Soliciting User Feedback in a Dataspace System

Shawn Jeffery
Michael Franklin
Alon Halevy

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-38
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-38.html

March 27, 2007

Soliciting User Feedback in a Dataspace System

Shawn R. Jeffery
UC Berkeley
jeffery@cs.berkeley.edu

Michael J. Franklin
UC Berkeley
franklin@cs.berkeley.edu

Alon Y. Halevy
Google Inc.
halevy@google.com

ABSTRACT

A primary challenge to large-scale data integration is creating semantic equivalences between elements from different data sources that correspond to the same real-world entity or concept. Dataspaces propose a pay-as-you-go approach: automated mechanisms such as schema matching and reference reconciliation provide a initial correspondences, termed *candidate matches*, and then user feedback is used to incrementally confirm these matches. The key to this approach is to determine in what order to solicit user feedback for confirming candidate matches.

In this paper, we develop a decision-theoretic framework for ordering candidate matches for user confirmation using the concept of the *value of perfect information* (*VPI*). At the core of this concept is a *utility function* that quantifies the desirability of a given state; thus, we devise a utility function for dataspaces based on query result quality. We show in practice how to efficiently apply VPI in concert with this utility function to order user confirmations. A detailed experimental evaluation shows that the ordering of user feedback produced by this VPI-based approach yields a dataspace with a significantly higher utility than a wide range of other ordering strategies. Finally, we outline the design of Roomba, a system that incorporates this decision-theoretic framework to guide a dataspace in soliciting user feedback in a pay-as-you-go manner.

1. INTRODUCTION

As the amount and complexity of structured data increases in a variety of applications, such as enterprise data management, large-scale scientific collaborations [25], sensor deployments [26], and an increasingly structured Web [15], there is a growing need to provide unified access to these heterogeneous data sources. Dataspaces [10] provide a powerful abstraction for accessing, understanding, managing, and querying this wealth of data by encompassing multiple data sources and organizing their data over time in an incremental, "pay-as-you-go" fashion.

One of the primary challenges facing dataspace systems is large-scale semantic data integration [8]. Heterogeneous data originating from disparate sources may use different representations of the same real-world entity or concept. For example, two employee records in two different enterprise databases may refer to the same person. Similarly, on the Web there are multiple ways of referring to the same product or person, for instance. Typically, a DataSpace Support Platform (DSSP) employs a set of mechanisms for semantic integration, such as schema matching [20] and entity resolution [6], to determine semantic equivalences between elements in the dataspace. The output of these mechanisms are a set of *candidate matches* that state with some confidence that two elements in the dataspace refer to the same real-world entity or concept.

To provide more accurate query results in a dataspace system, candidate matches should be confirmed by soliciting user feedback. Since there are far too many candidate matches that could benefit from user feedback, a system cannot possibly involve the user in all of them. Here is where the pay-as-you-go principle applies: the system incrementally understands and integrates the data over time by asking users to confirm matches as the system runs. One of the main challenges for soliciting user feedback in such a system is to determine in what *order* to confirm candidate matches. In fact, this is a common challenge in a set of recent scenarios where the goal is to leverage mass collaboration, or the so-called *wisdom of crowds* [24], in order to better understand sets of data [28, 17, 9].

In this paper, we consider the problem of determining the order in which to confirm candidate matches to provide the *most benefit* to a dataspace. To this end, we apply decision theory to the context of data integration to reason about data integration tasks in a principled manner.

We begin by developing a method for ordering candidate matches for user confirmation using the decision-theoretic concept of the *value of perfect information* (*VPI*) [21]. VPI provides a means of estimating the benefit to the dataspace of determining the correctness of a candidate match through soliciting user feedback. One of the key advantages of our method is that it considers candidate matches produced from *multiple* mechanisms in a uniform fashion. Hence, the system can weigh, for example, the benefit of asking to confirm a schema match versus confirming the identity of references to two objects in the domain. In this regard, our work is distinguished from previous research in soliciting user feedback for data integration tasks [22, 29, 7] that are tightly integrated with an individual mechanism (i.e., schema matching or object matching).

At the core of VPI is a *utility function* that quantifies the desirability of a given state of a dataspace; thus, we devise a utility function for dataspaces based on query result quality. Since the exact utility of a dataspace is impossible to know as the DSSP does not know the correctness of the candidate matches, we develop a set of approximations that allow the DSSP to efficiently estimate the utility of a dataspace.

We describe a detailed experimental evaluation showing that the ordering of user feedback produced by our VPI-

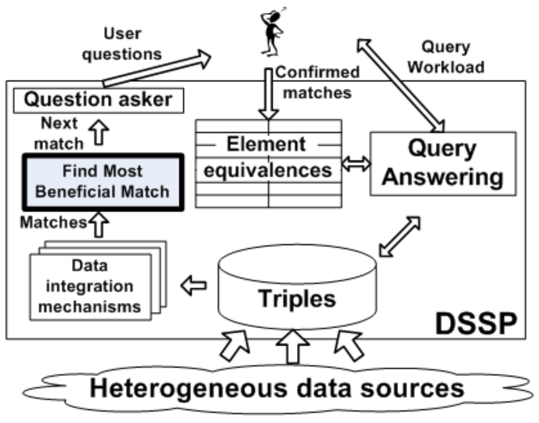

Figure 1: Overall setup

based approach yields a dataspace with a significantly higher utility than a variety of other ordering strategies. Moreover, in the experiments the utility produced by our approach is shown to be within 5% of that produced by an oracle strategy that knows all correct matches and query results. We also illustrate experimentally the benefit of considering multiple schema integration mechanisms uniformly: we show that an ordering approach that treats each class of mechanism separately for user-feedback purposes yields poor overall utility. Furthermore, our experiments explore various characteristics of data integration environments to provide insights as to the effect of environmental properties on the efficacy of user feedback.

Finally, we outline the design of Roomba, a system that incorporates this decision-theoretic framework to guide a dataspace in soliciting user feedback in a pay-as-you-go manner.

This paper is organized as follows. Section 2 describes our terminology and problem setting. Section 3 discusses our decision-theoretic framework for ordering match confirmations. Section 4 presents a detailed empirical evaluation of our match ordering strategy. In Section 5 we show how to relax the query answering model to consider unconfirmed matches. We describe Roomba in Section 6. Section 7 presents related work. Section 8 presents our conclusions.

2. PRELIMINARIES

We begin by describing our problem setting and defining the terms we use throughout the paper. Our overall setup is shown in Figure 1

Dataspace Triples

We model a dataspace D as a collection of *triples* of the form $\langle object, attribute, value\rangle$ (see Table 1 for an example).

Objects and attributes are represented as strings. Values can be strings or can come from several other domains (e.g., numbers or dates). Intuitively, an object refers to some real-world entity and a triple describes the value of some attribute of that entity. Triples can also be thought of as representing rows in binary relations, where the attribute is the table name, the first column is the object, and the second column is the value. Of course, in a dataspace, we do not know the set of relations in advance.

