Feedback-Based Annotation, Selection and Refinement of Schema Mappings for Dataspaces*

Khalid Belhajjame
School of Computer Science
University of Manchester
Manchester, UK
khalidb@cs.man.ac.uk

Norman W. Paton
School of Computer Science
University of Manchester
Manchester, UK
norm@cs.man.ac.uk

Suzanne M. Embury
School of Computer Science
University of Manchester
Manchester, UK
sembury@cs.man.ac.uk

Alvaro A. A. Fernandes
School of Computer Science
University of Manchester
Manchester, UK
afernandes@cs.man.ac.uk

Cornelia Hedeler
School of Computer Science
University of Manchester
Manchester, UK
chedeler@cs.man.ac.uk

ABSTRACT

The specification of schema mappings has proved to be time and resource consuming, and has been recognized as a critical bottleneck to the large scale deployment of data integration systems. In an attempt to address this issue, dataspaces have been proposed as a data management abstraction that aims to reduce the up-front cost required to setup a data integration system by gradually specifying schema mappings through interaction with end users in a pay-as-you-go fashion. As a step in this direction, we explore an approach for incrementally annotating schema mappings using feedback obtained from end users. In doing so, we do not expect users to examine mapping specifications; rather, they comment on results to queries evaluated using the mappings. Using annotations computed on the basis of user feedback, we present a method for selecting from the set of candidate mappings, those to be used for query evaluation considering user requirements in terms of precision and recall. In doing so, we cast mapping selection as an optimization problem. Mapping annotations may reveal that the quality of schema mappings is poor. We also show how feedback can be used to support the derivation of better quality mappings from existing mappings through refinement. An evolutionary algorithm is used to efficiently and effectively explore the large space of mappings that can be obtained through refinement. The results of evaluation exercises show the effectiveness of our solution for annotating, selecting and refining schema mappings.

Categories and Subject Descriptors

H.0 [**Information Systems**]: General

General Terms

Algorithms, Experimentation

*The work reported in this paper was supported by a grant from the EPSRC.

EDBT 2010, March 22–26, 2010, Lausanne, Switzerland.

Keywords

Dataspaces, Pay-as-you-go, User Feedback, Data Integration, Mapping Annotation, Mapping Selection, Mapping Refinement

1. INTRODUCTION

The problem of data integration has been investigated for the past two decades with the aim of providing end users with integrated access to data sets that reside in multiple sources and are stored using heterogeneous representations [16]. Data integration has numerous potential applications, e.g., it can be used for enabling cross-querying of data stored in databases that belong to different departments or organizations, or to promote collaboration in large scientific projects by providing investigators with a means for querying and combining results produced by multiple research labs (e.g., [29]).

The recent increase in the amount of structured data available on the Internet, due in significant measure to the Deep Web [13], has created new opportunities for using data integration technologies. Yet, in spite of the long standing investigations in data integration, this technology seems to have had a limited impact in practice. By and large, data integration mechanisms are manually-coded and tightly bound to specific applications. The limited adoption of data integration technology is partly due to its cost-ineffectiveness [12]. In particular, the specification of schema mappings (by means of which, data structured under the source schemas is transformed into a form that is compatible with the integration schema against which user queries are issued) has proved to be both time and resource consuming, and has been recognized as a critical bottleneck to the large scale deployment of data integration systems [12, 17].

To overcome the above issue, schema mappings can be derived using information obtained from schema matching techniques [22]. In their simplest form, the schema matchings derived are binary relationships, each of which connects an element of a schema, e.g., a relational table in a source schema, to an element that is (predicted to be) semantically equivalent in another schema, e.g., a relational table in the integration schema. Schema matching techniques can be used as a basis for the generation of complex mappings that specify, for example, how the instances of one element of an integration schema can be computed by using the instances of two or more elements in source schemas [21, 27].

The mappings output by the above methods are derived based on heuristics. Therefore, some (if not most) of them may not meet

end users' needs. Consider, for example, the case of Clio [21]. To specify complex mappings that involve two or more relations in the source schemas, these relations are combined using, for example, a join predicate that is derived based on referential integrity constraints between the relations in question. While intuitive, this approach does not guarantee that the mapping obtained meets the requirements of end users. This raises the question as to how generated schema mappings can be verified.

A handful of researchers have investigated the problem of schema mapping verification [4, 5]. For example, the Spicy system provides functionalities for checking a set of mappings to choose the ones that represent better transformations from a source schema into a target schema [4]. To do this, the instances obtained using the mappings under verification are compared with instances of the target schema, which are assumed to be available. Chiticariu *et al.* developed a tool for debugging schema mapping [5]. Specifically, the tool can be used to compute *routes* that describes the relationship between source and target data of a given schema mapping.

Using the above tools, the verification of schema mappings takes place before the data integration system is set-up, which may incur a considerable up-front cost [9, 12]. In this paper, we explore a different approach in which generated schema mappings co-exist, and are verified in a pay-as-you-go fashion. Instead of verifying schema mappings before they are used, we consider that the data integration system is setup using as input schema mappings that are derived using mapping generation techniques. These mappings are then incrementally annotated with estimates of precision and recall [25] derived on the basis of feedback from end users. Our approach to mapping annotation is consistent with the dataspaces aim to provide the benefits of classical data integration while reducing up-front costs [12]. We do not expect users to be able to (directly) confirm the accuracy of a given mapping nor do we require them to give feedback based on the mapping specification; direct manual refinement of schema mappings requires significant expertise [5]. Instead, the feedback required from users provides information about the usefulness of the results obtained by evaluating queries posed using the generated mappings. With this information, we then annotate the mappings. Specifically, we estimate the precision and recall of the mappings, given the results they return and based on the feedback supplied by the user. For example, consider a mapping *m* that is candidate for populating a relation *r* in the integration schema. Based on user feedback that picks tuples that belong to *r* and tuples that do not, we estimate the precision and recall of the results retrieved using *m*. They are no more than estimates because we do not assume the user has complete knowledge of the correct extent to be returned and, therefore, do not ask the user to judge every tuple returned. In this paper, we report on an evaluation of the quality of the resulting mapping annotations for different quantities of user feedback.

Individual elements of the integration schema will frequently be associated with many candidate mappings. We consider a setting in which the candidate mappings are generated based on a large number of matches obtained using multiple matching mechanisms. Therefore, evaluating a user query using all candidate mappings incurs a risk of dramatically increasing the query processing time, and of obtaining a too large collection of results, the majority of which do not meet user needs. We present a method that, given user feedback, a query, and user requirements in terms of precision and recall, selects the set of mappings to be used for evaluating the query that are likely to meet the stated requirements. Specifically, this method casts the problem of mapping selection as a constrained optimization problems, i.e., that of identifying the subset of the candidate mappings that maximize the recall (resp. precision) given a minimum threshold for the precision (resp. recall).

Mapping annotations may reveal that the quality of schema mappings is poor, i.e., they have low precision and recall. We address this issue by refining schema mappings with a view to constructing new, *better* quality-mappings. Our approach to mapping refinement combines and mutates the candidate mappings to construct new mappings with better precision and recall. The space of mappings that can be obtained through refinement is potentially very large. To address this issue, we present an evolutionary algorithm for exploring this space.

In summary, the contributions of this paper are:

- An approach for incrementally annotating schema mappings based on user feedback. This is to our knowledge the first study that investigates the use of feedback supplied by end users to annotate schema mappings with estimates of their precision and recall.
- A method for schema mapping selection. Given a set of feedback instances supplied by the user, we present a method that can be used for selecting mappings whose results meet user requirements in terms of precision and recall.
- An evolutionary algorithm for mapping refinement. Using a set mutation and cross-over operators, we show how user feedback can inform the construction of better quality mappings from an initial set of candidate mappings.
- An empirical evaluation assessing the effectiveness of our solutions for annotating, selecting and refining schema mappings.

The paper is structured as follows. We begin by presenting the model for capturing user feedback, in Section 2. We then show how feedback instances of this model can be used for annotating schema mappings, and we report the results of an evaluation exercise that assesses the quality of annotations obtained, in Section 3. We describe, in Section 4, a method for selecting schema mappings, and present the results of an experiment that assesses the effectiveness of this method. We describe, in Section 5, a collection of mutation and cross-over operators that can be used to support mapping refinement. We also present an algorithm for exploring the space of mappings that can be obtained using those operators, and report on the results of an experiment that assesses the effectiveness of this algorithm. We analyze and compare existing works to ours, in Section 6, and conclude the paper in Section 7.

2. CANDIDATE MAPPINGS AND USER FEEDBACK

We begin by introducing the notion of candidate mappings and by presenting the model for defining user feedback.

