem of broken links in Linked Data, identifying structurally broken links (the Web of Data's version of a "deadlink") and semantically broken links, where the original meaning of an identifier changes after a link has been remotely asserted [52]. The authors subsequently present the DSNotify system, which monitors dynamicity over a given subset of Linked Data and can detect and act upon changes—e.g., to notify another agent or correct broken links—and can also be used to indirectly link the dynamic target.

Analyses of `owl:sameAs`

Recent papers have analysed and discussed the use of `owl:sameAs` on the Web. These papers have lead to debate on whether `owl:sameAs` is appropriate for modelling coreference across the Web.

Halpin et al. [26] look at the semantics and current usage of `owl:sameAs` in Linked Data, discussing issues relating to *identity*, and providing four categories of `owl:sameAs` usage to relate entities that are closely related, but for which the semantics of `owl:sameAs`—particularly substitution—does not quite hold. The authors also take and manually inspect 500 `owl:sameAs` links sampled (using logarithmic weights for each domain) from Web data. Their experiments suggest that although the majority of `owl:sameAs` relations are considered correct, many were still incorrect, and disagreement between the judges indicates that the quality of specific `owl:sameAs` links can be subjective [26]. In fact, their results are much more pessimistic than ours with regards the quality of `owl:sameAs` on the Web: whereas

[54] http://www.rkbexplorer.com/sameAs/
[55] http://sameas.org/
[56] http://ws.nju.edu.cn/objectcoref/

we established a precision figure of 97.2% for explicit owl:sameAs through manual inspection, Halpin et al. put the figure at 51% (±21%). Even taking the most optimistic figure of 72% from Halpin's paper, there is a wide gap to our result. Possible reasons for the discrepancy include variations in sampling techniques, as well as different criteria for manual judging. [57].

Ding et al. [18] also provide quantitative analysis of owl:sameAs usage in the BTC-2010 dataset; some of these results correspond with analogous measures we have presented in Section 4.4, but for a different (albeit similar) sampled corpus. They found that URIs with at least one coreferent identifier had an average of 2.4 identifiers (this corresponds closely with our measurement of 2.65 identifiers per equivalence-class for baseline consolidation). The average path length of owl:sameAs was 1.07, indicating that few transitive links are given. The largest equivalence class found was 5 thousand (versus 8,481 in our case). They also discuss the landscape of owl:sameAs linkage between different publishers of Linked Data.

10. Conclusion

In this paper, we have provided comprehensive discussion on scalable and distributed methods for consolidating, matching, and disambiguating entities present in a large static Linked Data corpus. Throughout, we have focused on the scalability and practicalities of applying our methods over real, arbitrary Linked Data in a domain agnostic and (almost entirely) automatic fashion. We have shown how to use explicit owl:sameAs relations in the data to perform consolidation, and subsequently expanded this approach, leveraging the declarative formal semantics of the corpus to materialise additional owl:sameAs relations. We also presented a scalable approach to identify weighted concurrence relations for entities that share many inlinks, outlinks, and attribute values; we note that many of those entities demonstrating the highest concurrence were *not* coreferent. Next, we presented an approach using inconsistencies to disambiguate entities and subsequently repair equivalence classes: we found that this approach currently derives few diagnoses, where the granularity of inconsistencies within Linked Data is not sufficient for accurately pinpointing all incorrect consolidation. Finally, we tempered our contribution with critical discussion, particularly focusing on scalability and efficiency concerns.

[57] Again, our inspection results are available at http://swse.deri.org/entity/

We believe that the area of research touched upon in this paper—particularly as applied to large scale Linked Data corpora—is of particular significance given the rapid growth in popularity of Linked Data publishing. As the scale and diversity of the Web of Data expands, scalable and precise data integration technique will become of vital importance, particularly for data warehousing applications; we see the work presented herein as a significant step in the right direction.

Acknowledgements We would like to thank Andreas Harth for involvement in early works. We would also like to thank the anonymous reviewers and the editors for their time and valuable comments. *The work presented in this paper has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2), and by an IRCSET postgraduate scholarship.*

References

[1] Riccardo Albertoni and Monica De Martino. Semantic Similarity of Ontology Instances Tailored on the Application Context. In *Proc. of OTM 2006, Part I*, volume 4275 of *LNCS*, pages 1020–1038. Springer, 2006.

