Discovering Domain-Specific Public SPARQL Endpoints: A Life-Sciences Use-Case

Muntazir Mehdi*†, Aftab Iqbal†, Aidan Hogan‡, Ali Hasnain†, Yasar Khan†, Stefan Decker†, and Ratnesh Sahay†

*Department of Computer Science
Technical University of Kaiserslautern
m_mehdi10@cs.uni-kl.de
Germany

†INSIGHT Centre for Data Analytics
National University of Ireland, Galway
{first.last}@insight-centre.org
Ireland

‡Department of Computer Science
Universidad de Chile
ahogan@dcc.uchile.cl
Chile

ABSTRACT

A significant portion of the LOD cloud consists of Life Sciences data sets, which together contain billions of clinical facts that interlink to form a "Web of Clinical Data". However, tools for new publishers to find relevant datasets that could potentially be linked to are missing, particularly in specialist domain-specific settings. Based on a set of domain-specific keywords extracted from a local dataset, this paper proposes methods to automatically identify relevant public SPARQL endpoints from a list of candidates.

Keywords

Linked Open Data (LOD) Cloud, Web of Data, SPARQL, Healthcare and Life Sciences

1. INTRODUCTION

Over the past several years, a variety of publishers – coming from academia, governmental organisations, online communities and companies alike – have begun exposing their corpora on the Web as Linked Data. The Linking Open Data (LOD) Cloud[1] provides an overview of 295 Linked Datasets, which, according to publisher statistics[2], incorporate over 30 billion facts. Of these, 41 datasets relate to Life Sciences, incorporating 3 billion facts. With regards to accessing this content, aside from crawling the raw data, 68% of the LOD datasets offer a link to at least one SPARQL endpoint that can be used to directly query the dataset. The Datahub catalogue[3] lists at least 427 such SPARQL endpoints available on the Web (though indeed some are offline or unreliable [1]).

The LOD Cloud also contains 500 million cross-dataset links (191 million links specifically in the life-sciences domain)[4], following the fourth Linked Data principle: "*link to related data*". From the perspective of a consumer, these links allow for recursively discovering and navigating detailed information about related entities elsewhere on the Web. From the perspective of a publisher, links encourage modularity, where high-quality links (once in place) can reduce the amount of content they need to host: for example, instead of each publisher redundantly providing a basic description of all the countries they mention in their data, each publisher can link to a detailed description for each country in a legacy dataset elsewhere (such as GeoNames or DBpedia). Likewise such links can be used to disambiguate entities. From the perspective of the Web, these links form the mesh upon which the Web of Data is based.

But creating links is a challenging task for publishers. Addressing this challenge, a number of linking frameworks, such as Silk [12] and LIMES [7], have been proposed to help publishers link their local datasets to a remote LOD dataset through a specified SPARQL endpoint. However, given that there are now hundreds of public SPARQL endpoints, a still-more fundamental question has not been tackled: *how can publishers find SPARQL endpoints that are relevant targets for links in the first place*? As we will see, answering this question is non-trivial, particularly for specialised domains.

Currently, the selection of relevant endpoints relies on manual effort and requires experience and knowledge of available datasets and endpoints. One potential method is to manually inspect the list of datasets and endpoints listed on the DataHub catalogue, but only very high level dataset descriptions are available regarding the topic of the dataset and available access mechanisms. The content of SPARQL endpoints can be described using VoID[5], SPARQL 1.1 Service Descriptions, and specialised vocabulary (or ontology) [2], which may help, but these are not available for many endpoints [1, 11].

The most general option is to consider the SPARQL endpoints as black boxes whose content is opaque and directly query them to determine if they are relevant. In this paper, we explore this option. We assume that a high-quality, representative set of domain-specific keywords is made available as input to the process; this set of keywords may be extracted from any local source in any format – such as a taxonomy, a relational schema or a term dictionary – or can even be provided manually by an expert. Based on

[1] `http://lod-cloud.net/`; l.a. 2014/04/07.
[2] Statistics are quoted from `http://lod-cloud.net/state/`; l.a. 2014/04/07.
[3] `http://datahub.io/group/lodcloud`; l.a. 2014/04/07.
[4] See footnote 2.
[5] `http://www.w3.org/TR/void/`; l.a. 2014/04/07.

this set of domain-specific keywords, we propose to directly probe SPARQL endpoints with queries to determine their relevance. Initially, we identified three potential methods:

Full-text Search: SPARQL does not provide any standard full-text search functionality. Although some SPARQL vendors offer custom full-text search features implemented by efficient inverted indexes (e.g., Virtuoso provides a `bif:contains` keyword), not all SPARQL endpoints will support such features.

