Dynamic query matching

Fiona McNeill, Andriana Gkaniatsou, Alan Bundy

School Of Informatics
University of Edinburgh
f.j.mcneill@ed.ac.uk, agkaniat@ed.ac.uk, a.bundy@ed.ac.uk

Abstract. This paper describes the CHAIn system, which uses data matching based on the Structure-Preserving Semantic Matching (SPSM) algorithm to match queries which are incompatible with the schema of the queried datasources to relevant parts of the schema. These matchings are then used to reformulate the query and retrieve relevant responses to those expected by the original query. Despite the growing interest in intelligent query answering, we believe that integration of data matching into query answering is novel, and allows users to successfully query datasources even if they do not know how the data in that source is organised. We describe the process of a proof-of-concept system and briefly describe the encouraging evaluation we have so far carried out.

1 Introduction

Fast, effective data sharing is essential in many situations; one of the fields in which this is particularly pressing is in the field of emergency response. Emergency response situations are characterised by the coming together of large numbers of organisations, each of which is likely to have large amounts of data, much or some of which may be pertinent to the current emergency. Some of these organisations may be well known to each other, but others will be unknown and potentially untrusted. Datasources even of known organisations may be highly dynamic. Reports into previous emergency responses frequently cite poor communication and failure to effectively share information as a significant barrier to an effective response. There is much interest in increasing levels of automation in this process as an attempt to address these problems [9]. There are many problems surrounding such automation; the one in which we are particularly interested is that of mismatch between datasources.

Successful querying of a datasource depends on a good understanding of that datasource, thereby ensuring that the schema of the query correctly aligns with the schema of the queried datasource. If data querying is part of an automated process, such knowledge depends on being able to anticipate in advance exactly what datasources will be relevant and knowing accurately what the schema and data representations of that source will be at the time of querying. In a highly dynamic environment, such as emergency response, such assumptions are often not valid. There is always a possibility of dynamic interaction with new organisations, not anticipated at design-time, and of interacting with known organisations who have updated or altered their data in some way. Thus, pre-alignment of datasources is impractical. Since speed is of the essence in emergency situations, relying on human ability to identify these problems and update queries accordingly is not feasible; in addition, this depends on the ability to access the schema of other people's data, which is not always permitted. The sources of the queries in most cases are unknown and the queries are not predefined, thus, aligning sources before the querying process seems impractical. We therefore believe that automatic reformulation of queries based on matching between the schema of the original query and that of the queried datasource is a necessary part of automating this communication process.

This paper introduces the CHAIn (Combining Heterogeneous Agencies' Information) system, which can be used by an owner of a datasource to formulate appropriate responses to incoming queries, even when these queries fail to match the datasource at the schema level and/or the data level. We can assume that the schema is available without loss of privacy, as CHAIn functions owner side and details of the schema need not be passed to any other party. Potential responses are ranked, and the process may be fully automated or may be used as a basis for fast, efficient human interaction with large datasources.

New sentence about schema privacy

The matching component of CHAIn is based on the Structure-Preserving Semantic Matching (SPSM) algorithm [4]. This is originally developed in the OpenKnowledge project[1] and it is, currently, the only system that can match entire structures and not only nodes. The main contribution of this paper is in the adaptation of the structure-preserving matching process to the problem of dynamic query reformulation in (potentially) large data. Our hypothesis is that we will be able to provide results to queries that otherwise would fail, and that such results will be meaningful.

Hypothesis

The paper is organised as follows. Section 2 uses a worked example to describe the aims of the system. Section 3 describes the process of the system in more detail. Section 4 discusses the results we have obtained by using this system on various emergency response datasources. Section 5 puts our work in the context of other related work. Section 6 concludes the paper and discusses some of the key issues we need to address in developing CHAIn from a proof-of-concept system to a system that is usable in the field.

2 Worked Example

We have primarily been considering query matching in relation to an emergency response scenario. Such situations are data rich, and are characterised by the need to share data quickly and effectively with collaborators who may be previously unknown, and may not be trusted. Formulating correct queries under these circumstances is extremely difficult, and there is, therefore, a pressing need for automated query reformulation.

Consider a flooding scenario. Imagine an environmental organisation which is trying to determine how full the rivers are to anticipate the course of the floods.