[2] Riccardo Albertoni and Monica De Martino. Asymmetric and Context-Dependent Semantic Similarity among Ontology Instances. *Jour. on Data Semantics*, 4900(10):1–30, 2008.

[3] David Beckett and Tim Berners-Lee. Turtle – Terse RDF Triple Language. W3C Team Submission, January 2008. http://www.w3.org/TeamSubmission/turtle/.

[4] Tim Berners-Lee. Linked Data. Design issues for the World Wide Web, World Wide Web Consortium, 2006. http://www.w3.org/DesignIssues/LinkedData.html.

[5] Philip A. Bernstein, Sergey Melnik, and Peter Mork. Interactive Schema Translation with Instance-Level Mappings. In *Proc. of VLDB 2005*, pages 1283–1286. ACM Press, 2005.

[6] Barry Bishop, Atanas Kiryakov, Damyan Ognyanov, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. Factforge: A fast track to the web of data. *Semantic Web*, 2(2):157–166, 2011.

[7] Christian Bizer, Richard Cyganiak, and Tom Heath. How to Publish Linked Data on the Web. linkeddata.org Tutorial, July 2008. http://linkeddata.org/docs/how-to-publish.

[8] Jens Bleiholder and Felix Naumann. Data fusion. *ACM Comput. Surv.*, 41(1), 2008.

[9] Piero A. Bonatti, Aidan Hogan, Axel Polleres, and Luigi Sauro. Robust and scalable linked data reasoning incorporating provenance and trust annotations. *J. Web Sem.*, 9(2):165–201, 2011.

[10] Paolo Bouquet, Heiko Stoermer, Michele Mancioppi, and Daniel Giacomuzzi. OkkaM: Towards a solution to the "identity crisis" on the semantic web. In *Proceedings of SWAP 2006, the 3rd Italian Semantic Web Workshop*, volume 201 of *CEUR Workshop Proceedings*, December 2006.

[11] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, February 2004. http://www.w3.org/TR/rdf-schema/.

[12] Silvana Castano, Alfio Ferrara, Stefano Montanelli, and Davide Lorusso. Instance Matching for Ontology Population. In

Salvatore Gaglio, Ignazio Infantino, and Domenico Saccà, editors, *SEBD*, pages 121–132, 2008.

[13] Zhaoqi Chen, Dmitri V. Kalashnikov, and Sharad Mehrotra. Exploiting relationships for object consolidation. In *IQIS '05: Proceedings of the 2nd international workshop on Information quality in information systems*, pages 47–58, New York, NY, USA, 2005. ACM Press.

[14] Gong Cheng, Weiyi Ge, Honghan Wu, and Yuzhong Qu. Searching Semantic Web Objects Based on Class Hierarchies. In *Proceedings of Linked Data on the Web Workshop*, 2008.

[15] Gong Cheng and Yuzhong Qu. Searching Linked Objects with Falcons: Approach, Implementation and Evaluation. *Int. J. Semantic Web Inf. Syst.*, 5(3):49–70, 2009.

[16] Martine De Cock and Etienne E. Kerre. On (un)suitable fuzzy relations to model approximate equality. *Fuzzy Sets and Systems*, 133(2):137–153, 2003.

[17] Philippe Cudré-Mauroux, Parisa Haghani, Michael Jost, Karl Aberer, and Hermann de Meer. idMesh: Graph-Based Disambiguation of Linked Data. In *WWW*, pages 591–600, 2009.

[18] Li Ding, Joshua Shinavier, Zhenning Shangguan, and Deborah L. McGuinness. SameAs networks and beyond: analyzing deployment status and implications of owl:sameAs in linked data. In *Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I*, ISWC'10, pages 145–160, Berlin, Heidelberg, 2010. Springer-Verlag.

[19] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[20] Jérôme Euzenat, Alfio Ferrara, Christian Meilicke, Andriy Nikolov, Juan Pane, François Scharffe, Pavel Shvaiko, Heiner Stuckenschmidt, Ondrej Sváb-Zamazal, Vojtech Svátek, and Cássia Trojahn. Results of the ontology alignment evaluation initiative 2010. In *OM*, 2010.

[21] Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. Reasoning about Record Matching Rules. *PVLDB*, 2(1):407–418, 2009.