`REGEX` Filters: SPARQL allows for matching literals with `REGEX` filter expressions. However, filter expressions are applied as a post-processing step: using `REGEX` for full-text search would involve scanning all object literals in the dataset, rendering this method inefficient and impractical.

Exact Literal Matching: A final option is to create exact literals from the domain keywords that can be looked up directly. However, this method requires an exact literal to be matched, meaning an exact case-sensitive phrase match with the correct language tag.

Herein we focus on the third option given that it will work for any SPARQL endpoint and will involve efficient lookups (as opposed to inefficient post-filtering). In previous work, we proposed an algorithm (called QTermEx) for combining raw clinical terms into literals [6]. The QTermEx algorithm takes as input a terminology, in this case a set of Clinical Terms (*CTerms*), and generates a set of Query Terms (*QTerms*). The QTermEx algorithm incorporates stop-word removal, word permutations, and resolving unique tokens from *CTerms* represented as sentences (e.g., "Migraine cumulative (with aura)") or URIs (e.g., `http://www.chuv.ch/variables/schizophrenia/code:SZAPU2`). In the former case, for example, the *QTerms* extracted would be "Migrane", "cumulative", and "Migrane cumulative" after the punctuation and stop-words "aura" and "with" are removed. To keep the output succinct, the order of words is preserved in the QTermEx algorithm (i.e., it would not produce "cumulative Migrane"). We then query endpoints for the *QTerm* literals created from the terminology in this fashion.

We take the output from the QTermEx algorithm and expand it by creating multiple case and language-tag variants for each *QTerm* so as to generate more hits. We also present comparative evaluation of our methods for a real-world use-case involving three clinical partners who wish to publish Linked Data and need to find existing relevant datasets that can be linked to.

The rest of the paper is as follows: Section 2 presents the methodology towards term extraction and a multi-matching algorithm that creates literal variants. Section 3 presents evaluation of our method for three clinical terminologies in our use-case, seeking relevant LOD datasets for linking. Section 4 discusses related work and Section 5 concludes. But first we introduce our motivating scenario involving three clinical partners.

Motivating Scenario:. Our work is conducted in the context of the Linked2Safety[6] EU Project, where the consortium features three clinical partners, namely, University Hospital Lausanne (`CHUV`)[7], Cyprus Institute of Neurology and Genetics (`CING`)[8], and `ZEINCRO` (a contract research organisation offering clinical-trial services)[9]. These partners are associated with the high-level domains listed in Table 1.

Table 1: Example terms for Clinical Partners (CP)

	Domain	**Example Terms**
CHUV	Cardiovascular	Coronary Heart Disease
	Psychiatric Disorder	Major Depressive disorder
	Migraine	Migraine cumulative (with aura)
CING	Diabetes	Urine Microalbumin
	Breast Cancer	Breastfeeding duration
	Neurology	Spinal Muscular Atrophy
ZEINCRO	Concomitant Meds	Hepatic or Biliary
	Respiratory	Rate of Spirometry
	Medical History	Musculoskeletal

One of the core goals of the Linked2Safety project is to publish biomedical datasets provided by the clinical partners as high-quality Linked Data. Each clinical partner has provided clinical terminologies for their specialised domain with 150–215 terms each; Table 1 provides some examples.

The life-sciences community have been very active within the Linking Open Data movement: as aforementioned, 41 datasets on the LOD cloud are classified as specialising in the "Life Sciences" domain and 70 SPARQL endpoints have been made available by these publishers, most prominently by the Bio2RDF[10] and Linked Life Data[11] initiatives. Other general-knowledge datasets, such as DBpedia, also contain rich information about the life sciences. Manually identifying which of these LOD datasets are potential targets for links from the local datasets of each clinical partner is a time consuming process. Hence we propose methods that take as input three sets of terminologies exemplified in Table 1 and produce as output a list of potentially relevant SPARQL endpoints for linking.

2. TERM EXTRACTION AND MULTI MATCHING

As discussed earlier, we use our previously proposed algorithm (QTermEx) [6] for extraction of literals from *CTerms* (as exemplified in Table 1). These literals are then searched against public endpoints. We propose an algorithm in this paper that queries for case and language-tag variants for extracted literals.

2.1 QTermEx Algorithm

The first algorithm, *Query Term Extractor* (QTermEx, Algorithm 1), takes as input a terminology, in this case a set of Clinical Terms (*CTerms*), and generates a set of *Query Terms* (*QTerms*). Each $QTerm \in QTerms$ will subsequently be used to generate an RDF literal that can be queried against the SPARQL endpoints of various LOD datasets.

[6]`http://www.linked2safety-project.eu/`

[7]`http://www.chuv.ch/`; l.a. 2014/04/07.