We do not assume that the data in the dataspace is stored as triples. Instead, the triples may be a logical view over multiple sets of data residing in independent systems.

We use the term *element* to refer to anything that is either an object, attribute, or value in the dataspace. Note that the sets of strings used for objects, attributes, and values are *not* necessarily disjoint.

	⟨**object**, **attribute**, **value**⟩
t_0 =	⟨Wisconsin, SchoolColor, Cardinal⟩
t_1 =	⟨Cal, SchoolColor, Blue⟩
t_2 =	⟨Washington, SchoolColor, Purple⟩
t_3 =	⟨Berkeley, Color, Navy⟩
t_4 =	⟨UW-Madison, Color, Red⟩
t_5 =	⟨Stanford, Color, Cardinal⟩

Table 1: An example dataspace

EXAMPLE 2.1. *The example in Table 1 shows a dataspace with six triples, describing properties of universities.* □

Dataspace Heterogeneity and Candidate Matches

Since the data in a dataspace come from multiple autonomous sources, they display a high degree of heterogeneity. For example, the triples in the dataspace could be collected from a set of databases in an enterprise or from a large collection of tables on the Web. As a result, different strings in a dataspace do not necessarily denote different objects or attributes in the real world. In our example, the objects "Wisconsin" and "UW-Madison" refer to the same university, the attributes "Color" and "SchoolColor" describe the same property of universities, and the values "cardinal" and "red" are the same in reality.

We assume that there is a set of *mechanisms* that try to identify such equivalences between elements. In particular, there are techniques for schema matching that predict whether two attributes are the same [20], and there are techniques for entity resolution (also referred to as object deduplication) that will predict whether two object or value references are about the same real-world entity [6].

We model the output of these mechanisms as a set of *candidate matches*, each of the form (e_1, e_2, c), where e_1 and e_2 are elements in the dataspace and c is a number between 0 and 1 denoting the confidence the mechanism has in its prediction. In this paper, we are agnostic to the details of the mechanisms.

While in some cases the mechanisms can predict equivalence between elements with complete confidence, most of the time they cannot. Since query processing in a dataspace system depends on the quality of these matches, query results will be better when the matches are certain. Thus, our goal is to solicit feedback from users to *confirm* the matches produced by the mechanisms. Confirming a candidate match involves posing a question about the match to the user that can be answered with a "yes" or a "no". We assume there exists a separate component that can translate a given candidate match into a such a question.

Ideally, the system should to ask user to confirm most or even all uncertain matches. This approach, however, is not feasible given the scale and nature of large-scale data integration. The number of candidate matches is potentially large and users may find it inconvenient to be inundated with confirmation requests. Hence, our approach is to confirm matches in a *pay-as-you-go* manner [10], where confirmations are requested incrementally. This approach takes advantage of the fact that some matches provide more benefit to the dataspace when confirmed than others: they are involved in more queries with greater importance or are associated with more data. Similarly, some matches may never be of interest, and therefore spending any human effort on them is unnecessary.

Since only a fraction of the candidate matches can be confirmed, the challenge is to determine which matches provide the most benefit when confirmed. Hence, the problem we consider in this paper is ordering candidate matches for confirmation to provide the most benefit to the dataspace.

Clearly, the means by which the system asks for confirmation is important. There needs to be some way to formulate a natural language question given a candidate match. Also, the system will likely have to ask multiple users the same question in order to form a consensus in the spirit of the ESP Game [28]. Furthermore, there may be subjective cases where two elements may be the same to some users, but not the same to others (e.g., red and cardinal are the same color to most people, but are different to artist or graphic designers). These issues are outside the scope of this paper.

Perfect and Known Dataspace States

To model the benefit of confirming individual candidate matches, we define the *perfect* dataspace D^P corresponding to D. In D^P, all the correct matches have been reconciled and two different strings necessarily represent distinct objects, attributes, or values in the real world. Once the equivalence between strings in D is known, D^P can be produced by replacing all the strings belonging to the same equivalence class by one representative element of that class. Of course, keep in mind that we do not actually know D^P.

The *known* dataspace for D, on the other hand, consists of the triples in D and the set of equivalences between elements determined by confirmed matches. Whenever the system receives a confirmation of a candidate match, it applies the match to the dataspace, updating the known dataspace state. That is, if a match (e_1, e_2, c) is confirmed, then we replace all occurrences of e_2 with e_1 (or vice versa). In practice, considering that this replacement operation may be expensive to apply very often, or that the triples are only a logical view of the data, we may want to model the confirmed matches as a separate concordance table.

At any given point, query processing is performed on the current known dataspace state. A confirmed match (e_1, e_2, c) causes the system to treat the elements e_1 and e_2 as equivalent for query processing purposes. Initially, we assume that *only* confirmed matches are used in query processing; we relax this requirement in Section 5 to accommodate other query answering models.

Queries and Workloads

Our discussion considers atomic queries, keyword queries and conjunctive queries.

An *atomic query* is of the form $(object = d)$, $(attribute = d)$, or $(value = d)$ where d is some constant. The answer to an atomic query Q over a dataspace D, denoted by $Q(D)$, is the set of triples that satisfy the equality.

A *keyword query* is of the form k, where k is a string. A keyword query is shorthand for the following: $((object = k) \vee (attribute = k) \vee (value = k))$; i.e., a query that requests all the triples that contain k *anywhere* in the triple.

Finally, a *conjunctive query*, a conjunction of atomic and keyword queries, is of the form $(a_1 \wedge \ldots \wedge a_n)$ where a_i is either an atomic query or a keyword query. The answer returned by a conjunctive query is the intersection of the triples returned by each of its conjuncts.

Recall that when querying the known dataspace D, the query processor utilizes all the confirmed candidate matches, treating the elements in each match equivalently. On the other hand, the query Q over D^P, the perfect dataspace corresponding to D, takes into consideration all matches that are correct in reality. We denote the result set of Q over D^P by $Q(D^P)$.

When determining which candidate match to confirm, the DSSP takes into consideration how such a confirmation would affect the quality of query answering on a *query workload*. A query workload is a set of pairs of the form (Q, w), where Q is a query and w is a weight attributed to the query denoting its relative importance. Typically, the weight assigned to a query is proportional to its frequency in the workload, but it can also be proportional to other measures of importance, such as the monetary value associated with answering it, or in relation to a particular set of queries for which the system is to be optimized.

Dataspace Statistics

We assume the DSSP contains basic statistics on occurrences of elements in triples in the dataspace. In particular, we assume the DSSP maintains statistics on the cardinality of the result set of any atomic query over the dataspace D. For example, for the atomic query $Q : (object = d)$ we assume the DSSP stores the number of triples in D that have d in their first position, denoted $|D^1_d|$. For conjunctive queries, the DSSP may either maintain multi-column statistics to determine the result sizes of conjunctive queries or use standard techniques in the literature [12, 3] to estimate such cardinalities.