[1] http://www.openk.org/

They will have their own sensors monitoring this, but may want to get additional information from other organisations which also have sensors - for example, other environmental agencies from upstream regions and private companies who monitor water levels for their own commercial purposes, but may be agreeable to sharing data during an emergency. Perhaps in their own datasources they frame readings from their sensors as follows:

$measurement(Reporter\_ID, Node, Level, Date)$

They will then send this query to other organisations which they believe may have relevant information. If the organisation were automatically developing queries to populate this table, the query that would be formulated, in SQL-like format, would be: **SELECT** $reporter\_ID, node, level, date$

FROM $measurement$

If there is no mismatch at either the data or the schema level, the query will succeed and appropriate responses will be returned without the need to invoke the CHAIn system. However, the queried organisation may organise their data differently on many levels. For instance:

$reading(Node, Date, Water\_level)$

There are various mismatches here, in the words used, in the organisation of the arguments and in the number of arguments. The query above will fail, so, returning an appropriate response will require alignment with the datasource schema.

3 Query Response Process

The goal of the CHAIn system is to return an appropriate response to a query on a datasource despite the fact that the query is not correctly formulated for that datasource. Whenever a query fails, CHAIn will first investigate whether the schema used by the query is correct with reference to the schema of the queried datasource. If it is not, matching is used to determine whether there is anything in the schema of the datasource which approximates to the schema of the query, and, if so, reformulates the query accordingly. If a query fails when there is no mismatch in the schema - either because the original schema of the query was correct according to the schema of the datasource, or because the query has been reformulated to reflect the schema of the datasource - the CHAIn system will then consider potential mismatches at the data level. Ultimately, the query will be responded to using the data that CHAIn considers the most appropriate response to the original query, or it will fail if there is nothing in the queries datasource which is considered sufficiently relevant.

In this section, we describe the various aspects of the process, with reference (where appropriate) to the worked example in Section 2.

3.1 Lifecycle of the Process

1. A query to a particular datasource is received by the organisation (or individual) that owns the datasource.

2. If the query fails, it is first determined whether this is due to mismatch at the schema level. If not, go to step 6. If so, naive matching is done on the schema of the datasource to narrow down the search space to likely matches (see Section 3.3).
3. Potential matches are sent pair-wise with the query to the SPSM matcher.
4. The SPSM matcher returns matches together with a score, which can be used to reject poor matches and rank good matches (see Section 3.4).
5. The query is reformulated according to the matches and resent to the datasource (see Section 3.5).
6. If the query fails, since at this stage the query is correct with respect to the schema, we look for mismatches at the data level (see Section 3.5).
7. Potential responses to the query passing a given threshold are presented, with appropriate annotation describing the match, to a user who is knowledgable about the datasource. This user then choses any number of matches to be returned to the querier. (This step can be omitted and the responses can be returned according to automatic ranking if required) (see Section 3.6).
8. The querier receives the response(s) to their original query, together with information about the matches that were used to produce the response, and uses these as they feel appropriate.

3.2 Format of the datasource

The matching part of the process matches first-order terms: that is, terms of the form $predicate(Arg_1, Arg_2, ..., Arg_n)$, where each Arg_i may itself be a function. This process is therefore applicable to data formats that can be translated into this format. Since this is a very general format, this includes a vast number of common representations. Currently, the CHAIn system can be used for SPARQL queries to RDF datasources and for SQL queries to RDB datasources, but it could easily be extended to be applicable to more formats. Extending to a new format requires extensions to the querying and reformulating part of the process, but does not affect matching, the central process, or the interaction with the user.

3.3 Narrowing down search space

The schema matching part of the process is done by the SPSM algorithm, which does pairwise matching of structured terms. This is an expensive process, and it is not feasible to do it between a single query and every possible (potentially large) datasource that we own. It is therefore necessary to perform a filtering step, narrowing down the datasources to a subset of likely candidates. This is done through fairly naive keyword matching the query datasource name to the datasources we own. This allows us to select a set of datasources and their corresponding schemata as candidates for matching the query. Throughout all this process we do not alter any of the data[2].There is, of course, a tension in deter-

[2] There is a considerable body of work on matching keywords, and incorporating some of this would allow us to develop a more sophisticated approach to this step of the process. We have chosen to do this in a naive way because we are developing a proof-of-concept system, but this would be an important part of future work.

mining how wide this net should be cast: we do not wish to exclude potentially good matches; however, we do not wish to end up with a very large subset of the data, to perform complex matching on which would be very expensive. Initial results (see Section 4) indicate our approach is workable, but extensive simulation is necessary to determine more precisely where this sweet spot lies.