[8]`http://www.cing.ac.cy/`; l.a. 2014/04/07.

[9]`http:/www.zeincro.com/`; l.a. 2014/04/07.

[10]`http://bio2rdf.org/`; l.a. 2014/04/07.

[11]`http://linkedlifedata.com/`; l.a. 2014/04/07.

All *CTerms* in the terminology are iterated over. The input *CTerm* is first pre-processed by replacing junk characters with spaces (*JunkCharSet*; e.g., punctuation, parentheses, symbols, etc.), by removing stop words (*SWSet*, e.g., "the", "and', etc.) and by removing general terms (*GTSet*; e.g., "duration", "rate", "family", etc.). Junk characters are pre-defined. Stop words are collected from the terminology using the Ranks.nl text-analysis tool[12]. General terms are specific to a given terminology and are defined by domain experts; these general terms are used to reduce the number of *QTerms* created in the final output. Taking the example *CTerm* "Migraine cumulative (with aura)" from Table 1, first the parentheses will be removed as junk characters, "with" will be removed as a stop-word and "aura" will be removed as a general term (if defined). A list of unique token words, *KwList*, is computed as a result.

Algorithm 1: QTermEx: Extracts Query Terms

```
Input: A finite set of Clinical Terms ($CTerms$)
Output: A finite set of Query Terms ($QTerms$)
$JunkCharSet$ := set of Junk Characters;
$SWSet$ := set of Stop Words;
$GTSet$ := set of General Terms;
$QTerms := \emptyset$;
for $CTerm$ in $CTerms$ do
    $CTerm'$ := replace each occurrence of $JunkCharSet$ in $CTerm$ with space;
    $TokenList$ := tokenise $CTerm'$ based on spaces;
    $KwList$ := empty list;
    for $Token$ in $TokenList$ do
        if $Token$ not in $SWSet, GTSet$ or $KwList$ then
            Add $Token$ to $KwList$;
    $QTerms'$ := all $n$-grams from $KwList$ that preserve ordering;
    $QTerms := QTerms' \cup QTerms$;
return $QTerms$;
```

Table 2: Distribution of *KwList* cardinalities

	$\|KwList\| = 1$	$\|KwList\| = 2$	$\|KwList\| = 3$
CHUV	140	70	5
CING	92	58	24
ZEINCRO	138	7	5

All combinations of n-grams[13] that preserve the ordering of the original input term (but potentially skip terms) are then computed from the resulting *KwList*. Thus, the result of our example would be three query terms: "Migraine", "cumulative" and "Migraine cumulative". These n-grams are added to the output *QTerms*. The total number of such n-gram query terms is $2^k - 1$ for $k = |KwList|$. For our use-case, Table 2 lists the distribution of cardinalities for *KwList* (as defined in Algorithm 2; e.g., 70 input keyword phrases from CHUV contained precisely two non-filtered keywords): the exponential combination of tokens into n-grams does not pose a significant problem since the size of *KwList* never exceeds 3. Thus our algorithm is best suited to terminologies with concise domain-specific phrases (after the removal of stop words and general terms).

[12]http://www.ranks.nl/

[13]We slightly abuse the term: an n-gram is a string containing n consecutive items (e.g., words) from the input in sequential order; we do not require that the n-gram contains consecutive words, only that it preserves order.

2.2 Multi-Matching: μMatch Algorithm

The *QTerms* generated from our algorithm QTermEx can be used to query the SPARQL endpoints of LOD datasets. We call this a Direct Matching (DM) approach, where each *QTerm* is used to generate a single query literal. Each literal is then used to generate a simple SPARQL query as "SELECT ?s ?p WHERE {?s ?p "Term" .}", which can be used to probe for relevant endpoints.

However, querying for exact literals is case sensitive and many literals in LOD datasets contain language tags. Our second algorithm, Multi Matching (μMatch; Algorithm 2), queries SPARQL endpoints for multiple case variants of each *QTerm* and for literals with language tags.

Algorithm 2: μMatch: Multi Matching Algorithm

```
Input: A Term ($CTerm$ or $QTerm$), A language tag ($langTag$)
Output: A finite set of dataset endpoints ($EPSet$)
$SEndpoints$ := set of catalogued Dataset Endpoints;
for $EP$ in $SEndpoints$ do
    $add$ := QueryAndLog($Term$, $langTag$, $EP$);
    $PCase$ := Proper Case $Term$;
    $add := add \vee$ QueryAndLog($PCase$, $langTag$, $EP$);
    $LCase$ := lower case $Term$;
    $add := add \vee$ QueryAndLog($LCase$, $langTag$, $EP$);
    $UCase$ := UPPER CASE $Term$;
    $add = add \vee$ QueryAndLog($UCase$, $langTag$, $EP$);
    if $add$ = true then
        $EPSet := EPSet \cup \{EP\}$;
return $EPSet$;
QueryAndLog($Term$, $langTag$, $EP$)
    $match$ := false;
    $Query$ := create SPARQL query for $Term$;
    $Result$ := supply $Query$ to $EP$ and retrieve results;
    if $Result$ not empty then
        $match$ := true;
        log $Result$;
    $LangCaseQuery$ := Add $langTag$ to supplied $Term$ and create query;
    $Result$ := supply $LangCaseQuery$ to $EP$ and retrieve results;
    if $Result$ not empty then
        $match$ := true;
        log $Result$;
    return $match$;
```