3. ORDERING MATCH CONFIRMATIONS

This section introduces a decision-theoretic approach to ordering candidate matches for user confirmation in a dataspace. The key concept from decision theory we use is the *value of perfect information* (*VPI*) [21]. The value of perfect information is a means of quantifying the potential benefit of determining the true value for some unknown. In what follows, we explain how the concept of VPI can be applied to the context of obtaining information about the correctness of a given candidate match, denoted by m_j.

Suppose we are given a dataspace D and a set of candidate matches $M = \{m_1, \ldots, m_l\}$. Let us assume that there is some means of measuring the *utility* of the dataspace w.r.t. the candidate matches, denoted by $U(D, M)$, which we explain shortly. Given a candidate match m_j, if the system asks the user to confirm m_j, there are two possible outcomes, each with their respective dataspace: either m_j is confirmed as correct or it is disconfirmed as false. We denote the two possible resulting dataspaces by $D^+_{m_j}$ and $D^-_{m_j}$.

Furthermore, let us assume that the probability of m_j being correct is p_j, and therefore the expected utility of confirming m_j can be expressed as the weighted sum of the two possible outcomes: $U(D^+_{m_j}, M \setminus \{m_j\}) \cdot p_j + U(D^-_{m_j}, M \setminus \{m_j\}) \cdot (1 - p_j)$. Note that in these terms we do not include m_j in the set of candidate matches because it has either been confirmed or disconfirmed.

Hence, the benefit of confirming m_j can be expressed as the following difference:

$$\begin{aligned} Benefit(m_j) =& U(D^+_{m_j}, M \setminus \{m_j\}) \cdot p_j + \\ & U(D^-_{m_j}, M \setminus \{m_j\}) \cdot (1 - p_j) - \\ & U(D, M). \end{aligned} \quad (1)$$

Broadly speaking, the utility of a dataspace D is measured by the quality of the results obtained for the queries in the workload W on D compared to what the DSSP would have obtained if it knew the perfect dataspace D^P. To define $U(D, M)$, we first need to define the result quality of the query Q over a dataspace D, which we denote by $r(Q, D, M)$.

Recall that Q is evaluated over the dataspace D with the current *known* set of confirmed matches. Since our queries do not involve negation, all the results the DSSP returns will be correct w.r.t. D^P, but there may be missing results because some correct matches are not confirmed. Hence, we define

$$r(Q, D, M) = \frac{|Q(D)|}{|Q(D^P)|}$$

and the utility of the dataspace is defined as the weighted sum of the qualities for each the queries in the workload:

$$U(D, M) = \sum_{(Q_i, w_i) \in W} r(Q_i, D, M) \cdot w_i. \quad (2)$$

Our goal is to order matches to confirm by the benefit outlined in Equation 1: the matches that potentially produce the most benefit when confirmed are presented to the user first. However, in order to put this formula to use, we still face two challenges. First, we do not know the probability p_j of the candidate match m_j being correct. Second, since we do not know the perfect dataspace D^P, we cannot actually compute the utility of a dataspace as defined in Equation 2.

We address the first challenge by approximating the probability p_j by c_j, the confidence measure associated with candidate match m_j. In practice, the confidence numbers associated with the candidate matches are *not* probabilities, but for our purposes it is a reasonable approximation to interpret them as such. The second challenge is the topic of the next subsection.

3.1 Estimated Utility of a Dataspace

In what follows, we show how to estimate the utility of a dataspace using *expected utility*. Note that the set M represents the uncertainty about how D differs from the perfect dataspace D^P; any subset of the candidate matches in M may be correct. We denote the expected utility of D w.r.t. the matches M by $EU(D, M)$ and the expected quality for a query Q w.r.t. D and M as $Er(Q, D, M)$.

Once we have $EU(D, M)$, the value of perfect information w.r.t. a particular match m_j is expressed by the following equation, obtained by reformulating Equation 1 to use c_j instead of p_j and to refer to expected utility rather than utility:

$$\begin{aligned} VPI(m_j) =& EU(D^+_{m_j}, M \setminus \{m_j\}) \cdot c_j + \\ & EU(D^-_{m_j}, M \setminus \{m_j\}) \cdot (1 - c_j) - \\ & EU(D, M). \end{aligned} \quad (3)$$

The key to computing $EU(D, M)$ is to estimate the size of the result of a query Q over the perfect dataspace D^P. We illustrate our method for computing $Er(Q, D, M)$ for atomic queries of the form $Q : (object = d)$, where d is some constant. The reasoning for other atomic, keyword, and conjunctive queries is similar. Once we have $Er(Q, D, M)$, the formula for $EU(D, M)$ can be obtained by applying Equation 2.

Let us assume that the confidences of the matches in M being correct are independent of each other and that M is a *complete* set of candidates; i.e., if e_1 and e_2 are two elements in D and are the same in D^P, then there will be a candidate match (e_1, e_2, c) in M for some value of c. Given the dataspace D, there are multiple possible perfect dataspaces that are consistent with D and M. Each such dataspace is obtained by selecting a subset $M_1 \subseteq M$ as *correct* matches, and $M \setminus M_1$ as *incorrect* matches. We denote the perfect dataspace obtained from D and M_1 by D^{M_1}.

We compute $Er(Q, D, M)$ by the weighted result quality of Q on each of these candidate perfect dataspaces. Since we assume that the confidences of matches in M are independent of each other, we can compute $Er(Q, D, M)$ as follows:

$$Er(Q, D, M) = \sum_{M_1 \subseteq M} \frac{|Q(D)|}{|Q(D^{M_1})|} Pr(D^{M_1}) \quad (4)$$

where

$$Pr(D^{M_1}) = \prod_{m_i \in M_1} c_i \cdot \prod_{m_i \notin M_1} (1 - c_i).$$

Finally, to compute Equation 4 we need to show how to evaluate $|Q(D^{M_1})|$, the estimated size of Q on one of the possible candidate perfect dataspaces.

Recall that the size of Q over D, $|Q(D)|$, is the number of triples in D where d occurs in the first position of the triple. Hence, $|Q(D)| = |D^1_d|$, which can be found using the statistics available on the dataspace. In D^{M_1}, the constant d is deemed equal to a set of other constants in its equivalence class, $d_1, \ldots, d_m$. Hence, the result of Q over D^{M_1} also includes the triples with $d_1, \ldots, d_m$ in their first position and therefore $|Q(D^{M_1})| = |D^1_d| + |D^1_{d_1}| + \ldots + |D^1_{d_m}|$, which can also be computed using the dataspace statistics.

3.2 Approximating Expected Utility

In practice, we do not want to compute Equation 4 exactly as written because it requires iterating over all possible candidate perfect dataspaces, the number of which is exponential in $|M|$, the size of the set of candidate matches. Hence, in this subsection we show how we approximate $EU(D, M)$ with several simplifying assumptions. Our experimental evaluation shows that despite our approximations, our approach produces a good ordering of candidate matches.