[22] Hugh Glaser, Afraz Jaffri, and Ian Millard. Managing Co-reference on the Semantic Web. In *Proc. of LDOW 2009*, 2009.

[23] Hugh Glaser, Ian Millard, and Afraz Jaffri. `RKBExplorer.com`: A Knowledge Driven Infrastructure for Linked Data Providers. In *ESWC Demo*, Lecture Notes in Computer Science, pages 797–801. Springer, June 2008.

[24] Bernardo Cuenca Grau, Boris Motik, Zhe Wu, Achille Fokoue, and Carsten Lutz (eds.). OWL 2 Web Ontology Language: Profiles. W3C Working Draft, April 2008. `http://www.w3.org/TR/owl2-profiles/`.

[25] Laura M. Haas, Martin Hentschel, Donald Kossmann, and Renée J. Miller. Schema AND Data: A Holistic Approach to Mapping, Resolution and Fusion in Information Integration. In *ER*, pages 27–40, 2009.

[26] Harry Halpin, Patrick J. Hayes, James P. McCusker, Deborah L. McGuinness, and Henry S. Thompson. When owl:sameAs Isn't the Same: An Analysis of Identity in Linked Data. In *International Semantic Web Conference (1)*, pages 305–320, 2010.

[27] Harry Halpin, Ivan Herman, and Pat Hayes. When owl:sameAs isn't the Same: An Analysis of Identity Links on the Semantic Web. In *Linked Data on the Web WWW2010 Workshop (LDOW2010)*, 2010.

[28] Patrick Hayes. RDF Semantics. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-mt/`.

[29] Aidan Hogan. *Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora*. PhD thesis, Digital Enterprise Research Institute, National University of Ireland, Galway, 2011. Available from `http://aidanhogan.com/docs/thesis/`.

[30] Aidan Hogan, Andreas Harth, and Stefan Decker. Performing Object Consolidation on the Semantic Web Data Graph. In *1st I3 Workshop: Identity, Identifiers, Identification Workshop*, 2007.

[31] Aidan Hogan, Andreas Harth, and Axel Polleres. Scalable Authoritative OWL Reasoning for the Web. *Int. J. Semantic Web Inf. Syst.*, 5(2):49–90, 2009.

[32] Aidan Hogan, Andreas Harth, Jürgen Umbrich, Sheila Kinsella, Axel Polleres, and Stefan Decker. Searching and browsing Linked Data with SWSE: The Semantic Web Search Engine. *Journal of Web Semantics*, 2011. (In Press).

[33] Aidan Hogan, Jeff Z. Pan, Axel Polleres, and Stefan Decker. SAOR: Template Rule Optimisations for Distributed Reasoning over 1 Billion Linked Data Triples. In *International Semantic Web Conference*, 2010.

[34] Aidan Hogan, Axel Polleres, Jürgen Umbrich, and Antoine Zimmermann. Some entities are more equal than others: statistical methods to consolidate Linked Data. In *4th International Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic (NeFoRS2010)*, 2010.

[35] Wei Hu, Jianfeng Chen, and Yuzhong Qu. A self-training approach for resolving object coreference on the semantic web. In *WWW*, pages 87–96, 2011.

[36] Wei Hu, Yuzhong Qu, and Xingzhi Sun. Bootstrapping object coreferencing on the semantic web. *J. Comput. Sci. Technol.*, 26(4):663–675, 2011.

[37] Ekaterini Ioannou, Odysseas Papapetrou, Dimitrios Skoutas, and Wolfgang Nejdl. Efficient semantic-aware detection of near duplicate resources. In *ESWC (2)*, pages 136–150, 2010.

[38] Anja Jentzsch, Jun Zhao, Oktie Hassanzadeh, Kei-Hoi Cheung, Matthias Samwald, and Bo Andersson. Linking Open Drug Data. In *International Conference on Semantic Systems (I-SEMANTICS'09)*, 2009.

[39] Frank Klawonn. Should fuzzy equality and similarity satisfy transitivity? Comments on the paper by M. De Cock and E. Kerre. *Fuzzy Sets and Systems*, 133(2):175–180, 2003.

[40] Vladimir Kolovski, Zhe Wu, and George Eadon. Optimizing Enterprise-scale OWL 2 RL Reasoning in a Relational Database System. In *International Semantic Web Conference*, 2010.