A data integration system is essentially composed of four elements, namely the schemas of the sources, the data sets to be integrated, an integration schema over which users pose queries, and schema mappings that specify how data structured under the schemas of the sources can be transformed and combined into data structured according to the integration schema [8]. A schema mapping can be defined by the pair $\langle q_i, q_s \rangle$, where q_i and q_s are two queries of the same arity over the integration schema and the source schemas, respectively. It specifies that the concepts represented by the queries q_i and q_s are semantically equivalent [16]. For the purposes of this paper, we confine ourselves to mappings that relate one element in the integration schema to a query over the source schemas: these mappings are called in the literature *global-as-view* mappings [16]. We also adopt the relational model for expressing integration and source schemas. We, therefore, define a schema mapping m by the pair $m = \langle r_i, q_s \rangle$, where r_i is a relation in the integration schema, and q_s is a relational query over the source schemas. We use $m.integration$ to refer to r_i, and $m.source$ to refer to q_s.

To respond to the need for rapid data integration, existing schema matching techniques can be used to produce the input for algorithms capable of automatically generating the mappings between the integration schema and the source schemas (e.g., [22]). Multiple matching mechanisms can be used, each of which could give rise to multiple mapping candidates for populating the elements of the integration schema. To answer a user query *uq*, issued against the integration schema, each relation r_i involved in *uq* needs to be reformulated in terms of the source relations using a mapping candidate. This raises the question as to which mappings among the candidate mappings of r_i to use for answering a user query.

Candidate mappings may be labeled by scores that are derived from the confidence of the matches used as input for the generation of mappings (e.g., [6]). This suggests that the candidate mappings with the highest scores can be used for reformulating users' queries. However, since the confidences of matches, and therefore the scores of mappings, are computed based on heuristics, there is no guarantee that the mapping with the highest score reflects the true needs of end users [4, 10]. Moreover, in a data integration setting, (a sample of) the content of the integration schema is rarely available, and therefore instance-based matchers may not be able to be used to match the source schemas to the integration schema. Thus, the likelihood that the scores associated with the mappings are inaccurate can be higher than in situations in which the contents of the schemas to be matched are available, e.g., in data exchange [8].

In this paper, we use a different source of information for assessing candidate mappings, namely user feedback. In doing so, the user is not provided with a set of (possibly complex) mapping expressions; rather, s/he is given a set of answers to a query issued against the integration schema and which was answered using one or more candidate mappings. To further illustrate the kinds of feedback that can be supplied, consider that the user issued a query to retrieve the tuples of the relation r_i in the integration schema. This query is evaluated using one or more mappings that are candidates for populating r_i, and the query results are displayed to the user. The user then examines and comments on the results displayed using the following kinds of feedback:

- That a given tuple was expected in the answer.
- That a certain tuple was not expected in the answer[1].
- That an expected tuple was not retrieved.

The kinds of feedback we have just described are tuple-based in the sense that they comment on the correctness of the membership relation between tuples and the result set returned by a set of mappings. A finer grained form of feedback can also be supported. In particular, the user can indicate that a given attribute of r_i cannot have a given value. As in information retrieval [23], we assume that users provide feedback on a voluntary basis: they are not required to comment on every single result they are given, rather, they supply feedback on the results of their choice.

To cater for the types of feedback introduced above, we define a feedback instance uf provided by the user by the tuple:

$$uf = \langle AttV, r, exists, provenance \rangle$$

where r is a relation in the integration schema, $AttV$ is a set of attribute-value pairs $\langle att_i, v_i \rangle$, $1 \leq i \leq n$, such that $att_1, \ldots, att_n$ are attributes of r, and $v_1, \ldots, v_n$ are their respective values. *exists* is a boolean specifying whether the attribute value pairs in $AttV$ conform to the user's expectations. To specify whether *exists* is true or not, we assume the existence of a function $extent(r)$, that

[1]This form of feedback is called negative relevance feedback in the information retrieval literature [23].

Protein

	accession	name	gene	length	mappings
t_1	P17110	Chorion protein	cp36	320	m_1, m_4
t_2	X51342	genomic DNA	cp36	142	m_1, m_2
t_3	HSP70_CERCA	HSP70	Ensembl	231	m_1
t_4	P91902	HP70	CTXD	526	m_2, m_3
t_5	Q06589	Cecropin-1	CEC1	63	
t_6	Uniprot	NU5M_CERCA	Oxidoreductase	234	m_2

Figure 1: Example of query results

returns the set of tuples that belong to r in the user conceptualization of the world. Note that $extent(r)$ is not available, rather we obtain incomplete information about the tuples that belong to $extent(r)$ through user feedback. *exists* is true iff there is a tuple in $extent(r)$ in which the attributes $att_1, \ldots, att_n$ take the values $v_1, \ldots, v_n$, respectively. That is:

$$|t \in extent(r) \; s.t. \; \forall \; i \in \{1, \ldots, n\}, \; t[att_i] = v_i| \geq 1$$

$t[att]$ denotes the value of the attribute att in the tuple t. $provenance$ specifies the origin of the attribute value pairs on which feedback was given. These could have been provided by the user, or obtained from the sources using one or multiple mappings. Therefore:

$$provenance \in \{\text{`}userSpecified\text{'}, Map\}$$

$provenance =$ '*userSpecified*' means that the attribute value pairs $AttV$ are provided by the user. $AttV$ may also be retrieved from the sources in which case $provenance = Map$, where Map is the set of mappings that can be used to retrieve $AttV$ from the sources.

As an example, consider a life scientist who is interested in studying the proteome of the Fruit Fly. Given that data describing this proteome is stored across multiple bioinformatics sources (e.g., Uniprot[2], IPI[3] and Ensembl[4]), the scientist needs to access and combine data that belong to these sources. In doing so, the scientist prefers to use a given (integration) schema: this schema can be manually designed or automatically derived from a set of queries that are of interest to the scientist. Rather than attempting to manually specify the schema mappings between the source schemas and the integration schema, the scientist opts for a low upfront-cost option whereby candidate schema mappings are automatically derived. Once the mappings have been derived, the scientist issues a query to find the available proteins of the Fruit Fly. This query is evaluated using the mapping candidates for populating the *Protein* relation. Assume that the evaluation results are displayed as shown in Figure 1. The *mappings* column specifies the candidate mappings that were used for retrieving a given tuple. Note that, in general, this column would not be visible to the user; it is displayed in Figure 1 for ease of exposition only.

The user examines the results displayed and supplies feedback specifying whether they meet the requirements. For example, the feedback instance uf_1 given below specifies that the tuple t_1, which was retrieved using the mappings m_1 and m_4 is a *true positive*, i.e., meets the user's expectations.

$uf_1 = \langle AttV_1, Protein, true, \{m_1, m_4\} \rangle$
$AttV_1 = \{\langle accession, \text{`}P17110\text{'} \rangle, \langle name, \text{`}Chorionprotein\text{'} \rangle, \langle gene, \text{`}cp36\text{'} \rangle, \langle length, \text{`}320\text{'} \rangle\}$

[2]http://www.uniprot.org
[3]http://www.ebi.ac.uk/IPI
[4]http://www.ensembl.org

The user can also provide feedback specifying *false positives*, i.e., results that do not meet the requirements. For example, the feedback instance uf_2 below specifies that the accession of a protein cannot have the value *X51342*; indeed, this is a DNA accession, not a protein one. Similarly, the feedback instance uf_3 below specifies that the protein named *HP70* does not belong to the *CTXD* gene.

$$uf_2 = \langle\{\langle accession, \text{'}X51342\text{'}\rangle\}, Protein, false, \{m_1, m_2\}\rangle$$
$$uf_3 = \langle\{\langle name, \text{'}HP70\text{'}\rangle, \langle gene, \text{'}CTXD\text{'}\rangle\}, Protein, false, \{m_2, m_3\}\rangle$$

In addition to true positives and false positives, the user can also specify false negatives, i.e., results that are expected by the user and are not returned. The tuple t_5 is, for example, a false negative as specified by the following feedback instance.

$$uf_4 = \langle AttV_4, Protein, true, \text{'}userSpecified\text{'}\rangle$$
$$AttV_4 = \{\langle accession, \text{'}Q06589\text{'}\rangle, \langle name, \text{'}Cecropin-1\text{'}\rangle, \langle gene, \text{'}CEC1\text{'}\rangle, \langle length, \text{'}63\text{'}\rangle\}$$

Note that for the purposes of this paper, we assume that a group of users that supply feedback have the same requirements, in the sense that they agree about *extent(r)* for any relation *r*, and that those requirements do not change over time. The study of changing requirements and its symptoms, e.g., conflicts between provided feedback instances, is outside the scope of this paper.