μMatch (Algorithm 2) takes as input the *QTerms* keywords generated by QTermEx and a list of URLs indicating the location of candidate SPARQL endpoints to query (*SEndpoints*). We generate *SEndpoints* by logging URLs of all candidate SPARQL endpoints in a particular domain. For our use-case, we created a list of *SEndpoints* specifically for the life-sciences domain. We considered SPARQL endpoints made publicly available by Bio2RDF[14] or that are

[14]http://download.bio2rdf.org/release/2/release.html; l.a. 2014/04/07.

tagged in the Datahub[15] repository with "LIFESCIENCES" or "HEALTHCARE"; we also include DBpedia as a central cross-domain dataset.

In the μMATCH algorithm, for each *Term*, we queried *SEndpoints* by executing the SPARQL query presented before. Since SPARQL is case-sensitive – for example, a literal value "cancer" is not same as "Cancer" or "CANCER" – the algorithm is refined to check for proper case, upper case and lower case literals. Furthermore, language tags are often used with literal values, for example, in some cases the literal value "cancer" is defined as "cancer"@en. Our algorithm is thus refined to check for literals with language tags. We use the @en language tag since the clinical terminologies contained in the datasets from our scenario are primarily in English. However, the proposed algorithm can easily accommodate other regional settings (for example: @de, @en-us, @en-uk, etc.), with the caveat that four additional queries will be required for each additional language tag. Likewise, we do not consider the datatype xsd:string, where a plain literal without a language tag, such as "cancer", is semantically equivalent to the same literal with a string datatype, such as "cancer"^^xsd:string. We omit this datatype since it is rarely used with prose strings (it is incompatible with language-tags which are recommended for internationalisation of prose strings); however, another variant could be added to our algorithm to support this datatype.

For each $Term$ supplied to the algorithm, we run μMATCH with a set of 8 queries: { original-case, proper-case, lower-case, upper-case } × { no-lang-tag, @en-tag }. All non-empty results are logged along with the URLs and names of the respective SPARQL endpoints in *EPSet*. In addition, the bindings returned for the ?s (subject) and ?p (predicate) query variables are also recorded and logged when a *Term* match is found on some particular SPARQL endpoint. The subject bindings are useful to count the number of unique entities in the remote dataset that are matched by some literal term. The predicates returned are potentially useful for linking frameworks, like LIMES [7] or SILK [12], that are dependent on mapping information for generating links, such as which predicates to use. For example, the predicates found when searching for "Cancer" on different SPARQL endpoints (given in Listing 1) could provide an entry point for linking based on string-similarity functions.

[15]http://datahub.io/; l.a. 2014/04/07.

```
P1: http://www.w3.org/ns/dcat#keyword.
P2: http://www.w3.org/.../rdf-schema#label
P3: http://cu.sgd.org/vocabulary:synonym
P4: http://www.w3.org/.../rdf-syntax-ns#value
P5: http://.../pubmed_vocabulary:keyword
```

Listing 1: Set of predicates for "Cancer"

3. EVALUATION

We carried out a series of experiments to compare our proposed algorithms using the three datasets provided by the three clinical partners mentioned in Table 1.

3.1 Experimental Setup

All experiments were conducted on a computer running 64-bit Windows 7 OS, with 4GB RAM and an Intel Core i5 (2.53 GHz) CPU. We use a MySQL RDBMS to store experimental data, including the endpoints to search and the results of successful queries. As previously discussed, we consider a total of 44 public endpoints from the Bio2RDF project and from the Datahub catalogue with "LIFESCIENCES" or "HEALTHCARE" tags. Queries are sent to the public endpoints over HTTP using the standard SPARQL protocol.

Given three sets of raw clinical terms (*CTerms*) as input, we generate three sets of query terms (*QTerms*) by applying the QTERMEX algorithm [6]. Based on these sets of terms, we then perform the following four experiments:

***CTerms–DM*:** We query endpoints using the raw clinical terms with the Direct Matching (DM) approach: each *CTerm* creates a single literal and a single query.