[41] Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. Rimom: A dynamic multistrategy ontology alignment framework. *IEEE Trans. Knowl. Data Eng.*, 21(8):1218–1232, 2009.

[42] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, February 2004. `http://www.w3.org/TR/owl-features/`.

[43] Georgios Meditskos and Nick Bassiliades. DLEJena: A practical forward-chaining OWL 2 RL reasoner combining Jena and Pellet. *J. Web Sem.*, 8(1):89–94, 2010.

[44] Martin Michalowski, Snehal Thakkar, and Craig A. Knoblock. Exploiting secondary sources for automatic object consolidation. In *Proceeding of 2003 KDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.

[45] Alistair Miles, Thomas Baker, and Ralph Swick. Best Practice Recipes for Publishing RDF Vocabularies, March 2006. `http://www.w3.org/TR/2006/WD-swbp-vocab-pub-20060314/1`. Superceded by Berrueta and Phipps: `http://www.w3.org/TR/swbp-vocab-pub/`.

[46] Fergal Monaghan and David O'Sullivan. Leveraging Ontologies, Context and Social Networks to Automate Photo Annotation. In *SAMT*, pages 252–255, 2007.

[47] H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic Linkage of Vital Records: Computers can be used to extract "follow-up" statistics of families from files of routine records. *Science*, 130:954–959, October 1959.

[48] Andriy Nikolov, Victoria S. Uren, Enrico Motta, and Anne N. De Roeck. Integration of Semantically Annotated Data by the KnoFuss Architecture. In *EKAW*, pages 265–274, 2008.

[49] Jan Noessner, Mathias Niepert, Christian Meilicke, and Heiner Stuckenschmidt. Leveraging Terminological Structure for Object Reconciliation. In *ESWC (2)*, pages 334–348, 2010.

[50] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice.com: A document-oriented lookup index for open linked data. *Int. J. Metadata Semant. Ontologies*, 3(1):37–52, 2008.

[51] Eyal Oren, Spyros Kotoulas, George Anadiotis, Ronny Siebes, Annette ten Teije, and Frank van Harmelen. Marvin: Distributed reasoning over large-scale Semantic Web data. *J. Web Sem.*, 7(4):305–316, 2009.

[52] Niko Popitsch and Bernhard Haslhofer. DSNotify: handling broken links in the web of data. In *WWW*, pages 761–770, 2010.

[53] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, January 2008. `http://www.w3.org/TR/rdf-sparql-query/`.

[54] Yves Raimond, Christopher Sutton, and Mark B. Sandler. Interlinking Music-Related Data on the Web. *IEEE MultiMedia*, 16(2):52–63, 2009.

[55] Raymond Reiter. A Theory of Diagnosis from First Principles. *Artif. Intell.*, 32(1):57–95, 1987.

[56] Fatiha Saïs, Nathalie Pernelle, and Marie-Christine Rousset. Combining a logical and a numerical method for data reconciliation. *J. Data Semantics*, 12:66–94, 2009.

[57] Manuel Salvadores, Gianluca Correndo, Bene Rodriguez-Castro, Nicholas Gibbins, John Darlington, and Nigel R. Shadbolt. LinksB2N: Automatic Data Integration for the Semantic Web. In *OTM Conferences (2)*, pages 1121–1138, 2009.

[58] F. Scharffe, Y. Liu, and C. Zhou. RDF-AI: an Architecture for RDF Datasets Matching, Fusion and Interlink. In *IJCAI 2009 Workshop on Identity, Reference, and Knowledge Representation (IR-KR)*, 2009.

[59] Lian Shi, Diego Berrueta, Sergio Fernández, Luis Polo, and Silvino Fernández. Smushing RDF instances: are Alice and Bob the same open source developer? In *PICKME Workshop*, 2008.

[60] Sameer Singh, Michael L. Wick, and Andrew McCallum. Distantly Labeling Data for Large Scale Cross-Document Coreference. *CoRR*, abs/1005.4298, 2010.

[61] Jennifer Sleeman and Tim Finin. Learning Co-reference Relations for FOAF Instances. In *Poster and Demo Session at ISWC*, 2010.

[62] Dezhao Song and Jeff Heflin. Automatically generating data linkages using a domain-independent candidate selection approach. In *International Semantic Web Conference*, 2011.