Two approximations are already built into our development of Equation 4. First, the confidences of the matches in M are not necessarily independent of each other. Second, the set M may not include *all* possible correct matches, though we can always assume there is a candidate match for every pair of elements in D.

The main approximation we make when we compute the VPI w.r.t. a candidate match m_j of the form (e_1, e_2, c_j) is to assume that $M = \{m_j\}$. That is, we assume that M includes *only* the candidate match for which we are computing the VPI. The effect of this assumption is that we consider only two candidate perfect dataspaces, one in which m_j holds and the other in which m_j does not hold. We denote these two perfect dataspaces by $D_{m_j}^{e_1=e_2}$ and $D_{m_j}^{e_1\neq e_2}$, respectively.

Given this approximation, we can rewrite Equation 4 where $\{m_j\}$ is substituted for M:

$$\begin{aligned} Er'(Q, D, \{m_j\}) = & \frac{|Q(D)|}{|Q(D_{m_j}^{e_1=e_2})|} c_j + \\ & \frac{|Q(D)|}{|Q(D_{m_j}^{e_1\neq e_2})|}(1 - c_j) \end{aligned} \tag{5}$$

and therefore the expected utility of D w.r.t. $M = \{m_j\}$ can be written as

$$\begin{aligned} EU(D, \{m_j\}) = & \sum_{(Q_i,w_i)\in W} w_i \cdot \Big(\frac{|Q_i(D)|}{|Q_i(D_{m_j}^{e_1=e_2})|} c_j + \\ & \frac{|Q_i(D)|}{|Q_i(D_{m_j}^{e_1\neq e_2})|}(1 - c_j)\Big) \\ = & \sum_{(Q_i,w_i)\in W} w_i \cdot \frac{|Q_i(D)|}{|Q_i(D_{m_j}^{e_1=e_2})|} c_j + \\ & \sum_{(Q_i,w_i)\in W} w_i \cdot \frac{|Q_i(D)|}{|Q_i(D_{m_j}^{e_1\neq e_2})|}(1 - c_j). \end{aligned} \tag{6}$$

3.3 The Value Of Perfect Information

Now let us return to Equation 3. By substituting $\{m_j\}$ for M, we obtain the following:

$$\begin{aligned} VPI(m_j) = & EU(D_{m_j}^{+}, \{\}) \cdot c_j + \\ & EU(D_{m_j}^{-}, \{\}) \cdot (1 - c_j) - \\ & EU(D, \{m_j\}). \end{aligned} \tag{7}$$

Now note that once we employ the assumption that $M = \{m_j\}$, $Er'(Q, D_{m_j}^{+}, \{\})$ and $Er'(Q, D_{m_j}^{-}, \{\})$ are both 1 because they evaluate the utility of a dataspace that is the same as its corresponding perfect dataspace. Thus, using these values with Equation 6, $EU(D_{m_j}^{+}, \{\}) \cdot c_j$ and $EU(D_{m_j}^{-}, \{\}) \cdot (1 - c_j)$ become $\sum_{(Q_i,w_i)\in W} w_i \cdot c_j$ and $\sum_{(Q_i,w_i)\in W} w_i \cdot (1 - c_j)$, respectively. Furthermore, note that the last term at the end of Equation 6 also evaluates the utility of a dataspace that is the same as its corresponding perfect dataspace and thus simplifies to $\sum_{(Q_i,w_i)\in W} w_i \cdot (1 - c_j)$. Therefore, this term cancels with the second term of Equation 7. Hence we are left with the following:

$$\begin{aligned} VPI(m_j) = & \sum_{(Q_i,w_i)\in W} w_i \cdot c_j - \\ & \sum_{(Q_i,w_i)\in W} w_i \cdot c_j \frac{|Q_i(D)|}{|Q_i(D_{m_j}^{e_1=e_2})|} \\ = & \sum_{(Q_i,w_i)\in W} w_i \cdot c_j \left(1 - \frac{|Q_i(D)|}{|Q_i(D_{m_j}^{e_1=e_2})|}\right). \end{aligned}$$

Finally, we observe that only queries in W that refer to either e_1 or e_2 can contribute to the above sum; otherwise, the numerator and denominator are the same. Hence, if we denote by W_{m_j} the set of queries that refer to either e_1 or e_2, then we can restrict the above formula to yield the following, which is the one we use in our VPI-based user feedback ordering approach:

$$VPI(m_j) = \sum_{(Q_i,w_i)\in W_{m_j}} w_i \cdot c_j \left(1 - \frac{|Q_i(D)|}{|Q_i(D_{m_j}^{e_1=e_2})|}\right). \tag{8}$$

By calculating the VPI value for each candidate match using this equation, we can produce a list of matches ordered by the potential benefit of confirming the match.

EXAMPLE 3.1. *Consider an example unconfirmed candidate match m_j = (“red”, “cardinal”, 0.8) in the dataspace from Example 2.1. We compute the value of perfect information for m_j as follows.*

Assume that the dataspace workload W contains two queries relevant to m_j, Q_1 : (value = “red”) with a weight $w_1 = 0.9$ and Q_2 : (value = “cardinal”) with a weight $w_1 = 0.5$, and thus $W_{m_j} = \{(Q_1, 0.9), (Q_2, 0.5)\}$. In our example dataspace D, the cardinalities of the two relevant values in the third position is $|D^3{}_{\text{“red”}}| = 1$ and $|D^3{}_{\text{“cardinal”}}| = 2$. Therefore, the query cardinalities in the known dataspace D are $|Q_1(D)| = |D^3{}_{\text{“red”}}| = 1$ and $|Q_2(D)| = |D^3{}_{\text{“cardinal”}}| = 2$. In the perfect dataspace where the values “cardinal” and “red” refer to the same color, the cardinality for both queries is $|Q(D_{m_j}^{\text{“red”}=\text{“cardinal”}})| = |D^3{}_{\text{“red”}}| + |D^3{}_{\text{“cardinal”}}| = 3$.

Applying Equation 8 to compute the VPI for m_j, we have:

$$\begin{aligned} VPI(m_j) = & w_1 \cdot c_j \left(1 - \frac{|Q_1(D)|}{|Q_1(D_{m_j}^{\text{“red”}=\text{“cardinal”}})|}\right) + \\ & w_2 \cdot c_j \left(1 - \frac{|Q_2(D)|}{|Q_2(D_{m_j}^{\text{“red”}=\text{“cardinal”}})|}\right) \\ = & 0.9 \cdot 0.8 \left(1 - \frac{1}{3}\right) + 0.5 \cdot 0.8 \left(1 - \frac{2}{3}\right) = 0.61. \end{aligned}$$

This value represents the expected increase in utility of the dataspace after confirming candidate match m_j. □

4. EVALUATION

In this section we present a detailed experimental evaluation of the VPI-based approach for choosing the most beneficial candidate match to confirm presented in the previous section.