After a query failure for the query dataset name, we compute the *Related Terms* set, to which we then try to match the dataset names we own. The schema of each of the matching dataset names and the query schema are the SPSM input. *Related Terms* is computed based on the SUMO and Wordnet ontologies, and the process is explained below.

SUMO Compute *Related Terms*: For the following process we consider (i) the original query datasource name Q, (ii) the set $SQ = \{m_1, m_2, ..., m_l\}$ which consists of all subwords of the original query datasource name (if Q is multi-word).

- $i \in$ *Related Terms* if:
  - $relates(i, Q)$ and/or $relates(Q, i)$ and $relates \in SUMO$
- $n \in$ *Related Terms* if:
  - $relates(n, m_i)$ or $relates(m_i, n)$ and $relates \in SUMO$ and $m_i \in SQ$
  - $relates(n, m_k)$ or $relates(m_k, n)$ and $relates \in SUMO$ and $m_k \in SQ$
  - $m_i \neq m_k$

Wordnet Compute *Related Terms*: For the following process we consider the set $SQ = \{m_1, m_2, ..., m_l\}$ which consists of all subwords of the original query datasource name (if it is not multi-word, then SQ consists of a single word -the original datasource name).

- For each $m \in QS$
  - find the Wordnet id of that word
  - *Related Terms* is the set of all $Hyponym$, $Hypernym$, *Subclass*, *Superclass*, $Meronym$, $Similar$ that relate to m.
- Repeat the same process for each $Hyponym$, $Hypernym$, *Subclass*, *Superclass*, $Meronym$ and $Similar$.

Schema Candidates For Matching Decompose the datasources names we own into sub-words, if possible. Then, (i) Match the datasources names with the SUMO and Wordnet *Related Terms* we previously computed, (ii) Perform string matching.

For example, the *Related Terms* for the predicate name *measurement* would consist of 38 instances, including *sampling*, *observation* and *reading*. We would then target datasource for any predicates with any of these names, and return them. Perhaps, for example, it would find an exact match for *measurement* and a match for the *Related Term reading*.

3.4 Structure-preserving semantic matching

Explained why we need spsm

To match the query to any of the datasources that we own, we need to establish whether there exists a correspondence between their structures *i.e.* schemata. Thus, we need full graph-like structure matching, rather than matching the nodes of graph-like structures. The Structure-Preserving Semantic Matching (SPSM) algorithm performs full graph-like structure matching matching, thus, we consider it as the most appropriate matcher for our project. SPSM has been extensively described in [4], so we do not cover it in detail here.

SPSM requires two trees as input and returns a score in [0 1] indicating their similarity, together with mappings from each element of the source tree to elements in the target tree, or to null. These element mappings must be $\in \{=, \subset, \supset\}$.

SPSM is a two-step process. We first make use of adapted conventional ontology matching techniques to investigate relationships between the individual words in the nodes of the trees. Then, we match the structure of the trees to discover an overall relationship. This is crucial because the structure of the tree itself contains a lot of semantic information that must be considered if we are determining whether two terms are equivalent or similar. We therefore, need to preserve a set of structural properties (e.g., vertical ordering of nodes) to establish whether two trees are globally similar and, if so, how similar they are and in what way.

3.5 Reformulating the query

The SPSM process returns a list of ranked responses. We see from the narrow-down process (see Section 3.3) that there are two predicates in the target data-source which are considered relevant: *measurement* and *reading*. These predicates, together with their arguments, are then sent to SPSM. In this case, the results are:

1. $reading(Node, Date, Water\_level)$
2. $measurement(Area, Wind\_speed, Direction, Date)$

Here, we see that the exact predicate match is returned lower than the related predicate match, because its arguments are much less relevant to the arguments of the original query. For each response which exceeds a given threshold, the mappings in the match are then used to reformulate the query.