***CTerms–μM*:** We query endpoints using the raw clinical terms with the Multi Matching (μM) approach: each *CTerm* creates eight literals with combinations of case and language tags, resulting in eight queries per term.

***QTerms–DM*:** We query endpoints using the extended query terms with the Direct Matching (DM) approach: each *QTerm* creates a single literal and a single query.

***QTerms–μM*:** We query endpoints using the extended query terms with the Multi Matching (μM) approach: each *QTerm* creates eight literals and eight queries.

Table 3: Number of Queries generated per dataset per experiment

Dataset	CTerms		QTerms	
	DM	μM	DM	μM
CHUV	215	1720	385	3080
CING	174	1392	434	3472
ZEINCRO	150	1200	194	1552

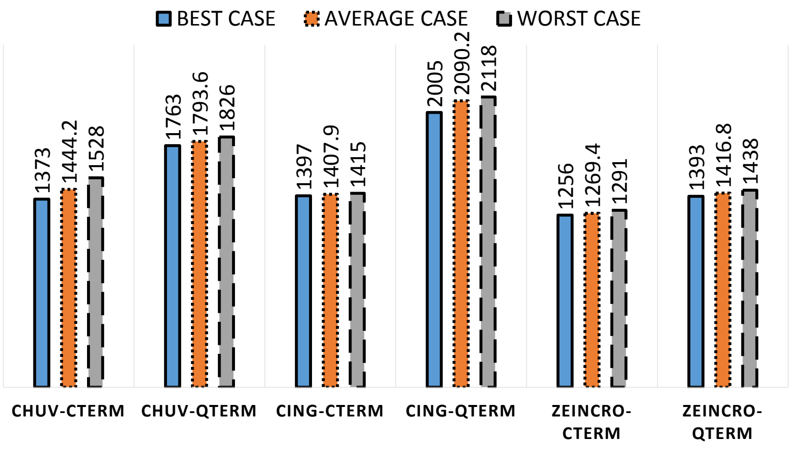

Figure 1: Query Generation Time (ms)

The total number of queries generated for each of the three datasets and each of the four experiments is shown in Table 3. We can see that the query load of Multi Matching approach is fixed at 8× that of the Direct Matching approach. We see that when using the expanded set of query terms, the query load increases by 1.3–2.5× versus the raw clinical terms. Furthermore, we see that our approach generates a non-trivial load of thousands of queries per dataset. However, the queries we issue are single-pattern atomic lookups, which should be efficient for the endpoint to execute.

Figure 1 shows the query generation time (in milliseconds) for all queries in each scenario categorised along two dimensions (1) μMatch using *CTerms*; and (2) μMatch using *QTerms*. Based on 20 runs (with terms shuffled between each run), we present times for the best, average, and worst runs. We observe that execution time for *CTerms Vs. QTerms* on all three cases does not deviate significantly. Thus we see that the performance of query generation is not a distinguishing factor for the different approaches.

Table 4: Fre-EPR for all datasets

Dataset	#	Endpoint Name	Results
CHUV	1	Pubmed	190651
	2	UniProt UniRef	51460
	3	Toxkb	5518
	4	CKAN	5044
	5	DBPedia	885
CING	1	Pubmed	187153
	2	UniProt UniRef	31938
	3	Toxkb	25614
	4	KEGG Pathway	17241
	5	SGD	11112
ZEINCRO	1	Bio2RDF Atlas	12672
	2	Pubmed	2671
	3	GOA	736
	4	NCBI-Gene	432
	5	Toxkb	345

3.2 Results

We compare the results of our four experiments based on three metrics:

Matched Results (*MR*): represents the total number of query results obtained for all terms against all SPARQL endpoints.

Matched Terms (*MT*): represents the number of distinct terms for which non-empty results were found for at least one SPARQL endpoint.

Matched Endpoints (*ME*): represents the number of distinct endpoints for which some term generated a non-empty result.

To illustrate these metrics, consider an example where a set of two $Terms$ = { "lung", "cancer" } is searched (using DM or μM) on $SEndpoints$ = { "GeneBank", "DrugBank", "PubMed" }, where the results contained 5 matches of "lung" in "GeneBank", 0 matches in "DrugBank" and 2 matches in "PubMed". Similarly, the returned results for "cancer" had 0 matches in "GeneBank", "DrugBank" and "PubMed". In this example, $MR = 7$, $MT = 1$ and $ME = 2$.

A detailed comparison of MR, MT and ME for all datasets and experiments is presented in Figures 2–4 respectively. In general, we see a small increase in the number of matches when considering DM versus μM; whether or not the 8× query-load of μM is cost effective depends on the scenario, where the trade-off is the completeness of results versus the efficiency of the discovery process. Conversely, we see a more significant increase for *QTerm* versus *CTerm*, which is associated with an increased query load of 1.3–2.5×: the process of removing junk characters, stop words and general terms, and generating a variety of n-grams, leads to a significant increase in matches, particularly for the dataset provided by the second clinical partner (i.e., CING).