In what follows, given a mapping candidate *m* for populating a relation *r* in the integration schema, and given a set of feedback instances *UF* supplied by the user, we use *tp(m,UF)*, *fp(m,UF)*, *fn(m,UF)*, respectively, to denote the true positives, false positives and false negatives of *m* given the feedback instances in *UF*.

3. ANNOTATING SCHEMA MAPPINGS

Using feedback instances of the form described in the previous section, we now show how candidate mappings can be labeled with annotations specifying their fitness to user expectations.

Using a simple annotation scheme, we can annotate a schema mapping as either correct or incorrect: a mapping is correct iff it meets the needs of users, i.e., its source query returns all the answers users expect, and does not return any results that are not expected by users. However, since the set of candidate mappings derived using matching algorithms is likely to be incomplete in the sense that it may not contain any candidate mapping that exactly meets user needs, there is a risk of annotating as incorrect all the mapping candidates for populating a given relation in the integration schema. We, therefore, opt for a less stringent annotation scheme that tags schema mappings with metrics specifying the degree to which they meet user requirements.

3.1 Cardinal Annotations

The quality of mapping candidates for populating a relation *r* in the integration schema can be quantified using precision and recall [25]. Of course, we cannot compute these metrics since they require the availability of the extent of *r*, i.e., the set of tuples that belong to *r* in the users' conceptualization of the world. Notice, however, that the feedback instances supplied by users provide *partial* information about the extent of *r*. Specifically, they allow the identification of (some of the) true positives, false positives and false negatives of a given candidate mapping. Using this information, we can compute precision and recall *relative* to (the extent of *r* identified through) the feedback supplied by the user.

We adapt the notions of precision and recall defined in information retrieval [25] to measure the quality of a mapping. We define the precision of a mapping *m* relative to the feedback instances in *UF* as the ratio of the number of true positives of *m* given *UF* to the sum of true positives and false positives of *m* given the feedback instances in *UF*. That is:

$$Precision(m, UF) = \frac{|tp(m,UF)|}{|tp(m,UF)| + |fp(m,UF)|}$$

where $|s|$ denotes the magnitude of the set *s*. Similarly, the recall of a mapping *m* relative to the feedback instances in *UF* is defined as the ratio of the number of true positives of *m* given *UF* to the sum of true positives and false negatives of *m* given the feedback instances in *UF*. That is:

$$Recall(m, UF) = \frac{|tp(m,UF)|}{|tp(m,UF)| + |fn(m,UF)|}$$

Following Van Rijsbergen [25], we can also compute a form of F-measure relative to user feedback that combines both precision and recall, and can, therefore, be used for ranking candidate mappings. The relative F-measure of a mapping *m* w.r.t. the feedback instances in *UF* can be defined as:

$$F(m, UF) = \frac{(1+\beta^2) \times Precision(m,UF) \times Recall(m,UF)}{\beta^2 \times Precision(m,UF) + Recall(m,UF)}$$

where β is a parameter that controls the balance between precision and recall. For example, if β is *1* then precision and recall have the same weight.

Given that relative precision and recall are computed based on feedback instances that provide only *partial* knowledge about the extent of the integration relation, the following question arises:

How much user feedback is required for approximating the real precision and recall, i.e., those based on complete knowledge of the extent of the integration relation?

To investigate the above question, we used two datasets: the Mondial geographical database[5] and the Amalgam data integration benchmark[6]. We created an integration relation *FavoriteCity* and mapped it to the relations in the Mondial database, and created an integration relation *MyReading* and mapped it to the source relations in Amalgam. To specify the mappings, we used the tool provided by the IBM Infosphere Data Architect[7]. For the purposes of our experiment, we needed to increase the number of candidate mappings for the relations *FavoriteCity* and *MyReading*. To do this, we randomly mutated the mappings obtained using the IBM Infosphere Data Architect by joining them with other relations in the schema of the Mondial database and Amalgam data sources respectively, and randomly combined them using the union and intersection relational operators. In total, we obtained 50 candidate mappings for populating the *FavoriteCity* relation, and 100 candidate mappings for populating the *MyReading* relation. We then specified the extents of the integration relations *FavoriteCity* and *MyReading* by randomly selecting a subset of the tuples returned by their candidate mappings. These extents serve two purposes. They allow automatic generation of synthetic user feedback. Also, they allow computation of "gold standard" mapping annotations against which the annotations computed based on user feedback, are compared.

To annotate the mappings, we applied the two-step process illustrated below for *60* iterations.

1. Generate feedback instances.
2. Compute the relative precision and recall of the candidate mappings given cumulative feedback.

[5]http://www.dbis.informatik.uni-goettingen.de/Mondial
[6]http://dblab.cs.toronto.edu/ miller/amalgam
[7]http://www-01.ibm.com/software/data/studio/data-architect

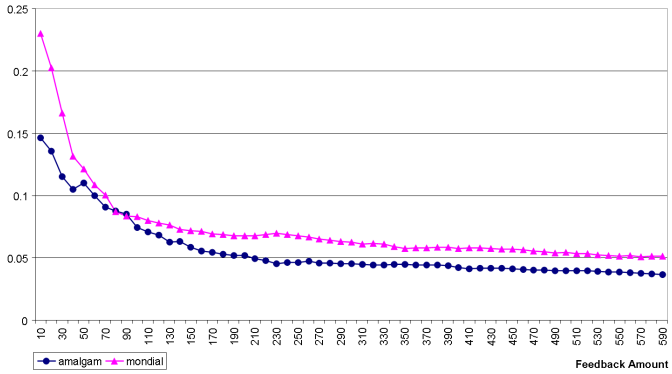

Figure 2: Average error in precision

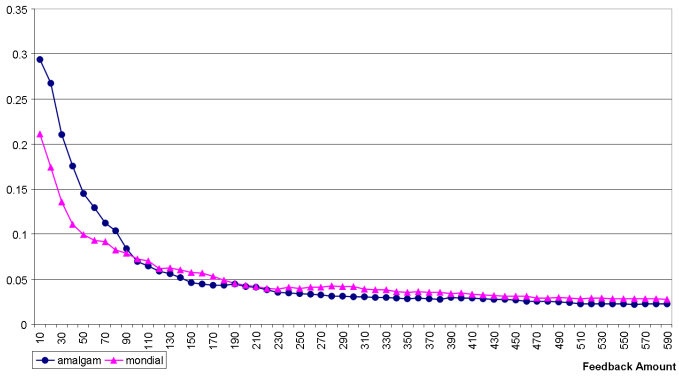

Figure 3: Average error in recall

At every iteration, we generated a set of feedback instances. Specifically, we randomly generated *10* feedback instances by applying a stratified sampling to the set of tuples returned by the candidate mappings. Together, the candidate mappings for populating the *FavoriteCity* relation return *2210* tuples, and those for populating the *MyReading* relation return *2138* tuples. Stratified sampling is a method of sampling using which the members of a population are first grouped into relatively homogeneous sub-populations before sampling. In our case, there are three sub-populations, namely true positives, false positives and false negatives. This sampling method improves the representativeness of the feedback instances generated. It ensures that the number of true positives, false positives and false negatives specified by the sample of feedback instances generated are in proportion to the number of true positives, false positives and false negatives in the result set obtained using all candidate mappings.

We repeated the annotation experiment described above *25* times; we chose this number of executions as it has been shown statistically to yield good estimations [26]. To measure the quality of the relative precision and recall of the candidate mappings obtained given user feedback, we then computed the average error in precision and recall at every feedback iteration. Figure 2 illustrates the average error in precision, i.e., the difference between the relative precision of a candidate mapping computed using supplied feedback and the correct "gold standard" precision, and Figure 3 illustrates the average error in recall.

The analysis of these results shows that the errors in precision and recall drop significantly during the first few feedback iterations. Specifically, *80* feedback instances (resp. *90* feedback instances) yielded a precision and recall that are close to the actual ones with an error smaller than *0.1* for the candidate mappings of the *FavoriteCity* relation (resp. *MyReading* relation).

These numbers, i.e. *80* and *90*, are relatively small if we consider the number of tuples returned by the candidate mappings: *80* tuples represents *3.64%* of the total number of tuples retrieved by the mapping candidates for populating *FouvouriteCity*, and *90* tuples represents *4.21%* of the total number of tuples returned by the mapping candidates for populating *MyReading*. Also, they are relatively small if we consider the number of candidate mappings subject to annotation: *50* in the case of *FavoriteCity* and *100* in the case of *MyReading*. The results of this experiment also suggest that the number of feedback instances required is not dependent on the number of mappings subject to annotation. This is a good result since it means that the number of feedback instances required for obtaining good estimates for precision and recall does not increase with the number of candidate mappings.