4.1 Experimental Setup

In order to validate our VPI-based strategy in a wide range of large-scale data integration environments, we built

Parameter	Description	Default
$NumElements$	The number of elements in the dataspace.	100000
$TriplesPerElementDistribution$	The number of triples in which each element appears.	Pareto with a shape parameter of 1.5
$MatchesPerElementDistribution$	The number of matches in which an element participates.	Pareto, constrained to [1,50]
$AccuracyDistribution$	The accuracy of the mechanisms.	Normal(0.8,0.1), constrained to [0,1]
$CorrectPercentage$	The percentage of the matches that are correct in reality.	0.5
$NumQueries$	The number of representative queries.	25000
$QueryWeightDistribution$	The query weight w.	Pareto, constrained to [0,1]

Table 2: Dataspace generator parameters. Unless otherwise specified, our experiments use the default values.

a dataspace generator capable of recreating a variety of such scenarios.

Dataspace Generator: The dataspace generator is seeded with a set of parameters that govern the size and characteristics of the dataspace and workload as listed in Table 2.

First, the generator produces some number of elements, governed by the parameter $NumElements$. For each element, the generator determines the number of triples in which it appears based on a distribution specified in $TriplesPerElementDistribution$. Unless otherwise specified, we use the long-tailed Pareto distribution [1] with a shape parameter of 1.5 to represent the scenario where a few elements appear in many triples, but most elements appear in a small number of triples.

Each element participates in some number of matches, governed by $MatchesPerElementDistribution$. Elements are assumed to participate in at least one candidate match; elements that do not participate in matches can be disregarded for user feedback purposes. For this distribution, we use a Pareto distribution modified to return values between 1 and 50. This distribution represents a scenario where most elements are matched with one or a small number of other elements while a small number of elements match with many other elements.

Each candidate match is assigned to be either correct or incorrect in reality, as dictated by the parameter $CorrectPercentage$. Unless otherwise specified, we create equal numbers of correct and incorrect matches ($CorrectPercentage = 0.5$). Given the correctness of a candidate match, its confidence c is determined by the accuracy of the mechanism from which it was produced: the better the accuracy for a mechanism, the closer the confidence values are to 1 for correct matches. Thus, a match's confidence is set based on a value a drawn from the distribution $AccuracyDistribution$ as follows: if the match is correct, then c is set to a, otherwise it is set to $1 - a$. For our experiments, we selected values for a from a Normal distribution, constrained between 0 and 1, with a mean of 0.8 and a standard deviation of 0.1 ($Normal(0.8, 0.1)$), similar to the accuracy of data integration mechanisms in the literature [22, 29, 7].

The generator then creates a set of queries as specified by $NumQueries$. Each query refers to a single element and is representative of the set of queries that refer that element. For simplicity, the generator only produces keyword queries. The generator assigns to the query a weight w using a distribution $QueryWeightDistribution$ to represent the query frequency for queries on this element. Since the distribution of query term frequencies on Web search engines typically follows a long-tailed distribution [23], for w in our experiments we use values selected from a Pareto distribution.

The output of the dataspace generator is a query workload W and the set of candidate matches M. Additionally, the generator outputs the number of triples associated with each element ($|D_e|$) and the correctness of each match.

Match Ordering Strategies: We compare a variety of candidate match ordering strategies. Each strategy implements a $score(m_j)$ function, which returns a numerical score for a given candidate match $m_j = (e_1, e_2, c_j)$. A higher score indicates that a candidate match should be confirmed sooner.

• VPI: $score(m_j) = VPI(m_j)$. Each candidate match is scored with the value of perfect information as defined in Equation 8.

• $QueryWeight$: $score(m_j) = \sum_{(Q_i,w_i)\in W_{m_j}} w_i$. Each candidate match is scored with the sum of query weights for that match's relevant queries.

• $NumTriples$: $score(m_j) = |D_{e_1}| + |D_{e_2}|$. This strategy scores each candidate match by number of triples in which the two elements in the match appear.

• $Combined$: $score(m_j) = (|D_{e_1}| + |D_{e_2}|) \sum_{(Q_i,w_i)\in W_{m_j}} w_i$. This strategy combines the above two strategies.

• $GreedyOracle$: This strategy measures the actual increase in utility using the correctness for each candidate match to simulate running the workload W with each candidate match confirmed. The match with the highest resulting utility is chosen for confirmation next. Note that this strategy is not a realistic ordering approach as it relies on knowing the correctness of all candidate matches and running the entire workload for each confirmation. It represents an upper-bound on any myopic strategy.

• $Random$: Finally, the naive strategy for ordering confirmations is to treat each candidate match as equally important. Thus, the next match to confirm in this strategy is chosen randomly. This strategy provides a baseline to which the above strategies can be compared

We score all candidate matches m_j in M using each of the above strategies and choose the match with the highest score to confirm next.

Confirming candidate matches: Given the next candidate match to confirm, we confirm it using the correct answer as dictated by the dataspace generator. After each confirmed match, the we update the set of equivalence classes and recompute the next best match for each strategy.

Measurement: After confirming some percentage of candidate matches using each of the orderings produced by each strategy, we run the workload W over the dataspace and measure the utility using the utility function defined in Equation 2.

4.2 Basic Tests

To study the basic efficacy of our VPI-based approach under realistic conditions, the first experiment we present

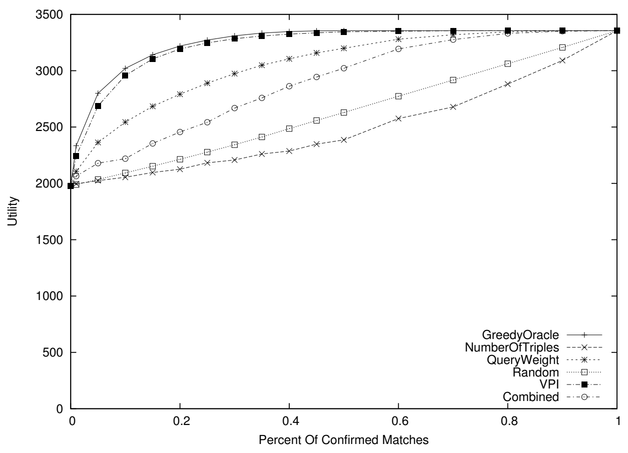

Figure 2: Basic tests comparing a VPI-based approach for ordering user feedback to other approaches in a large-scale data integration scenario.

investigates the performance of different ordering strategies using the default values for the parameters to the dataspace generator as stated in Table 2. We explore a range of other parameter values in subsequent experiments.

The resulting utility produced by confirming matches ordered by each strategy is shown in Figure 2. The results in this graph can be interpreted as follows. Since only a small fraction of candidate matches can be confirmed in a large-scale dataspace, the goal is to provide the highest utility with as few confirmations as possible. Thus, the slope of the curve at lower percentages of confirmations is the key component to the curve: the steeper the slope, the better the ordering.