1. Original Query Schema: $measurement(Reporter\_ID, Node, Level, Date)$
Mappings: $measurement \subset reading$
$measurement/Node = reading/Node$
$measurement/Level \supset reading/Water\_level$
$measurement/Date = reading/Date$
Suggested Query Schema: $reading(Node, Water\_level, Date)$
2. Original Query Schema: $measurement(Reporter\_ID, Node, Level, Date)$
Mappings: $measurement = measurement$
$measurement/Date = measurement/Date$
Suggested Query Schema: $measurement(Date)$

For aspects of the query for which mappings are found, the original term is replaced by the matched term; aspects of the query for which no mappings are found are removed from the query. If the latter occurs, the response will not constitute a complete response to the query but may contain sufficient detail to be useful.

The repair process takes as input the mismatched query (both data and schema) and the SPSM matching results and produces new queries which are considered schematically correct. We consider the following cases:

- The query dataset name can match to more than one dataset name.
- Each dataset match comes with the corresponding argument matches.
- Within the same dataset, an argument can have more than one match (more than one argument that can be mapped to).
- A dataset match might correspond to more than one new query.
- A dataset match that does not have any argument matching is discarded.

Forming the final query - Data Level Repair The inputs of this process are all query schemata that have been formed by the previous process and for each schema the corresponding data. For each schema we form the corresponding query and we send it to the appropriate dataset. If no answers are returned, assuming that the schema is correct, we uninstantiate the data values of the query. The goal of this process is to identify the data values that cause the query failure. The steps we perform are the following:

- Find all data values of the query.
  - Select one data value at the time and uninstantiate it.
  - Send new query to the dataset.
  - Check if it returned any answers.
    - If no answers are returned, repeat for all uninstantiated data combinations systematically until one returns answers.
- Uninstantiate all data values of the query.

The system currently does no matching at the data level, instead returning all responses which match some of the query data, or, if necessary, only the schema of the query. We consider matching at the data level important, and it is in our future plans.

3.6 Returning query responses

Up to this point, the process has been fully automated and, as mentioned earlier, it is possible to run the CHAIn system without any user interaction if desired. However, we feel[3] that some level of user interaction will be important in improving the quality of results. We also see the CHAIn system as a useful tool in allowing efficient human interaction with large datasource, guiding the human as to how to appropriately respond to queries to the datasource by reducing a large datasource to a handful of potential responses, together with information to help them understand in what way the responses answer the query, and which aspects of the query are unanswered.

Helping users, who may have a good understanding of the data used by their organisation, but are unlikely to be experts in representation and matching, to appropriately and efficiently evaluate a set of potential responses to a query, weighing them against each other with reference to the particular ways

[3] More extensive evaluation is necessary to clearly demonstrate this.

in which each individual response fails to be an exact response to the original query, is a complex and demanding task, requiring high-quality techniques in human-computer interaction. Whilst building such expertise into the system will be essential before this approach is usable in the field, we do not pretend to such expertise ourselves, and therefore have a much simpler approach to user interaction, with a complete approach to this relegated to future work.

4 Evaluation

Large-scale evaluation of this kind of matching is difficult. This is chiefly because:

- we need to develop specific queries in each case, ideally generated from data and schema from existing datasources;
- these need to be sent to a datasource from which we could realistically expect an answer - that is, the query and the receiving datasource need to be similar enough to expect results;
- we need to analyse the results, with the help of experts where appropriate, to judge whether or not the responses are appropriate;
- we need to examine the receiving datasource to see if there were any appropriate responses which were not returned.

A robust evaluation of this approach requires these queries to be generated during the kind of automated process we envisage this work be used in, and the effectiveness of these matches determined through large-scale simulation, to determine what the actual impact of using these matches are. This is outside the scale of our current pilot project (see Section 6 for discussions of our plans to do this). We have therefore carried out the evaluation as follows: (i) We have sourced appropriate data from a collaborator, SEPA (Scottish Environmental Protection Agency). (ii) We have sourced online different datasources which discuss similar things, such that we might expect a query from such a datasource to find an appropriate response in our SEPA data.