Of the 44 endpoints we consider, the querying process generates positive hits for 9–33 endpoints. With a high ratio of potentially relevant endpoints being found, we thus deem it important to rank the relevance of these endpoints, allowing domain experts to consider more highly ranked endpoints as better candidates for the linking process. Thus, in addition to comparing MR, MT and ME, we also investigated two intuitive ranking schemes for endpoint-relevance with respect to each of the three local datasets, as follows. Ranking schemes are illustrated using the results collected from *QTerms*–μM experiment.

In the first step: we computed a ranking based on the number of distinct terms found per SPARQL endpoint, which we call the "EndPoint Ranking (*EPR*)". This metric is a per-endpoint version of MT that indicates how broad the coverage of domain terms is for each endpoint. The top-10 most relevant endpoints according to this metric for each of the three datasets is shown in Figures 5–7 respectively, where the x-axis corresponds to the names of the SPARQL endpoints and the y-axis represents the number of distinct terms with non-empty results. From these figures, it can be

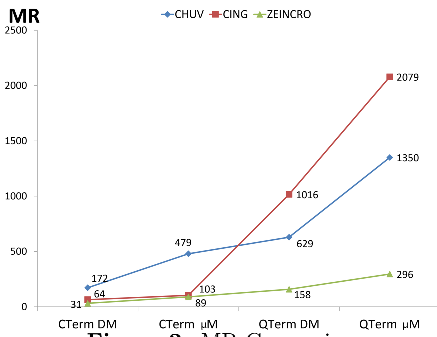

Figure 2: MR Comparison

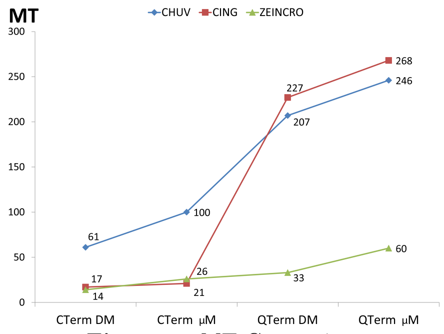

Figure 3: MT Comparison

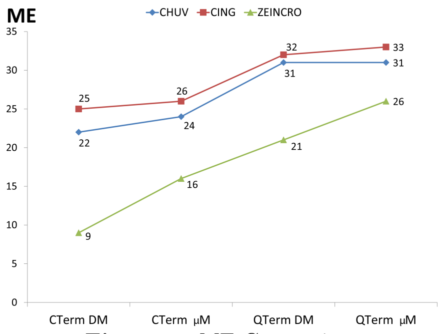

Figure 4: ME Comparison

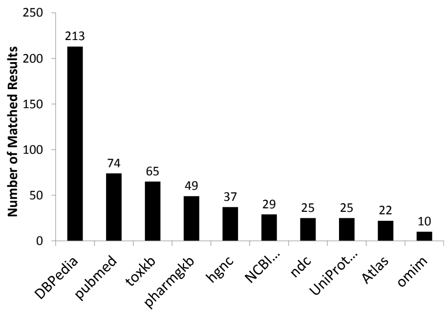

Figure 5: *EPR* for `CHUV`

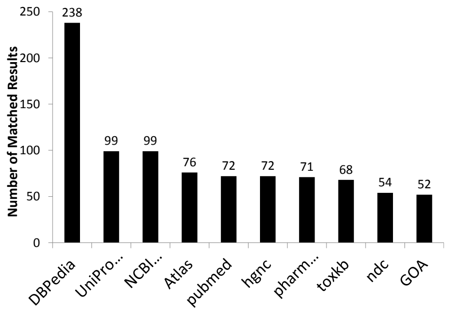

Figure 6: *EPR* for `CING`

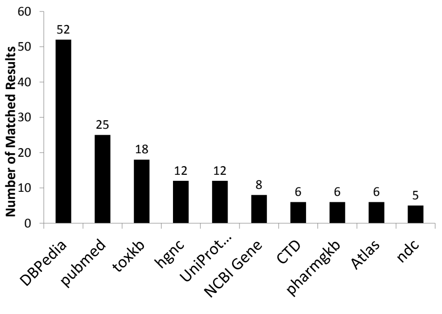

Figure 7: *EPR* for `ZEINCRO`

seen that the DBpedia endpoint had the broadest coverage of terms. We considered DBpedia for our experiments due to its cross-domain characteristics.