First observe the curve for the *GreedyOracle* strategy. This approach selects the most beneficial candidate matches to confirm first and thus the curve is very steep at low percentages of confirmations. As it confirms more matches, the curve flattens out as these matches provide less benefit to the dataspace. Finally, it converges to the perfect dataspace, the dataspace where all element equivalences are known.

As can be seen, the VPI strategy performs very well; it tracks the *GreedyOracle* curve closely. Thus, despite the approximations employed to efficiently calculate the VPI, this strategy is close to the best a greedy strategy can do.

In contrast, the slopes of the curves for the other strategies are much shallower; it takes many more confirmations to produce a dataspace with a high utility. The $NumTriples$ strategy does particularly poorly. These results emphasize the importance of considering the query workload in when selecting candidate matches for confirmation: $NumTriples$ performs poorly because it fails to consider the workload. The utility produced by *Combined*, as expected, is between $NumTriples$ and $QueryWeight$. The utility of the dataspace increases roughly linearly as the percent of confirmations increase for the *Random* curve since it treats each candidate match as equally important.

While this graph provides a holistic view of how each strategy performs, we present in Table 3 two alternative views of this data to better illustrate the effect of match ordering strategy on dataspace utility. First, since the goal of user feedback is to move the known dataspace state towards the perfect dataspace, we report how many confirmations are required from each strategy until the utility of the dataspace reaches some percentage of the utility of the perfect dataspace. Second, since a DSSP can only request feedback for a small number of candidate matches, we report how close the utility of the resulting dataspace is to the perfect dataspace after a fraction of confirmed matches.

	A	B	C	D
Strategy	90%	95%	10%	25%
VPI	15	20	0.88	0.97
$QueryWeight$	35	50	0.76	0.86
$NumTriples$	100	100	0.62	0.65
$Combined$	60	50	0.69	0.76
$Random$	80	100	0.62	0.68
$GreedyOracle$	10	20	0.90	0.97

Table 3: Two measures of candidate match ordering effectiveness. Columns A and B show the percent of confirmed matches required to reach a dataspace whose utility is 90% or 95% of the utility of the perfect dataspace. Columns C and D show the resulting fraction of the perfect dataspace after confirming 10% or 25% percent of the matches.

These measures are reported in Table 3. The first two columns show the percentage of confirmed matches at which the utility of the dataspace is at 90% and 95% of the utility of the perfect dataspace. The last two columns give the utility relative to the perfect dataspace after confirming 10% and 25% of the matches from each strategy.

These numbers further emphasize the effectiveness of a VPI-based ordering approach. With only a small percentage of confirmations in using the VPI strategy, the utility of the dataspace closely approaches the utility of the perfect dataspace. For instance, if it is desired that a dataspace be within 95% of the perfect dataspace, then using the VPI-based match ordering approach needs to confirm only 20% of the candidate matches, equivalent to the oracle strategy and over twice as fast as the next best approach.

4.3 Partitioned Ordering

To study the need for a single unifying means of reasoning about user feedback in a dataspace, we compare our approach to a ordering algorithm that treats candidate matches produced by different mechanisms separately. We term this ordering strategy *Partitioned*, where $score(m_j)$ is calculated as follows. The algorithm partitions its matches into two categories, entity resolution and schema matching. To represent entity resolution matches, we create candidate matches that refer to elements that each have one associated triple. Schema matching, on the other hand, is represented by candidate matches whose elements have multiple associated triples. The intuition behind this strategy is that when comparing entity resolution to schema matching, the output of a schema matcher should be confirmed first as its matches usually affect more data than entity resolution. To provide a fair comparison, within each partition the matches are ordered by their VPI score.

For this experiment, we compare the *Partitioned* strategy to the VPI and *Random* strategy using the experimental setup as in the basic tests, but instead of generating a single class of matches, we set half of the matches to be entity resolution ($TriplesPerElementDistribution = Constant(1)$) and the other half to be schema matching ($TriplesPerElementDistribution = Normal(1000, 10)$).

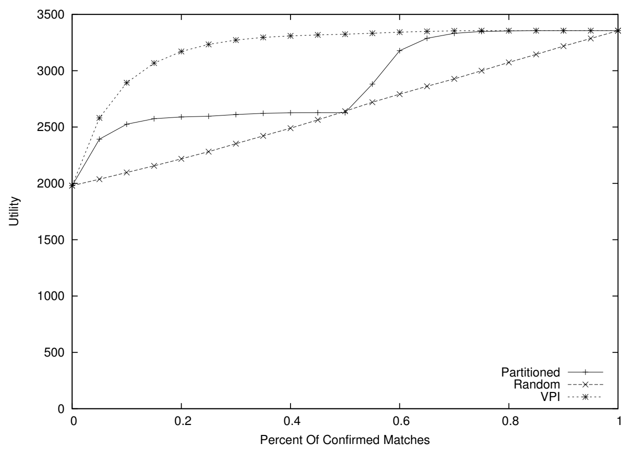

Figure 3: Partitioned ordering compared to a VPI-based approach.

The results are shown in Figure 3.

Here we can clearly see the two phases of match confirmations in the *Partitioned* curve: from 0 to 50% confirmations, the algorithm confirms matches from schema matching, after which it confirms entity resolution matches.

Of note in this figure is the tail end of the schema matching phase and the start of the entity resolution phase. The highly-ranked entity resolution confirmations provide more benefit than the lower-ranked schema matching confirmations, but since the two types of candidate matches are treated separately, these non-beneficial schema matchings are confirmed first and thus the overall utility of the dataspace suffers: the utility of the dataspace does not reach 90% of the perfect dataspace until confirming 60% of the matches, four times more confirmations than required by the VPI strategy.

This problem is not just a result of our particular setup or due to the details of the *Partitioned* strategy. Rather, any strategy that treats the output from different mechanisms separately will suffer from the same issues we see here. The problem is a result of multiple independent mechanisms using different, incomparable means of scoring candidate matches and thus there is no way to balance between the output of multiple mechanisms.

The key to solving this problem is that candidate matches from different mechanisms need to be ordered based on the overall benefit to the dataspace and not based on their ranking within each mechanism. The VPI strategy scores matches from different mechanisms in a uniform manner based on the expected increase in utility of the dataspace on confirmation; thus it is able to interleave match confirmations from multiple mechanisms in a principled manner to produce a dataspace with a higher utility using less user feedback.

4.4 Environmental Effects on VPI Ordering

Having demonstrated the basic advantage of our VPI-based ordering strategy in a realistic scenario, we now vary the parameters to the dataspace generator to produce different scenarios and measure the effect on the VPI strategy. We show that this approach is effective across a range of different environments. Additionally, these experiments yield insights as to which environmental factors matter when designing an ordering strategy for user feedback.