The figures also show that the more feedback instances are supplied, the smaller is the error in precision and recall, steadily decreasing but bounded by diminishing returns. This means that the quality of mapping annotations is incrementally improved as the user provides feedback instances, which is in line with the *pay-as-you-go* philosophy behind dataspaces.

3.2 Ordinal Annotations

Another source of information that can be exploited for mapping annotation is the dependencies between the candidate mappings in terms of the tuples they retrieve from the sources. We capture these dependencies in the form of ordinal annotations that partially order the candidate mappings in terms of true positives and false positives. Consider, for example, two candidate mappings m_1 and m_2 for populating a relation r in the integration schema. We say that m_1 covers m_2 in terms of true positives iff m_1 retrieves all the true positives that are obtained using m_2 given the feedback instances in *UF*, i.e., $tp(m_2, UF) \subseteq tp(m_1, UF)$. We write $m_2 \leq_{tp}^{UF} m_1$. Similarly, m_1 covers m_2 in terms of false positives iff m_1 returns all the false positives that are obtained using m_2 given the feedback instances in *UF*, i.e., $fp(m_2, UF) \subseteq fp(m_1, UF)$. We write $m_2 \leq_{fp}^{UF} m_1$.

Notice that computing ordinal annotations requires comparing the results of the source queries of every two candidate mappings, which can be time consuming. The time taken can be reduced by using query containment. Specifically, if the source query of a mapping m contains the source query of a mapping m', then we can deduce that m covers m' in terms of both true positives and false positives: $m' \leq_{tp} m$ and $m' \leq_{fp} m$.

Ordinal annotations are used as input to the mapping refinement process in Section 5.3.

4. SELECTING SCHEMA MAPPINGS

An element of the integration schema is likely to be associated with many candidate mappings; we consider a setting in which the candidate mappings are generated based on a large number of matches obtained using multiple matching mechanisms. In this context, by evaluating a user query using all candidate mappings, there is a risk of dramatically increasing the processing time of the user query, and obtaining a large collection of results, the majority of which do not meet the user's needs. We show in this section how the mapping annotations presented in the previous section can be employed for selecting from the set of candidate mappings, those to be used for populating a given element in the integration schema.

4.1 Mapping Selection as an Optimization Problem

Not all users of information integration systems will have the same requirements in terms of precision and recall. Consider, for example, a data integration system providing access to available proteomic data sources. A drug designer who issues queries to this data integration system may require high precision; the existence of false positives in query results may lead to the further expensive investigation of inappropriate candidate drugs. On the other hand, an immunologist who is using the proteomic data integration system to identify the proteins responsible for an infection may tolerate low precision, since further investigation may potentially give rise to the identification of new proteins associated with the infection under investigation.

To enable mapping selection to be tailored to meet user requirements, we use a selection method that aims to maximize the recall of the results obtained, while guaranteeing that their precision is higher than a given threshold λ_p, which can be specified by the user. The selection method can be formulated as a search problem that maximizes the following evaluation function.

$$eval_{recall}(V, UF, \lambda_p) = \begin{cases} 0, & \text{if } prec(V, UF) < \lambda_p \\ recall(V, UF), & \text{otherwise} \end{cases}$$

where $V = \langle b_1, \ldots, b_n \rangle$ is a vector that specifies the mappings selected: b_i is a boolean that is true iff the candidate mapping m_i is selected. *prec(V,UF)* and *recall(V,UF)* denote the precision and recall of the union of the results obtained using the mappings specified by *V* given the feedback instances in *UF*. An optimization problem is then to identify values for *V* (i.e., subsets of mappings) that maximize $eval_{recall}$ while meeting the constant λ_p.

The above evaluation function poses a problem during the search for suitable values for *V*, in that it associates all vectors of mappings with a precision below λ_p with the same value, zero. Therefore, a vector of mappings with a precision that closely misses λ_p is ranked equally as badly as a vector with zero precision. To overcome this problem, we make use of a function definition from previous work by Menascé and Dubey for estimating utility in service-oriented architectures [19] to refine that evaluation function into one that returns, for those vectors of mappings with a precision lower than the threshold λ_p, a decreasing value as the precision decreases, as follows:

$$eval_{recall}(V, UF, \lambda_p) =$$

$$\begin{cases} recall(V, UF) \times K_p \times C(prec(V, UF)), & \text{if } prec(V, UF) < \lambda_p \\ recall(V, UF), & \text{otherwise} \end{cases}$$

where $C(x)$ is a monotonic function that increases as x increases:

$$C(x) = \frac{1}{1 + e^{-x+\lambda_p}} - \frac{1}{1 + e^{\lambda_p}},$$

and K_p is scaling factor, $K_p = \frac{2 \times (1 + e^{\lambda_p})}{e^{\lambda_p} - 1}$, so that $lim_{prec(V,UF) \to \lambda_p}(eval_{recall}(V, UF, \lambda_p)) = recall(V, UF)$.

We then aim to maximize the $eval_{recall}$ function, i.e., we cast the problem of search for the best vector of mappings as a constrained optimization problem. For this purpose, we use the Mesh Adaptive Direct Search (MADS) approach [1], which is centered on a non-linear search algorithm that is appropriate for solving black-box constrained optimization problems.

The user may also be interested in maximizing the precision of the results obtained provided that the recall is higher than a given threshold, λ_r. As for the above case, this problem can be formulated as a search that aims to maximize the evaluation function obtained by swapping the roles of precision and recall in $eval_{recall}(V, UF, \lambda_p)$.

4.2 Experimental Evaluation

The approach described for mapping selection raises the following question:

Is the quality of the results obtained using schema mappings selected based on user feedback close *to the quality of the results obtained when the mappings are selected based on their correct "gold standard" annotations?*

To answer the above question, we applied the process illustrated below for multiple iterations to the mapping candidates of the *FavoriteCity* relation.

i Generate feedback instances.

ii Select mappings that maximize the evaluation function $eval_{recall}$ with a precision threshold λ_p.

At every iteration, we generate a set of *10* feedback instances. As in the experiment reported in Section 3, the number of true positives, false positives and false negatives specified by the feedback instances generated are proportionate to the number of true positives, false positives and false negatives returned by all mappings. We then select the mappings that maximize the recall with a given precision threshold λ_p, i.e., the mappings that maximize the evaluation function $eval_{recall}$ presented earlier in this section. To do this, we used NOMADm[8], an implementation of the Mesh Adaptive Direct Search algorithm [1].

We repeated the above experiment by changing the value of the precision threshold λ_p from *0* to *1*. The recall of the results obtained using the mappings selected at every feedback iteration appears in Figure 4, and their precision appears in Figure 5.

The analysis of these results shows that both recall and precision quickly converge to the correct values, i.e., the ones computed when the mappings are selected based on complete knowledge of the correct result set. Specifically, *170* feedback instances (which represent *7.72%* of the result set retrieved by the candidate mappings) allowed identifying all the mappings that can be used to increase the recall of the results, and closely estimating the precision of the results obtained using these mappings.

During the first feedback iterations the recall and precision of the results obtained using mappings selected based on user feedback are different from the correct ones. Specifically, the recall of the results obtained when the precision threshold λ_p belongs to $\{0, 0.2, 0.5\}$, is lower than the correct one, when the number of feedback instances is less than *160*, i.e., 7.26% of the result set retrieved by the candidate mappings. This is because at this stage, the feedback instances supplied did not cover all the mappings that can be used to increase the recall of the results. On the other hand, the recall of the results obtained when the precision threshold λ_p belongs to $\{0.7, 1\}$, is greater than the actual recall, and the precision computed based on user feedback is lower than the specified threshold. This can be explained by the following:

- For $\lambda_p = 0.7$, the precision computed for the mappings selected when the number of feedback instances supplied is lower than *50*, i.e., 2.27% of the result set retrieved by the candidate mappings, was greater than the true precision. Afterwards, the search algorithm selected mappings that respected the precision threshold at the detriment of the recall, which was lower than the correct recall. The recall was then