In the second step: we computed a ranking based on the number of results found per SPARQL endpoint across all terms, which we call the "Frequency of results per End-Point Ranking (*Fre-EPR*)". This metric is a per-endpoint version of *MR* that indicates how deep the coverage of specific domain terms is for each endpoint. A top-5 ranked comparison of the total number of matches found in different SPARQL endpoints for each dataset is provided by Table 4. As opposed to the *EPR* metric, which focuses on how broad the coverage of domain-specific terms, *Fre-EPR* catpures the depth of matches. Whereas DBpedia was most highly ranked for *EPR*, we find that more specialised endpoints are ranked more highly for *Fre-EPR*: the highest-ranked SPARQL endpoint for both `CHUV` and `CING` is "Pubmed" and for `ZEINCRO` is "BIO2RDF Atlas". More specialised datasets offer more hits for specific terms, suggesting that there are many potential entities that could be linked to in the target dataset.

We additionally look at the ratio of input terms for which some results were returned from some endpoint, which we refer to (loosely) as "recall". Based on a subset of already identified *CTerms* and/or *QTerms* on endpoints, we executed our experiments and computed recall where Table 5 presents the results for different configurations. The *QTerms* recall results for `CHUV` and `CING` improved significantly over the *CTerms* recall results, which can be attributed to the effect of trying multiple case variants and a language tag.

Figure 8 explores in more detail the effect of applying a language tag for the μMATCH algorithm, where Case-1 refers to Matched Results (*MR*) without language tag but with all case variants and Case-2 refers to Matched Results (*MR*) with and without a language tag on all case variants. We see results increase by a factor of 1.3–2.2× for different datasets. Many datasets (for example, DBpedia) use language tags on all labels for their resources.

4. RELATED WORK

With respect to finding relevant datasets on the Web, various keyword-based search engines have been proposed, including, for example, Sindice [10] or SWSE [3]. However, such search engines return entities as results and do not directly allow for finding relevant SPARQL endpoints.

To the best of our knowledge, only a few works have looked at identifying candidate datasets for interlinking. Leme at al. [4], rank the relevance of datasets for interlinking using probability measures based on existing links between datasets; however, the approach analyses the existing high-level link-network of datasets, which affects the generality of the approach (e.g., new datasets). Nikolov et al. [8, 9] propose an approach to identify relevant datasets for interlinking consisting of two steps: (1) searching relevant entities in other datasets using keywords; and (2) filtering irrelevant datasets based on semantic concept similarities using ontology matching techniques; however, their methods assume that the candidate datasets are centrally indexed.

Table 5: Recall for each dataset

Dataset	CTerms		QTerms	
	DM	μM	DM	μM
`CHUV`	0.17	0.23	0.64	0.88
`CING`	0.04	0.06	0.67	0.90
`ZEINCRO`	0.09	0.19	0.35	0.62

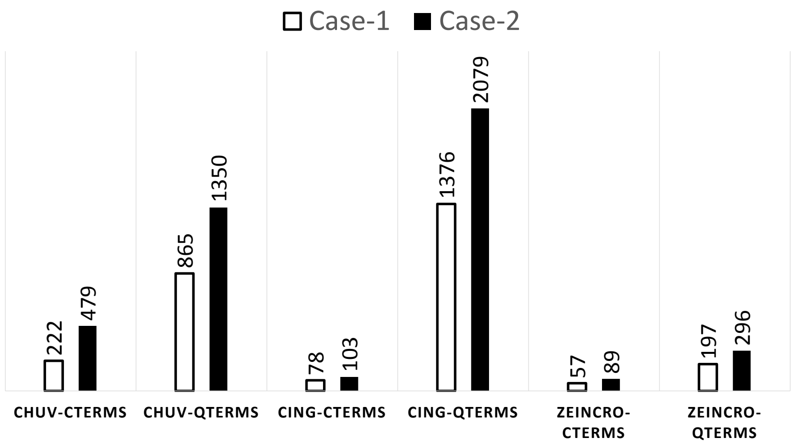

Figure 8: Impact of Language Tag on Matched Results (*MR*)

We previously mentioned that there are a number of linking frameworks available for Linked Data, including Silk [12] and LIMES [7]. Both offer a declarative language for guiding the creation of links between datasets based on predicates. However, both of these tools presume that a SPARQL endpoint for the target dataset is specified in the input. Our work addresses the prior step of identifying public endpoints of LOD datasets that are interesting to link to.

Maali et al. [5] propose an extension of the Google Refine tool to curate and RDFise local datasets. The extension can help find legacy URIs for entities from target endpoints specified by the user. The authors propose using custom full-text search over SPARQL endpoints to find relevant URIs for keyword terms in the local dataset; e.g., using `bif:contains` over Virtuoso endpoints. However, they presume that the endpoints of interest are manually specified by the user. As previously argued, we do not rely on the vendor-specific full-text search features that are supported only by some SPARQL endpoints.