[63] Mechthild Stoer and Frank Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997.

[64] Michael Stonebraker. The Case for Shared Nothing. *IEEE Database Eng. Bull.*, 9(1):4–9, 1986.

[65] Heiner Stuckenschmidt. A Semantic Similarity Measure for Ontology-Based Information. In *FQAS '09: Proceedings of the 8th International Conference on Flexible Query Answering Systems*, pages 406–417, Berlin, Heidelberg, 2009. Springer-Verlag.

[66] Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. Syst. Sci.*, 18(2):110–127, 1979.

[67] Herman J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, 3:79–115, 2005.

[68] Andreas Thor and Erhard Rahm. Moma – a mapping-based object matching system. In *CIDR*, pages 247–258, 2007.

[69] Giovanni Tummarello, Richard Cyganiak, Michele Catasta, Szymon Danielczyk, and Stefan Decker. Sig.ma: Live views on the Web of Data. In *Semantic Web Challenge (ISWC2009)*, 2009.

[70] Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri E. Bal. OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples. In *ESWC (1)*, pages 213–227, 2010.

[71] Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Scalable Distributed Reasoning Using MapReduce. In *International Semantic Web Conference*, pages 634–649, 2009.

[72] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and Maintaining Links on the Web of Data. In *International Semantic Web Conference*, pages 650–665, 2009.

[73] Denny Vrandečíc, Markus Krötzsch, Sebastian Rudolph, and Uta Lösch. Leveraging Non-Lexical Knowledge for the Linked Open Data Web. *Review of April Fool's day Transactions (RAFT)*, 5:18–27, 2010.

[74] Jesse Weaver and James A. Hendler. Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples. In *International Semantic Web Conference (ISWC2009)*, pages 682–697, 2009.

[75] Lotfi A. Zadeh, George J. Klir, and Bo Yuan. *Fuzzy Sets, Fuzzy Logic, Fuzzy Systems*. World Scientific Press, 1996.

Appendix A. Prefixes

Table A.1 lists the prefixes used throughout the paper.

Prefix	URI
"T-Box prefixes"	
atomowl:	http://bblfish.net/work/atom-owl/2006-06-06/#
b2r:	http://bio2rdf.org/bio2rdf:
b2rr:	http://bio2rdf.org/bio2rdf_resource:
dbo:	http://dbpedia.org/ontology/
dbp:	http://dbpedia.org/property/
ecs:	http://rdf.ecs.soton.ac.uk/ontology/ecs#
eurostat:	http://ontologycentral.com/2009/01/eurostat/ns#
fb:	http://rdf.freebase.com/ns/
foaf:	http://xmlns.com/foaf/0.1/
geonames:	http://www.geonames.org/ontology#
kwa:	http://knowledgeweb.semanticweb.org/heterogeneity/alignment#
lldpubmed:	http://linkedlifedata.com/resource/pubmed/
loc:	http://sw.deri.org/2006/07/location/loc#
mo:	http://purl.org/ontology/mo/
mvcb:	http://webns.net/mvcb/
opiumfield:	http://rdf.opiumfield.com/lastfm/spec#
owl:	http://www.w3.org/2002/07/owl#
quaffing:	http://purl.org/net/schemas/quaffing/
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
skipinions:	http://skipforward.net/skipforward/page/seeder/skipinions/
skos:	http://www.w3.org/2004/02/skos/core#
"A-Box prefixes"	
avtimbl:	http://www.advogato.org/person/timbl/foaf.rdf#
dbpedia:	http://dbpedia.org/resource/
dblpperson:	http://www4.wiwiss.fu-berlin.de/dblp/resource/person/
eswc2006p:	http://www.eswc2006.org/people/#
identicau:	http://identi.ca/user/
kingdoms:	http://lod.geospecies.org/kingdoms/
macs:	http://stitch.cs.vu.nl/alignments/macs/
semweborg:	http://data.semanticweb.org/organization/
swid:	http://semanticweb.org/id/
timblfoaf:	http://www.w3.org/People/Berners-Lee/card#
vperson:	http://virtuoso.openlinksw.com/dataspace/person/
wikier:	http://www.wikier.org/foaf.rdf#
yagor:	http://www.mpii.de/yago/resource/

Table A.1
Used prefixes