For each evaluation test we select a dataset and we formed a query based on the schema of that dataset[4]. Then, we send the query to the remaining datasets we own. We have so far carried out 26 evaluation tests. Of these, 14 returned responses to the queries. In all cases the responses were judged appropriate: answers were relevant to those expected. 12 did not return a result. In 4 of these cases, this was because the narrow-down process returned no results, and in all cases this was judged an appropriate response because there was nothing in the target datasources which matched the query predicate name (not returning irrelevant results is almost as important as returning relevant results). 8 of these failed at the SPSM matching after receiving input from the narrow-down process. Of these, 3 were appropriate failures, because the arguments of the narrowed-down predicates were not similar enough to the query to justify return data; 5 failed because of implementational issues which we believe can be fixed without

[4] The selected dataset always has relevant information with those to query.

difficulty. In this section, we present two tests, to give a flavour for the results we have achieved. In both cases the queries are sent to SEPA datasources[5].

I think we should show the datasources schemata, but i haven't due to space limitations

Test 1 We form a query based on the EuroStat[6] Fresh Water Resources dataset schema: water(resource, measure, geo, timePeriod) The narrow-down process returned the following dataset names as matching candidates:

waterBodyPressures
waterBodyMeasures
surfaceWaterBodies
bathingWaterLocations

SPSM retuned the following matches:

- water $\mapsto$ waterBodyPressures
  water(timePeriod) $\mapsto$ waterBodyPressures(identifiedDate)
  water(geo) $\mapsto$ waterBodyPressures(waterBodyId)
  water(measure) $\mapsto$ waterBodyPressures(assessmentCategory)
  water(resource) $\mapsto$ waterBodyPressures(source)

added 1)resulted query, 2) timings. review-2

After applying the matches, the reformulated query is: waterBodyPressures(identifiedDate, waterBodyId, assessmentCategory, source) , which we sent to the datasources and retrieved the corresponding answers. The timings from the first call of our system, until the finish are: 2m10.208s. From which 0m49.725s is the amount of CPU time spent in user-mode code and 0m8.023s is the amount of CPU time spent in the kernel within the process.

Test 2 We formed a query based on the UK Environment Agency[7], Bathing Waters dataset schema: uk_BathingWaters(sampleClassification, prefLabel, long, lat, northing, easting, latestComplianceAssessment, type, country, district,latestBathingWaterProfile, sedimentTypesPresent, uriSet, regionalOrganization, yearDesignated, latestSampleAssessment, eubWidNotation, waterQualityImpacted- ByHeavyRain, zoneOfInfluence, samplingPoint, complianceClassification, primary- Topic, extendedMetadataVersion, definition, label). The narrow-down process returned the following dataset names as matching candidates:

bathingWaters
bathingWaterLocations
waterBodyMeasures
waterBodyPressures

SPSM retuned the following matches:

- uk_BathingWaters $\mapsto$ waterBodyMeasures
  uk_BathingWaters(label)$\mapsto$ waterBodyMeasures(waterBodyId)
  uk_BathingWaters(primaryTopic)$\mapsto$ waterBodyMeasures(secondaryMeasure).
- uk_BathingWaters $\mapsto$ waterBodyPressures
  uk_BathingWaters(label)$\mapsto$ waterBodyPressures(waterBodyId)
  uk_BathingWaters(type) $\mapsto$ waterBodyPressures(pressureType)
  uk_BathingWaters(prefLabel) $\mapsto$ waterBodyPressures(activity)

[5] SEPA datasources: bathingWaterLocations, waterBodyPressures, waterBodyMeasures, surfaceWaterBodies, bathingWaters, classifications.
[6] epp.eurostat.ec.europa.eu
[7] www.environment-agency.gov.uk/

uk_BathingWaters(sampleClassification) ↦ waterBodyPressures(affectsGroundwater)

– uk_BathingWaters ↦ bathingWaters.
  uk_BathingWaters(label) ↦ bathingWaters(description)

added 1) resulted queries, 2)timings. review-2

After applying the above matches we form the following queries (i) waterBodyMeasures(waterBodyId, secondaryMeasure) (ii) waterBodyPreassures(waterBodyId, pressureType, activity) (iii) bathingWaters(description) which we send to the datasets and retrieve the corresponding answers. The timings from the first call of our system, until the finish are: 3m55.407s. From which 1m25.673s is the amount of CPU time spent in user-mode code and 0m18.614s is the amount of CPU time spent in the kernel within the process.