[8]http://www.gerad.ca/NOMAD/Abramson/nomadm.html

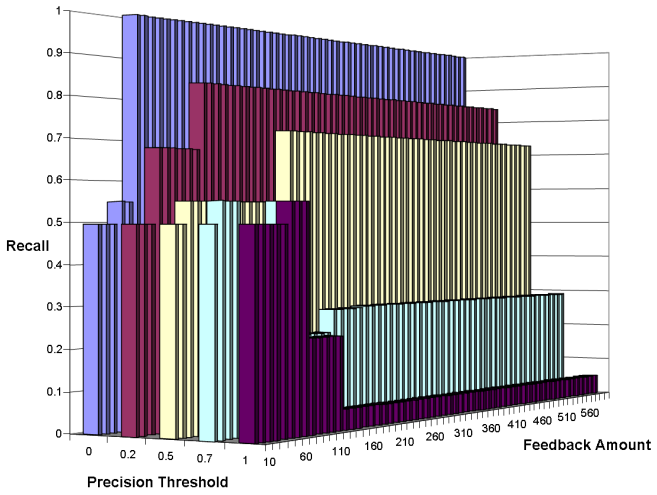

Figure 4: Recall of the results obtained using the mappings selected using the evaluation function $eval_{recall}$

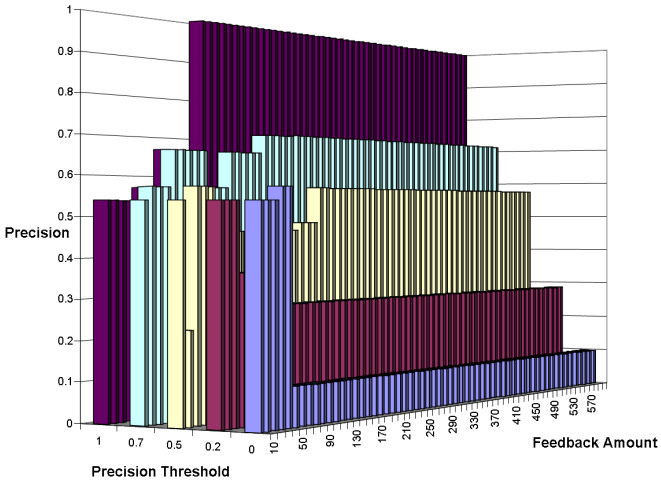

Figure 5: Precision of the results obtained using the mappings selected using the evaluation function $eval_{recall}$

Table 1: Number of mappings selected when the recall reaches the maximum

λ_p	0	0.2	0.5	0.7	1
Number of mappings selected	5	4	3	3	1

gradually corrected until it reached the actual recall, when the number of feedback instances supplied reached *170*, i.e., 7.72% of the result set retrieved by the candidate mappings.

- The precision threshold $\lambda_p = 1$ could not be initially satisfied since none of the mappings annotated using feedback have a precision of *1*. Later on, when the cumulative number of feedback instances reached *150*, i.e., 6.81% of the result set retrieved by the candidate mappings, the only mapping with a precision equal to *1* was identified, thereby allowing the solver to meet the constraint specified by the threshold to be satisfied.

It is worth mentioning that except for the case where λ_p equals *1*, the number of mappings required to reach the correct recall and precision ranges from *3* to *5* (see Table 1).

5. REFINING SCHEMA MAPPINGS

Mapping annotations may reveal that the quality of candidate mappings is poor. Specifically, they may indicate that the number of true positives obtained using the *best* mapping, i.e., the mapping with the highest F-measure, is small compared with the number of false positives. The quality of candidate mappings can be improved through refinement. In mapping refinement, one or more mappings are constructed out of existing ones in the light of user feedback. We distinguish two kinds of refinement: mapping refinement that seeks to reduce the number of false positives, and mapping refinement that aims to increase the number of true positives.

5.1 Refining Mappings to Reduce the Number of False Positives

A candidate mapping is refined by modifying a source query so that the number of false positives it returns is reduced. There are four operators in the relational algebra that allow filtering of the results of a query that can be used for this purpose, namely join, selection, intersection and difference.

Join.

Assume that in the user conceptualization of the world, the *Protein* relation in the integration schema is to be populated with tuples providing information about the proteins that belong to the Fruit Fly species, and consider the mapping $\langle Protein, ProteinEntry\rangle$, which maps the $Protein$ relation in the integration schema to the $ProteinEntry$ in the source schema illustrated in Figure 6. Using this mapping, the user will obtain some true positives, but also some false positives, viz., proteins that belong to species other than the Fruit Fly. The number of false positives returned may be reduced by joining the source query of the mapping with relations in the sources. To identify the source relations that can be used for this purpose, we exploit information provided by the source schemas. For example, Figure 6 shows that the *ProteinEntry* relation is connected to the *ChickenProteome* relation by a referential integrity constraint; this constraint can be used to identify the proteins that belong to the chicken proteome. Given that these proteins do not belong to the Fruit Fly species, we can reduce the number of false positives obtained using the above mapping, by ruling out the proteins in the *ChickenProteome* relation. The source query of the mapping obtained by this refinement is:

$$ProteinEntry - \Pi_{ProteinEntry.*}(ProteinEntry \bowtie_{acc=id} ChickenProteome)$$

Here, $ProteinEntry.*$ denotes the list of attributes of the $ProteinEntry$ relation. This style of refinement is based on input information that is readily available in the source schemas, viz., referential integrity constraints.

To specify which relations are to be joined with the source query of a given mapping, we use the notion of a *path*. A path p can be defined as a sequence:

$$p = (\langle R_1, att_1^{out}\rangle, \ldots, \langle R_i, att_i^{in}\rangle, \langle R_i, att_i^{out}\rangle, \ldots, \langle R_n, att_n^{in}\rangle)$$

where $n \geq 2$, and att_i^{in} and att_i^{out} are attributes of the relation R_i. Let m be a mapping and $\langle R_1, att_1^{out}\rangle$ an attribute that is involved

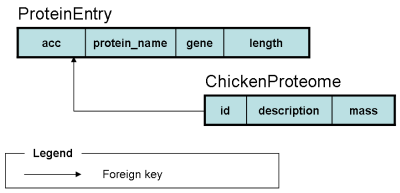

Figure 6: Source schema used for mapping refinement

in the source query of m (i.e., an attribute of a relation in the from clause of the query). To refine the source query of m using the path p, we use the function $constructJoinMapping(m, p)$, which returns a mapping, the query of which is obtained by joining the source query of *m* with the relations in *p* (except R_1), and projecting the values of the attributes of *m.source*. That is:
$constructJoinMapping(m, p) =$
$\langle m.integration, \Pi_{m.source.*}(m.source \bowtie_{jp_1} \ldots \bowtie_{jp_{n-1}} R_n)\rangle$
where $m.source.*$ denotes the list of attributes of $m.source$, and $jp_i = \text{“}(R_i.att_i^{out} = R_{i+1}.att_{i+1}^{in})\text{”}$, $1 \leq i \leq n-1$.

Note that only a subset of the paths that contain an attribute involved in the source query of the mapping m are useful for refinement; not every path reduces the number of false positives retrieved using m. Also, a path that reduces the number of false positives retrieved by the mapping m may reduce the number of true positives as well. We use the F-measure to quantify whether the decrease in terms of false positives outweighs the loss in terms of true positives.

Difference.

The number of false positives retrieved by a mapping *m* for populating *r*, a relation in the integration schema, may be reduced by applying the difference operator between the source query of *m* and a query that returns a subset of the false positives returned by *m.source*. That is:

$$m.source \leftarrow m.source - q_s$$

where q_s is a query over the sources that is union compatible with $m.source$. To identify q_s, we exploit the fact that there are multiple candidate mappings for populating *r*. In particular, if there is a mapping *m'* which is a candidate for populating *r* that is known to return some false positives that are retrieved using the mapping *m*, then the source query of *m'* can play the role of q_s. That is:

$$m.source \leftarrow m.source - m'.source$$

Intersection.

If there is a mapping *m'* that is known to have true positives in common with the mapping *m*, then *m* can be refined by applying the intersection operator between the source query of *m* and the source query of *m'*. That is:

$$m.source \leftarrow m.source \cap m'.source$$

Note that this refinement may lead to a loss of true positives that are not retrieved by m'. To avoid this, we can make use of ordinal annotations by using only the candidate mappings that are known to cover m in terms of true positives.

Selection.