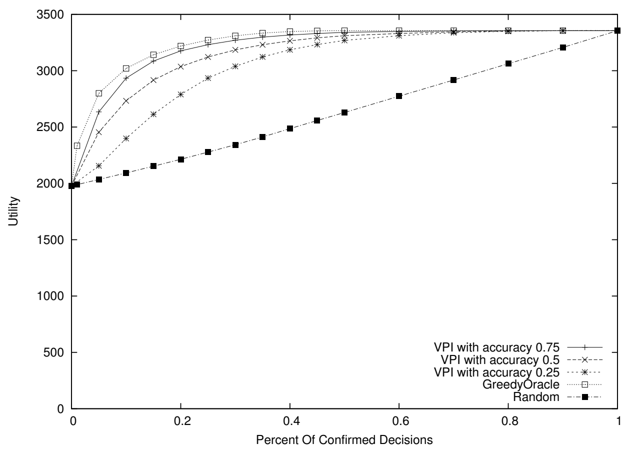

Figure 4: The effect of different accuracy of accuracy for mechanisms used to produce candidate matches.

Robustness to Inaccurate Mechanisms and Statistics

In the above tests, we assumed that the mechanisms producing candidate matches were reasonably accurate (i.e., 80% accuracy): if a candidate match is correct, the mechanism assigns it a high confidence. Here, we test how the VPI strategy is affected by inaccurate mechanisms.

For these tests, we use the same parameters as in the basic tests and then vary the accuracy of the mechanisms using different constant values. For each value of accuracy, we generate a set of candidate matches, order them using the VPI strategy, and then measure the results.

The results are shown in Figure 4. Each curve represents the VPI strategy at a given accuracy level. We also plot the curves for the *GreedyOracle* and *Random* strategies for comparison. Note that these strategies do not consider the accuracy of the mechanisms. As expected, the case where the accuracy is reasonably high (0.75), the VPI strategy produces a dataspace with high utility.

A more interesting case is when the accuracy level is set to 0.5, producing a dataspace where all candidate matches are given a confidence of 0.5. This curve is representative of a data integration environment without intelligent mechanisms: all pairs of elements can be considered candidate matches with arbitrary confidences. Even in this environment, the VPI strategy approach is still capable of producing a dataspace with a utility within 90% of the utility of a perfect dataspace with 30% of the matches confirmed, better than any other non-oracle strategy. Thus, the VPI-based approach is beneficial in helping guide user feedback when the DSSP does not employ any data integration mechanisms to assist in integration.

Finally, note that even when the mechanisms provide malicious confidence numbers (accuracy = 0.25), the VPI strategy still performs better than the *Random* approach. The VPI strategy orders candidate matches based on multiple measures including associated query weights and the number of triples in which each element appear and thus it is robust to a single inaccurate measure.

We also tested the effect of inaccurate element cardinality statistics on the performance of the VPI strategy. Similar to inaccurate mechanisms, the VPI strategy is robust to

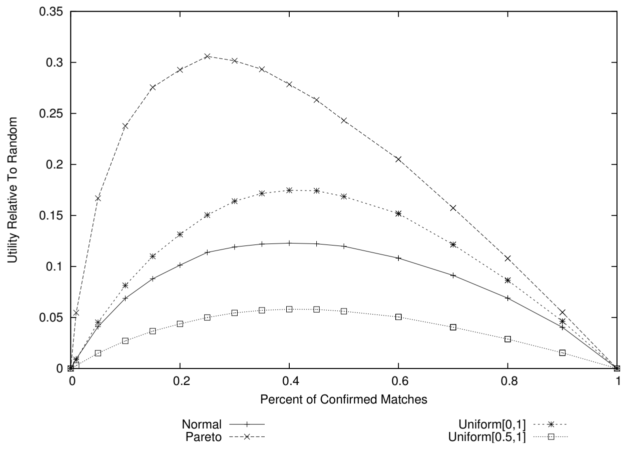

Figure 5: VPI utility relative to the baseline $Random$ for different values of $QueryWeightDistribution$.

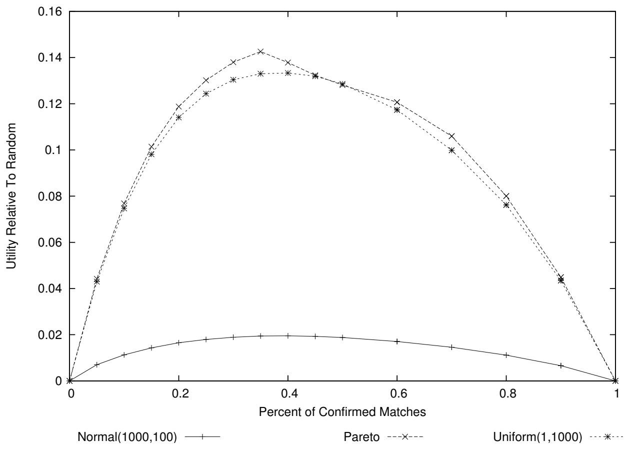

Figure 6: VPI utility relative to the baseline $Random$ for different values of $TriplesPerElementDistribution$.

poor statistics due to its multiple-measure scoring. We omit the details of this experiment due to their similarity to the accuracy tests.

Different Query Weight Distributions

Next, we explore the effect of different workloads on the VPI strategy. We generated workloads containing query weights using different types of distributions: *Pareto* (modified to produce values between 0 and 1), $Normal(0.5, 0.25)$, $Uniform(0, 1)$, and $Uniform(0.5, 1)$. Similar to the accuracy test above, we generate a curve for each query weight distribution using the VPI ordering strategy. Since changing the query weights used to generate the workload affects the overall utility of the dataspace, in order to present all curves on the same graph we report the utility relative the baseline strategy that treats all candidate matches equally (*Random*). The results are shown in Figure 5.

From this graph we can see two factors that affect the relative benefit of the VPI approach. First is the skew of the weights. As can be seen, the distribution that causes the most increase in relative utility is the highly-skewed Pareto distribution. This is expected as there are a small number of candidate matches that are involved in the queries that have a very high weight. The VPI strategy successfully selects these matches to confirm first causing the resulting utility of the dataspace to increase rapidly.

The second factor effecting the relative utility is the range of the weights. The smaller the range of query weights, the less distinguishing there is between the candidate matches and the less it matters to choose wisely. This can be seen in the difference between the $Uniform(0, 1)$ and $Uniform(0.5, 1)$ curve. The $Normal$ curve falls between these two as it has a larger range than $Uniform(0.5, 1)$, but it produces many queries with similar weights.

We note that this experiment illustrates the potential benefit from employing an intelligent ordering strategy in environments such as web search where some queries are valued much more than others. On the other hand, in environments with homogeneous queries, the selection method is less important.

Different Triples Per Element Distributions

Finally, we test the effect on our approach of the number of triples associated with each element: we use three different distributions for $TriplesPerElementDistribution$: $Normal(1000, 100)$, *Pareto*, and $Uniform(1, 1000)$. Similar to the query weight experiment, we plot a curve for each distribution using the VPI ordering strategy and compare the resulting utility to the *Random* strategy. The results are shown in Figure 6.