5 Related Work

The work in this paper is concerned with applying techniques that have already been evaluated and used in other contexts to a new domain. The SPSM algorithm is novel in that whilst it is built on standard tree-edit-distance techniques (e.g., in [10], using semantic matching to determine what impact on the overall meaning of a tree altering or removing will have. Previously, it has been used to perform service integration and peer selection in a peer-to-peer network [8]. The novelty of our work is in developing a system that can use this structured matching techniques to address the problem of returning useful results to a query on a datasource which has not been correctly formulated according to the schema and data within that datasource.

There is a considerable body of work on the problem of intelligent query answering, which aims to do more than simple look-up based on the query. However, this work tends to focus on different problems than the one we are interested in. The issue generally addressed concerns how to return maximum, or maximally useful information and may use information within the datasource - for example, a concept hierarchy - to return information relevant to terms not directly included in the query. However, it is not generally concerned with the problem of querying a datasource with which one is not familiar, so that the terms and structure used may be wholly or partially inappropriate for the datasource: the need to use matching techniques is not considered.

[6] describes various approaches to this issue, where intelligent query answering is used to provide more (relevant) detail than the query originally asked for, to provide summaries of data, and to reformulate queries into sub-queries. However, the issue of data matching is not considered; rather, the question is about how much relevant information can be returned, assuming the query is correctly formed. Work looking into intelligent query answering tend to assume far more complex databases than we have considered, where deduction rules, concept hierarchies and knowledge discovery tools are available. We have kept our attention focussed on simple databases because we believe these are most common in the emergency-response scenario we are focussing on; however, investigating the combination of our matching techniques with intelligent-query-answering techniques for situations where more complex databases were available would be very

interesting and could result in a more sophisticated approach to query answering. [6] also raises interesting questions about how to appropriately interact with the user and how to ensure that sensitive data is released in an appropriate way to appropriate people, which are also very relevant to an in-the-field version of our work. [3] proposes the *DB-Pattern-KB* framework and the type abstraction hierarchy to improve query answering.

Work has also been done on combining data from different sources to provide intelligent responses to queries (see, for example [5]). [1],[7] investigate how semantic relations based on Wordnet can improve query expansion. However, in all these works the problem of data mismatch between the query and the datasource is not considered.

In [2] the Ontology-Based Data Access framework proposes the use of an ontology to create views of the datasources, so, a user can have access to the data without knowing the data organization. This work follows the main idea of query rewriting whereas views of the original datasource are created and the original query is rewritten to retrieve answers from that views. In our approach, instead of creating views of the datasources we align the query on the fly to the most appropriate datasources to retrieve answers.

query rewriting and views

6 Conclusions and Further Work

The problem of successfully querying unknown datasources is an important one in terms of developing the vision of the Semantic Web, where interactions can be dynamic and unpredictable. The general assumption in dataset querying is that the querier knows the datasource and can use the correct schema and terminology when developing their query, but this is a massively limiting assumption as it restricts querying to situations where: 1) one knows in advance what datasources one will wish to query; 2) one has the time to devote to examining the datasource and correctly formulating any queries one may have; 3) the datasource is open and one can examine its schema. We believe this discounts a large number of situations in which one or more of these assumptions are not valid, and we particularly motivate this with reference to an emergency response situation, where collaborations are formed dynamically and quickly, where information needs to be passed rapidly, and where the size, number and potential sensitivity of available datasources ensure that it is impossible to adequately understand all relevant data.