The number of false positives returned by a given candidate mapping *m* can be reduced by applying a selection to its source query. That is:

$$m.source \leftarrow \sigma_C\ m.source$$

The problem is, of course, that of identifying a selection condition *C* with which the number of false positives can be reduced. This condition can be specified based on collected user feedback. Consider, for example, that the user is searching for proteins that belong to the Uniprot database[9]. If the user supplied feedback specifying the attribute *accession* of the *Protein* relation cannot have the values *X51342* and *AA6513* then we can specify a selection condition that rules out all the tuples in which the *accession* takes either of these values. The down side of this technique is that it may result in predicates that are over-fitted to the specific feedback received; the number of conjuncts in the predicate obtained is equal to the number of incorrect attribute values. To avoid this problem, we may be able to use of an external source of information for specifying selection conditions, e.g., ontologies that encode the application domains that are of interest to the user. An ontology is a set of concepts and relationships between them [11]. The concepts of a domain ontology are, in certain cases, associated with recognizers [7]. Typically, a recognizer is a regular expression using which it is possible to determine whether a given object is an instance of the concept in question [15]. If these recognizers are available, then they can be used to specify selection predicates for reducing the number of false positives. Consider, for example, an ontology that encodes the domain of bioinformatics, e.g., the myGrid ontology[10], and consider that this ontology contains a concept associated with the following regular expression: $re = \text{‘}[A-N, R-Z][0-9][A-Z][A-Z, 0-9][A-Z, 0-9][0-9]\text{’}$. This expression specifies the format of Uniprot accession numbers. By matching the values of the attribute *accession* against this regular expression, we will find out that the true positives match the expression whereas the false positives do not. In other words, the concept associated with *re* captures the semantic domain of the *accession* attribute. Therefore, a condition that can be used to rule out the proteins that do not belong to the *Uniprot* database is a predicate that is true when the value of the *accession* attribute matches the regular expression *re*.

5.2 Refining Mappings to Increase the Number of True Positives

To increase the number of true positives returned by a mapping *m* that is a candidate for populating a relation *r*, we union its source query with a query over the sources that is known to retrieve true positives that are not returned by *m*. That is:

$$m.source \leftarrow m.source \cup q_s$$

where q_s is a query over the sources that is union compatible with $m.source$. Here again, we use the fact that there are multiple candidate mappings for *r*, and union $m.source$ with the source query of a mapping *m'* that is known to retrieve true positives that are not retrieved by m. That is:

$$m.source \leftarrow m.source \cup m'.source$$

Notice that the increase in terms of true positives may be accompanied with an increase in terms of false positives. Once again, we use the F-measure to establish whether the increase in terms of recall outweighs the decrease in terms of precision that may have occurred.

Relaxing a selection condition is another means that can be used for augmenting the number of true positives retrieved using *m*. Consider that the source query of *m* is of the form $\sigma_C\ q_s$, where q_s is a

[9]http://www.uniprot.org
[10]www.mygrid.org.uk

query over the sources. The number of true positives retrieved by *m* may be increased by relaxing the selection condition *C*, by replacing *C* with a less stringent condition *C'*: $C \Rightarrow C'$. In its simplistic form, the relaxation consists in replacing *C* with $\top$, a predicate that always evaluates to *true*. In other words, $m.source \leftarrow q_s$.

5.3 Exploring the Space of Refined Mappings

The space of mappings that can be obtained by refinement is very large. The refinement operations can potentially be composed and recursively applied. For example, a refined mapping can be obtained by unioning the source query of a candidate mapping with the query obtained by joining the source query of another candidate mapping with some source relations. Also, a query obtained by joining the source query of a candidate mapping with some source relations, can be joined with other source relations. This raises the question as to how to explore the space of potential mappings that can be constructed by refinement, with the objective of discovering the *best* possible mapping(s). An exhaustive enumeration of all potential mappings is likely to be too expensive. In this section, we present an evolutionary algorithm for exploring the space of refined mappings in a bounded time.

Evolutionary algorithms are heuristics for solving combinatorial optimization problems [20]; the literature is rich with combinatorial optimization algorithms [3]. We chose to use an evolutionary algorithm because it is a population-based approach: unlike point-based algorithms, which process a single solution, population-based algorithms process and maintain multiple competing solutions at a time. This feature fits our purpose since we aim to process an initial set of candidate mappings to create a new set of (hopefully) better candidate mappings. Figure 7 presents the algorithm used for refining candidate mappings. The mappings given as input are iteratively refined. At each iteration, a subset of candidate mappings is selected *(line 2)*; the mappings with F-measures greater than a given threshold. This threshold can be specified either manually or based on the F-measure of the initial set of candidate mappings, e.g., the threshold can be set as the F-measure of the top mapping in the initial set of candidate mappings. Variation operators are then applied to the mappings selected to derive new mappings. To effectively and efficiently explore the space of mappings obtained by refinement, two kinds of variation operator are used, namely *cross over* and *mutation* operators *(lines 3 and 4)*. Cross over operators are used to construct new mappings by combining the *good parts* of existing mappings. Mutation operators, on the other hand, are used to avoid the premature convergence towards a sub-optimum solution by diversifying the space of candidate mappings to be explored [3].

The mappings constructed using cross over and mutation are then annotated based on collected feedback *(line 5)*, before the *next generation* of candidate mappings for the next iteration is selected *(line 6)*. Multiple schemes can be used for selecting candidate mappings. For example, the top-k mappings in terms of F-measure can be used in the next iteration. Note, however, that a mapping with a low F-measure can be crucial for constructing the best mapping(s): this is the case, for instance, of a mapping that retrieves true positives (or false positives) that are not retrieved by other mappings. Because of this, we also use, in addition to the top-k mappings, a minimal subset of mappings that cover all the true positives and false positives identified through user feedback.

The process presented above is repeated until a termination condition is met, e.g., when the time allocated for refinement expires, or when a candidate mapping with an F-measure higher than a given threshold, e.g., 0.95, is discovered.

Mutating a Candidate Mapping. A mapping is mutated by applying the join or selection relational algebra operators to their source queries. Figure 8 illustrates the subroutine used for mutating schema mappings. Given a mapping *m*, a path *p* that originates from an attribute of a relation that is involved in the source query of *m* is selected *(line 2)*. The mapping obtained by joining the source query of *m* with the relations in *p* is added to the set of mutated mappings *(line 3)*.

Algorithm RefineMappings
Inputs Map : A set of candidate mappings
UF: A set of user feedback instances
Outputs Map : A set of candidate mappings
Begin
1 **While** Termination condition not met **Do**
2 $S\_Map \leftarrow SelectMappings(Map)$
3 $M\_Map \leftarrow MutateMappings(S\_Map)$
4 $C\_Map \leftarrow CrossOverMappings(S\_Map, Map, UF)$
5 $AnnotateMappings(M\_Map \cup C\_Map, UF)$
6 $Map \leftarrow$
$SelectNewGeneration(Map \cup M\_Map \cup C\_Map)$
7 **Return** Map
End

Figure 7: Evolutionary Algorithm for Refining Candidate Mappings

Note that the number of paths that originate from an attribute involved in the source query of m can be large, even infinite if we consider cyclic paths. To reduce the number of paths to be explored, we exploit the dependencies between paths. To illustrate this idea, consider the path p such that:

$$p = (\langle R_1, att_1^{out}\rangle, \ldots, \langle R_i, att_i^{in}\rangle, \langle R_i, att_i^{out}\rangle, \ldots, \langle R_n, att_n^{in}\rangle)$$

and, let *p'* be a path that originates from the same attribute as *p*, and covers *p*, i.e., *p'* contains the sequence of attributes in *p*:

$$p' = (\langle R_1, att_1^{out}\rangle, \ldots, \langle R_n, att_n^{in}\rangle, \langle R_n, att_n^{out}\rangle, \ldots, \langle R_k, att_k^{in}\rangle)$$

The source query of the mapping *m'*$_r$, obtained by mutating *m* using *p'*, is equivalent to the join of the source query of the mapping m_r, constructed by mutating *m* using *p*, with a query over the sources. Specifically:

$$constructJoinMapping(m, p').source = constructJoinMapping(m, p).source \bowtie_{jp_{n+1}} \ldots \bowtie_{jp_k} R_k$$

where $jp_i, n \leq i \leq k-1$, is a predicate of the form: "$R_i.att_i^{out} = R_{i+1}.att_{i+1}^{in}$".

Therefore, if m_r does not retrieve any true or false positives, in which case it is of no relevance to the refinement process, then *m'*$_r$ will not retrieve any true or false positives. We exploit this observation to reduce the number of paths to consider in the next iterations for refining the mapping *m*. Specifically, if the annotations computed by the main algorithm (Figure 7, line 5) show that the mapping m_r does not return any true or false positives then none of the paths that cover *p*, including *p*, will be used for mutating *m* in in the next iterations of the refinement algorithm. As well as *m*, these paths will not be used in the next iterations for mutating any mapping *m'* with a source query contained within the source query of *m*. The above behavior is ensured by the function *getPath* (Figure 8, line 2).

Mappings can also be mutated by applying a selection to their source queries. If an ontology θ_{domain} that captures the application domain of the integration schema is available, and a selection predicate *prec* can be derived based on user feedback *(lines 4,5)*, then a mapping that is obtained by augmenting the source query of *m* with a selection using predicate *prec*, is added to the set of mutated mappings *(line 6)*.