Other works have discussed the difficulties of discovering relevant SPARQL endpoints on the Web. For example, Buil Aranda et al. [1] note that few structured descriptions are available for endpoints. Paulheim and Hertling [11] also note that the discovery of endpoints is a difficult problem and propose methods to find endpoints given a URI of interest.

To the best of our knowledge, no work has presented methods to discover public SPARQL endpoints that are most relevant to a list of domain-specific keywords (as needed by our three clinical partners).

5. CONCLUSION

In this paper, we propose methods for automatically discovering public SPARQL endpoints that are candidates for linking with a local domain-specific dataset. We are inspired by the needs of three clinical organisations that wish to generate Linked Data but are unsure which datasets are most relevant to link to.

Given a set of domain-specific keywords, we discussed three possible methods by which the relevance of SPARQL endpoints could be determined: we choose to investigate algorithms that seek exact literal matches since the generated queries (1) would involve standard SPARQL features supported by all endpoints, unlike vendor-specific full-text search primitives and (2) would involve efficient lookups, unlike `REGEX` filters that need to scan all indexed data.

However, exact literal matches require precise phrase matching and are sensitive to case, language tags, etc. We thus presented two algorithms to generate a variety of phrases from input keywords [6] and to query for variations of case and language-tag. In experimental results for three real-world clinical terminologies, we showed that these algorithms increase the number of hits generated versus the raw keywords. We also discussed some preliminary ranking methods for the relevance of individual endpoints.

In an extended work, we would like to investigate using the full-text search API of a Linked Data warehouse, such as Sindice, to generate a list of relevant entity URIs, querying the endpoint for the presence of these entities rather than for exact literal matches. We would also like to investigate the effectiveness of our methods for other domains. Finally, we would like to investigate ranking metrics for the relevance of endpoints in more depth, in particular how well they indicate the potential for generating high-quality links.

Acknowledgements

This publication has emanated from research supported in part by a research grant from Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289, by the EU project Linked2Safety (contract Number 288328), and by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004.

6. REFERENCES

[1] C. B. Aranda, A. Hogan, J. Umbrich, and P.-Y. Vandenbussche. SPARQL Web-querying infrastructure: Ready for action? In *ISWC*, pages 277–293. Springer, 2013.

[2] H. F. Deus, D. F. Veiga, P. R. Freire, J. N. Weinstein, G. B. Mills, and J. S. Almeida. Exposing the cancer genome atlas as a SPARQL endpoint. *Journal of Biomedical Informatics*, 43(6):998–1008, 2010.

[3] A. Hogan, A. Harth, J. Umbrich, S. Kinsella, A. Polleres, and S. Decker. Searching and browsing linked data with swse: The semantic web search engine. *J. Web Sem.*, 9(4):365–401, 2011.

[4] L. A. P. P. Leme, G. R. Lopes, B. P. Nunes, M. A. Casanova, and S. Dietze. Identifying candidate datasets for data interlinking. In *Web Engineering*, pages 354–366. Springer, 2013.

[5] F. Maali, R. Cyganiak, and V. Peristeras. Re-using cool URIs: Entity reconciliation against LOD hubs. In *Linked Data On the Web (LDOW) Workshop*. CEUR, 2011.

[6] M. Mehdi, A. Iqbal, A. Hasnain, Y. Khan, S. Decker, and R. Sahay. Utilizing Domain-Specific Keywords for Discovering Public SPARQL Endpoints: A Life-Sciences Use-Case. In *ACM SAC (SWA track)*, 2014 (to appear).

[7] A.-C. N. Ngomo and S. Auer. LIMES – a time-efficient approach for large-scale link discovery on the Web of Data. In T. Walsh, editor, *IJCAI*, pages 2312–2317. IJCAI/AAAI, 2011.

[8] A. Nikolov and M. d'Aquin. Identifying relevant sources for data linking using a Semantic Web index. In *Linked Data On the Web (LDOW) Workshop*. CEUR, 2011.

[9] A. Nikolov, M. d'Aquin, and E. Motta. What should I link to? Identifying relevant sources and classes for data linking. In *JIST*, pages 284–299. Springer, 2011.

[10] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *IJMSO*, 3(1):37–52, 2008.

[11] H. Paulheim and S. Hertling. Discoverability of SPARQL endpoints in Linked Open Data. In *ISWC (Posters & Demos)*, pages 245–248. CEUR, 2013.

[12] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk - a link discovery framework for the Web of Data. In *Linked Data On the Web (LDOW) Workshop*. CEUR, 2009.