The CHAIn system described in this paper uses an established structural matching algorithm - SPSM - to address the problem of mismatched dataset queries. We are centrally interested in how techniques developed for ontology matching can be used to address other situations in which data mismatch of some sort is a problem. CHAIn is currently a proof-of-concept system and requires significant further development and extensive evaluation based on simulation before we expect it to be usable in the field. However, the implementation and evaluation we have produced so far have demonstrated that this approach to queries which are mismatched with the datasource they are querying is valid and potentially useful. There are many ways in which we intend to extend this work:

- Extending the matching techniques, for example to be more domain-sensitive and to include matching at the data level.
- Building in information about provenance and trust so that users of the system have more confidence in query responses.
- Developing more sophisticated ways to interact with users.
- Integrating the matching system into a larger system which controls the interactions during emergency responses (e.g., how organisations identify and interact with one another).
- Extending the system to cope with more sophisticated datasources, and integrating techniques from intelligent query answering.
- Extensive evaluation based on simulation.

6.1 The role of humans

moved it to future work

CHAIn can operate completely automatically because the matching process ranks potential matches. It is therefore possible to always choose the best match at each stage, to produce a single best-ranked response to the original query. Setting CHAIn to be fully automated is the best approach where responses are needed very quickly and quality of matching is not important. However, we also view CHAIn as a method for allowing human users to interact with large datasources quickly and effectively, providing them with the tools to make informed decisions. It is therefore the general expectation that when the matching process is completed, a set of results passing a given threshold for matching quality, or constrained by the number of required results, is presented to a user that owns, or belongs to an organisation that owns, the queried datasource. These results will be presented with clear information about where approximations were made, where parts of the query were left unanswered, and so on. This allows for the input of local knowledge into context-free matching, and helps datasources where localised terminology is used to become more universal comprehensible. For example, *reading* is considered by WordNet to be a direct hypernym of *measurement*, so a match may be found on that basis. However, someone who understood the data may know that *reading* in this context refers to book readings, and is not a correct match for *measurement*. These kinds of poor matches are likely to be identified through poor matches in the arguments of the terms, but local input can still be valuable.

This user can decide to send back one or more (or none) of the responses suggested by CHAIn to the querier. Again, the querying organisation can automatically accept this response if desired, or it can also be filtered by a human user within that organisation to determine whether to accept the response, or which of a list of potential responses is most appropriate. This will depend on what the information is needed for; for example, a response missing one particular attribute of the query may be useless, whereas one missing another attribute may be of some value. In the running example, perhaps $reporter\_ID$ is not essential because the *Node* information can be used to determine where the reading is coming from, so a match where this is missing would be less valuable than one where it is present, but still of some use. However, perhaps the query is primarily to discover the water level at various places, so if the attribute *water_level* were missing, the response would have no value to the querier.

References

1. Davide Buscaldi, Paolo Rosso, and Emilio Sanchis Arnal. A wordnet-based query expansion method for geographical information retrieval. In *Working notes for the CLEF workshop*, 2005.
2. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The mastro system for ontology-based data access. *Semantic Web Journal*, 2(1):43–53, 2011.
3. Wesley W. Chu and Qiming Chen. Neighborhood and associative query answering. *Journal of Intelligent Information Systems*, 1(3-4):355–382, 1992.
4. Fausto Giunchiglia, Fiona McNeill, Mikalai Yatskevich, Juan Pane, Paolo Besana, and Pavel Shvaiko. Approximate Structure-Preserving Semantic Matching. In *7th International Conference on Ontologies, Databases and Applications of Semantics (ODBASE)*, pages 1217–1234, November 2008.
5. Stefan Hagedorn and Kai-Uwe Sattler. Discovery querying in linked open data. In *EDBT/ICDT Workshops*, pages 38–44, 2013.
6. Jiawei Han, Yue Huang, Nick Cercone, and Yongjian Fu. Intelligent query answering by knowledge discovery techniques. *IEEE Transactions on Knowledge and Data Engineering*, 8:373–390, 1995.
7. Laura Hollink, Schreiber Guus, and Bob Wielinga. Patterns of semantic relations to improve image content search. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5.3:195–203, 2007.
8. Nardine Osman, Carles Sierra, Fiona McNeill, Juan Pane, and John Debenhaml. Trust and Matching Algorithms for Selecting Suitable Agents. *ACM Transactions on Intelligent Systems and Technology (TIST)*, in press.
9. The Pitt review: Lessons learned from the 2007 floods. HMSO, London, 2007.
10. Dennis Shasha and Kaizhong Zhang. Approximate Tree Pattern Matching. *In Pattern Matching Algorithms*, pages 341–371, 1997.