Algorithm MutateMappings
Inputs S_Map : A set of candidate mappings
Outputs M_Map : A set of mutated mappings
Begin
1 **For Each** $m \in S\_Map$ **Do**
2 $p \leftarrow getPath(m)$
3 **Add** $constructJoinMapping(m, p)$ **To** M_Map
4 $prec \leftarrow getPredicate(m, \theta_{domain})$
5 **If** $(prec \neq null)$
6 **Add** $\langle m.integration, \sigma_{prec} m.source \rangle$ **To** M_Map
7 **Return** S_Map
End

Figure 8: Algorithm for Mutating Mappings

Combining Candidate Mappings. Two mappings are crossed over by applying the union, intersection or difference relational operators to their source queries. Figure 9 lists the algorithm used for crossing over candidate mappings. Given a mapping that is provided as input and a cross over operator, e.g., union, we identify candidate mappings that can act as recombination mappings, a.k.a. neighbors. To identify the neighbors of a mapping m, we use the ordinal annotations associated with candidate mappings (see Section 3). Consider, for example, the case of union. This operator is used to increase the number of true positives returned by a mapping m. The neighbors of m w.r.t. union are, therefore, the mappings that return true positives that are not retrieved by m. We reduce the set of neighbor mappings by considering only the mappings that are not covered by others in terms of true positives. That is:

$$union\_neighbors(m, Map, UF) = \{m_i \neq m \in Map \; s.t. \; (tp(m_i, UF) - tp(m, UF) \neq \emptyset) \; and \; (\nexists \, m_j \in Map, \; m_i <_{TP}^{UF} m_j)\}$$

Unlike union, the intersection and difference operators are used for reducing the number of false positives returned by a given mapping. The neighbors of a mapping m w.r.t. intersection are those that cover m in terms of true positives but not in terms of false positives (see below). The queries obtained by constructing the intersection of the source query of each of the neighbor mappings with the source query of m, retrieve all the true positives obtained using m and a subset of the false positives obtained using m. In other words, it allows an increase in precision without a reduction in the recall.

$$intersection\_neighbors(m, Map, UF) = \{m_i \neq m \in Map \; s.t. \; (m \leq_{TP}^{UF} m_i) \; and \; (m \not\leq_{FP}^{UF} m_i)\}$$

Similarly, the neighboring mappings of m w.r.t. difference are the mappings that cover m in terms of false positives but not in terms of true positives. That is:

$$difference\_neighbors(m, Map, UF) = \{m_i \neq m \in Map \; s.t. \; (m \leq_{FP}^{UF} m_i) \; and \; (m \not\leq_{TP}^{UF} m_i)\}$$

The query obtained by applying a difference operator to the source query of m and the source query of one of its neighbors m_i, i.e., $m.source - m_i.source$, does not return any false positive obtained using m. Note, however, that such queries may return a subset of the true positives obtained using m, i.e., the recall of the resulting mapping may be lower than that of m.

Note that some of the mappings obtained by mutation and crossover may have lower quality than the candidate mapping used as input. These mappings will simply be discarded when selecting the candidate mappings to be processed in the next iteration of refinement *(Figure 7, line 6)*.

Algorithm CrossOverMappings
Inputs S_Map : A set of candidate mappings
Map : A set of candidate mappings
UF : A set of feedback instances
Outputs C_Map : A new set of mappings
Begin
1 **For Each** $m \in S\_Map$ **Do**
2 $Union\_Map \leftarrow union\_neighbors(m, Map, UF)$
3 **For Each** $u\_m \in Union\_Map$ **Do**
4 **Add** $\langle m.integration, m.source \cup u\_m.source \rangle$ **To** C_Map
5 $Inter\_Map \leftarrow intersection\_neighbors(m, Map, UF)$
6 **For Each** $i\_m \in Inter\_Map$ **Do**
7 **Add** $\langle m.integration, m.source \cap i\_m.source \rangle$ **To** C_Map
8 $Diff\_Map \leftarrow difference\_neighbors(m, Map, UF)$
9 **For Each** $d\_m \in Diff\_Map$ **Do**
10 **Add** $\langle m.integration, m.source - d\_m.source \rangle$ **To** C_Map
11 **Return** C_Map
End

Figure 9: Algorithm for Combining Mappings

5.4 Experimental Evaluation

The approach we described above for mapping refinement raises the following questions: *Can mapping refinement improve the quality of initial candidate mappings, and, if so, at what cost, i.e., what is the amount of user feedback required?*

To answer the above questions, we conducted an experiment in which the candidate mappings for the *FavoriteCity* relation are refined using the *RefineMappings* algorithm in the light of user feedback. Specifically, we iterated over the process listed below until the F-measure of the top mapping constructed through refinement reaches the maximum, i.e., *1*.

1. Generate *10* feedback instances.
2. Annotate the set of candidate mappings.
3. Refine candidate mappings using the *RefineMappings* algorithm.

Regarding the *RefineMappings* algorithm, we chose the following setup. At every iteration in the algorithm, the top three mappings with the best F-measure are selected for constructing new *offspring* mappings (Figure 7, line 2). The algorithm iterates until the population of mappings remains unchanged for *10* consecutive iterations (Figure 7, line 1). If a newly constructed mapping returns the same result set as an existing mapping, then it is removed (Figure 7, line 6). We also considered the general case in which no domain ontology that captures the domain of the source schemas is known, and, therefore, the selection conditions used for mutating the mappings are derived based on the tuples annotated through the feedback.

We repeated the above experiment *10* times. In each repetition, we specified the set of correct tuples based on a mapping that was randomly created by mutating and combining candidate mappings using the selection, join, intersection, union and difference relational operators.

The results of this experiment are shown in Figure 10, which shows the F-measure of the top, second and third mappings constructed through refinement at every feedback iteration averaged over the *10* runs.

The figure shows that the quality of the top mappings improves substantially during the first few feedback iterations. The average F-measure of the top mapping which initially was equal to 0.58 increases to 0.8 after the 5th feedback iteration. That is, after collecting 50 feedback instances, which represents 2.27% of the result set retrieved by the candidate mappings. It then increases to

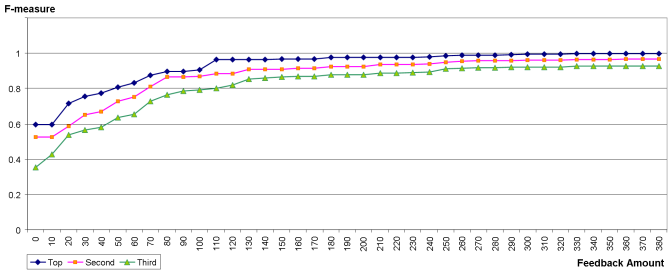

Figure 10: Average F-measure of the top, second and third mapping obtained by refinement

Table 2: Average percentage of the result set that needed to be annotated by feedback to reach a given F-measure

mapping	top mapping	second mapping	third mapping
F-measure $\geq$ 0.7	0.9	2.27	3.18
F-measure $\geq$ 0.8	2.27	3.18	4.99
F-measure $\geq$ 0.9	3.63	5.9	11.36
F-measure $\geq$ 0.95	4.99	11.81	NA
F-measure $\geq$ 0.99	13.17	NA	NA
F-measure = 1	16.36	NA	NA

0.9 after collecting 80 feedback instances, i.e., 3.63% of the result set retrieved by the candidate mappings. On the other hand, the number of feedback instances required to reach the maximum F-measure value, i.e., *1*, is important. Specifically, *360* feedback instances, i.e., 16.36% of the result set retrieved by the candidate mappings, were needed for the F-measure of the top mapping to reach the maximum value. This observation can be explained by the following. If the F-measure of a mapping is high, that is close to *1*, then the number of expected results that are not returned by the mapping and the number of unexpected results returned by the mapping is small. Therefore, there is a small chance that the feedback instances supplied cover the few expected tuples the mapping does not return or the few unexpected tuples that the mapping does return. Because of this, the chances of improving the quality of such a mapping are low in principle.

The F-measure of the second and third mappings follows a pattern similar to that of the top mapping, and the above observation applies to them as well (see Table 2).

In summary, this experiment shows that refinement can construct good quality mappings in a pay-as-you-go fashion; the more feedback instances are provided, the better the mappings constructed. The experiment also shows that refinement is more cost effective during the early feedback iterations. The quality of the mapping constructed improves substantially during the first feedback iterations, whereas the number of feedback required to reach an F-measure that is close to the maximum value was, in this example, almost four times larger for only a small increment in the F-measure value.

6. RELATED WORK

While schema mappings can be automatically derived using existing mapping generation techniques [21, 27], the mappings outputs by these techniques may not conform to users' expectations. Some researchers attempted to address the issue of mapping verification within the context of data exchange. Chiticariu *et al.* [5] proposed a debugger for understanding and exploring schema mappings. To do this, they compute, and display on request, the relationships, termed *routes*, between source and target data with the schema mapping in question. Bonifati *et al.* proposed Spicy [4], a system for verifying the quality of mappings between a source and target schema. To verify a collection of schema mappings, their source queries are issued against the source schema and the results obtained are compared with instances from the target schema, the contents of which are assumed to be available. The results of this comparison are meant to identify incorrect mappings, and to suggest to designers the mappings that are likely to be accurate.

Using the above tools, the verification of schema mappings takes place before the data integration system is setup, potentially incurring a considerable up-front cost [9, 12]. Differently, our proposal falls under the dataspaces vision, since we seek to annotate and refine the candidate mappings as the data integration proceeds incrementally. In this regard, the work by McCann *et al.* [18] is similar to ours; they developed a community-based approach that solicits feedback from the multitude of community members. The objective of the proposal by McCann *et al.* is, however, different from ours. Their aim is to inform the schema matching operation based on user feedback. In doing so, the feedback is used to assess the matches between attributes in two schemas. For example, user feedback can be used to verify the data type of an attribute (e.g., month), or the validity of a domain constraint (e.g., the value of an attribute is always less that the value of another). Differently, in our work, we seek to assess the quality of executable mappings that are candidates for populating the elements of an integration schema.

Jeffery *et al.* [14] developed a decision-theoretic framework for specifying the order in which candidate mappings can be confirmed by soliciting feedback from users with the objective of providing the *most benefit* to a dataspace. Our proposal is different from this work in the following respects. Firstly, Jeffery *et al.* assume that a mapping is either *correct* or *incorrect*. As we mentioned earlier, if the initial set of candidate mappings is not complete, i.e., does not contain a candidate mapping that meets the *exact* expectations of users, all the candidate mappings will ultimately be found to be incorrect. Because of this, we opted for a finer-grained annotation scheme that order candidate mappings and label them with metrics specifying their precision and recall. Secondly, Jeffery *et al.* do not specify the means by which feedback instances are collected. In our proposal, we showed how feedback can be provided by users by examining the results to queries that they issued, and which were evaluated using the candidate mappings. Thirdly, Jeffery *et al.* [14] did not address the problems of mapping selection or refinement.

Talkudar *et al.* presented a system for assisting users in authoring queries [24]. Similar to our approach, this system uses as input feedback provided by end users about the results obtained using the candidate queries. The solution proposed by Talkudar *et al.* is, however, different from ours. They use feedback to rank candidate queries. Differently, the annotations we compute estimate the precision and recall of candidate mappings, thereby opening the door to tailorable selection, i.e., one that seeks to meet specific user requirements in terms of these criteria. In addition to ranking candidate mappings, these annotations allow the quality of each candidate mapping to be measured. Also, the assumption underlying the proposal by Talkudar *et al.* is different from ours. Talkudar *et al.* assume that a candidate query returns few answers, and that feedback about a given tuple can be propagated to all the tuples of the same query. We do not make this assumption in our work; the same candidate mapping can produce tuples that meet users expectation (true positives), and others that do not (false positives).

Regarding mapping refinement, our proposal is inspired to some extent by the work by Yan *et al.* [28] and Alexe *et al.* [2] in that we

use information provided by the source schemas. Their approach is, however, different from ours. In particular, in the above proposals the mappings that do not meet users expectations are considered incorrect, and are ruled out. Differently, in our proposal, the refinement does not seek to remove "bad" candidate mappings; all candidate mappings, including those that are known to return false positives, co-exist. Instead, the refinement operation seeks to create new better quality candidate mappings from existing ones by iteratively mutating and crossing over the mappings in the initial set of mappings derived using existing generation techniques [21, 27].

7. CONCLUSIONS

In this paper, we explored the use of feedback supplied by end users for annotating, selecting and refining schema mappings in the context of dataspaces. We showed how schema mappings can be incrementally annotated with metrics that estimate the precision and recall of the results they retrieve based on feedback supplied by end users. We also presented a method for selecting mappings for populating an element of the integration schema that respond to user needs. This method casts the problem of mapping selection as an optimization problem that we solve using the mesh adaptive direct search algorithm. We also showed how better quality mappings can be constructed from an initial set of candidate mappings through refinement by using an evolutionary algorithm. The results of evaluation exercises showed the effectiveness of our solution. They demonstrated the *pay-as-you-go* aspect of our approach: the more feedback the user supplies, the better the outputs of mapping annotation, selection and refinement. The evaluation exercises also showed that mapping annotation and refinement are more cost effective in the first feedback iterations; using feedback about a small proportion of the results returned by the candidate mappings, the results of mapping annotation and refinement are close to the gold standard ones.

In addition to the issues tackled in this paper, there are several problems that need to be addressed to realise the dataspace vision. In particular, we are investigating as part of our ongoing work, inconsistencies that may exist in user feedback, which may occur because of changes in user expectations for example, and are analysing the impact such inconsistencies can have on mapping annotations of the form presented in this paper.

8. REFERENCES

[1] M. A. Abramson, C. Audet, and J. E. Dennis. Nonlinear programing with mesh adaptive direct searches. *SIAG/Optimization Views-and-News*, 17(1):2–11, 2006.

[2] B. Alexe, L. Chiticariu, R. J. Miller, and W. C. Tan. Muse: Mapping understanding and design by example. In *ICDE*, pages 10–19. IEEE, 2008.

[3] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[4] A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa. Schema mapping verification: the spicy way. In *EDBT*, pages 85–96. ACM, 2008.

[5] L. Chiticariu and W. C. Tan. Debugging schema mappings with routes. In *VLDB*, pages 79–90. ACM, 2006.

[6] X. L. Dong, A. Y. Halevy, and C. Yu. Data integration with uncertainty. *VLDB J.*, 18(2):469–500, 2009.

[7] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, Y.-K. Ng, D. Quass, and R. D. Smith. Conceptual model based data extraction from multiple-record web pages. *Data Knowl. Eng.*, 31(3):227–251, 1999.

[8] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[9] M. J. Franklin, A. Y. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

[10] A. Gal. Why is schema matching tough and what can we do about it? *SIGMOD Record*, 35(4):2–5, 2006.

[11] T. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[12] A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of dataspace systems. In S. Vansummeren, editor, *SIGACT-SIGMOD-SIGART*, pages 1–9. ACM, 2006.

[13] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the deep web. *Commun. ACM*, 50(5):94–101, 2007.

[14] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, pages 847–860. ACM, 2008.

[15] R. Kaschek and S. Zlatkin. Where ontology affects information systems. In *ISTA*, pages 35–46. GI, 2003.

[16] M. Lenzerini. Data integration: A theoretical perspective. In L. Popa, editor, *PODS*, pages 233–246. ACM, 2002.

[17] R. McCann, B. K. AlShebli, Q. Le, H. Nguyen, L. Vu, and A. Doan. Mapping maintenance for data integration systems. In *VLDB*, pages 1018–1030, 2005.

[18] R. McCann, A. Kramnik, W. Shen, V. Varadarajan, O. Sobulo, and A. Doan. Integrating data from disparate sources: A mass collaboration approach. In *ICDE*, pages 487–488. IEEE CS, 2005.

[19] D. A. Menascé and V. Dubey. Utility-based qos brokering in service oriented architectures. In *ICWS*, pages 422–430. IEEE CS, 2007.

[20] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, December 2004.

[21] R. J. Miller, L. M. Haas, and M. A. Hernández. Schema mapping as query discovery. In *VLDB*, pages 77–88, 2000.

[22] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[23] I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems. *Knowl. Eng. Rev.*, 18(2):95–145, 2003.

[24] P. P. Talukdar, M. Jacob, M. S. Mehmood, K. Crammer, Z. G. Ives, F. Pereira, and S. Guha. Learning to create data-integrating queries. *PVLDB*, 1(1):785–796, 2008.

[25] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[26] R. S. Witte and J. s. Witte. *Statistics*. Horcourt College Publishers, 2002.

[27] L. Xu and D. W. Embley. A composite approach to automating direct and indirect schema mappings. *Inf. Syst.*, 31(8):697–732, 2006.

[28] L.-L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. In *SIGMOD Conference*, pages 485–496, 2001.

[29] L. Zamboulis, H. Fan, K. Belhajjame, J. A. Siepen, A. C. Jones, N. J. Martin, A. Poulovassilis, S. J. Hubbard, S. M. Embury, and N. W. Paton. Data access and integration in the ispider proteomics grid. In *DILS*, pages 3–18. Springer, 2006.