Similar to query weight, there are two factors that affect the relative benefit of the VPI strategy: the skew of the values and range. In a *Pareto* distribution, the skew of values is high and thus there are some elements that appear in a large number of candidate matches. In this environment, the VPI strategy confirms matches containing these elements early to prevent large numbers of missing results in queries over these elements. Similarly, the range of the values in the $Uniform$ distribution is wide, causing a large differentiation in the benefit of confirming some candidate matches over others. On the other hand, the VPI strategy in the $Normal$ environment shows only a small benefit over *Random* as many of the elements are similar in this environment; there is little to distinguish the benefit of confirming one candidate match over another.

4.5 Experimental Summary

Here we briefly summarize the experimental results presented in this section.

• Under realistic conditions, selecting candidate matches for user confirmation based on the value of perfect information yields a dataspace with a utility substantially higher than a wide range of other strategies. Moreover, the utility of the dataspace produced by this approach is within 5% of that produced by an oracle strategy.

• A single framework for handling user feedback for the output of multiple mechanisms is necessary for good ordering; approaches that consider the output of each mechanism separately fails to interleave user feedback from different mechanisms and thus the overall utility of the dataspace suffers.

• Our approach is robust to the accuracy of the mechanisms themselves: in an environment without intelligent data integration mechanisms, a VPI-based strategy produces a dataspace with a higher utility than any other strategy.

• In environments where the query or data skew or range is large, our approach is especially beneficial as it identifies the very important candidate matches and confirms them first.

5. QUERY ANSWERING USING THRESHOLDING

Until this point, our query answering model considered only confirmed matches. Since the goal of a dataspace system is to to provide query access to its underlying data sources even when they have not been fully integrated, it is likely that the system will want to also provide results that are based on matches that are not confirmed, but whose confidence is above some threshold. In this approach, the elements e_1 and e_2 in match $m = (e_1, e_2, c)$ are considered equivalent if the confidence c is greater than a threshold T.

Here, we analyze the impact of such a query answering model on our match scoring mechanism and show that our decision-theoretic framework can be applied with only minor changes to the utility function. We follow a similar process as in Section 3 to derive an equation for the value of perfect information for confirming match m_j when the query answering module uses thresholding.

We first need to redefine result quality when thresholding is used for query answering. Here, the query answering module may use an incorrect match if its confidence is above the threshold; thus, some answers in $Q(D)$ may not be correct w.r.t. $Q(D^P)$. To account for these incorrect results as well as the missed results due to correct but unused matches as before, we alter the equation for result quality to consider both precision and recall using F-measure [27][1], defined as

$$\frac{2 \cdot precision \cdot recall}{precision + recall}.$$

Precision and recall are defined in our context as follows:

$$precision(Q, D, M) = \frac{|Q(D) \cap Q(D^P)|}{|Q(D)|}$$

$$recall(Q, D, M) = \frac{|Q(D) \cap Q(D^P)|}{|Q(D^P)|}.$$

We redefine the result quality of query Q using F-measure as follows:

$$\begin{aligned} r(Q, D, M) =& \frac{2\frac{|Q(D)\cap Q(D^P)|}{|Q(D)|} \cdot \frac{|Q(D)\cap Q(D^P)|}{|Q(D^P)|}}{\frac{|Q(D)\cap Q(D^P)|}{|Q(D)|} + \frac{|Q(D)\cap Q(D^P)|}{|Q(D^P)|}} \\ =& \frac{2(|Q(D) \cap Q(D^P)|)}{|Q(D)| + |Q(D^P)|}. \end{aligned}$$

Substituting this formula into Equation 4, we can express the expected result quality, $Er(Q, D, M)$, when using thresholding:

$$Er(Q, D, M) = \sum_{M_1 \subseteq M} \frac{2(|Q(D) \cap Q(D^{M_1})|)}{|Q(D)| + |Q(D^{M_1})|} Pr(D^{M_1}) \quad (9)$$

From Section 3.1, we already know how to compute $|Q(D)|$ and $|Q(D^{M_1})|$ in this equation. To compute $|Q(D) \cap Q(D^{M_1})|$, first recall that in D^{M_1}, the constant d is deemed equal by the matches in M_1 to a set of other constants in its equivalence class, $\{d_1, \ldots, d_m\}$, which we denote here as $E_d^{M_1}$. Similarly, the matches in M that are above the threshold T determine a set of constants in D that are assumed to be equal to d when computing $Q(D)$, denoted as E_d^M. The set $Q(D) \cap Q(D^{M_1})$ includes the triples that have an element from the intersection of these two equivalence classes in the first position. Therefore, $|Q(D) \cap Q(D^{M_1})| = |D_d^1| + \sum_{d_i \in (E_d^M \cap E_d^{M_1})} |D_{d_i}^1|$.

Since computing Equation 9 is prohibitively expensive, we approximate $Er(Q, D, M)$ by employing the same assumption made in Section 3.2 where $M = \{m_j\}$. Thus, we rewrite Equation 9 with $\{m_j\}$ substituted for M:

$$\begin{aligned} Er'(Q, D, \{m_j\}) =& \frac{2(|Q(D) \cap Q(D_{m_j}^{e_1=e_2})|)}{|Q(D)| + |Q(D_{m_j}^{e_1=e_2})|} c_j + \\ & \frac{2(|Q(D) \cap Q(D_{m_j}^{e_1 \neq e_2})|)}{|Q(D)| + |Q(D_{m_j}^{e_1 \neq e_2})|} (1 - c_j). \quad (10) \end{aligned}$$

Finally, following the same logic used to derive Equation 8, we have:

$$VPI(m_j) = \sum_{(Q_i, w_i) \in W_{m_j}} c_j \cdot w_i \left(1 - \frac{2(|Q_i(D) \cap Q_i(D_{m_j}^{e_1=e_2})|)}{|Q_i(D)| + |Q_i(D_{m_j}^{e_1=e_2})|}\right). \quad (11)$$

We implemented this VPI scoring method and experimentally evaluated the ordering it produced when the query answering module uses thresholding. These experiments yielded results very similar to those presented in the previous section, verifying the fact that our framework can be applied to query answering using thresholding. We omit the details due to their similarity with the previous sections results and space considerations.

6. ROOMBA

In this section, we outline the architecture of Roomba[2], a component of a DSSP that incorporates the decision-theoretic framework presented in the previous sections to provide efficient, pay-as-you-go match ordering using VPI. The overall architecture of Roomba is shown in Figure 7.

To facilitate a pay-as-you-go mode of interaction, the DSSP contains a user interaction module that determines the appropriate time to interrupt the user with confirmation request, such as in [11]. At such a time, this module calls the method *getNextMatch*() exposed by Roomba that returns the next best match to confirm. The naive approach to supporting such a method call is to order all matches once and then return the next best match from the list on each call to *getNextMatch*().

In a pay-as-you-go DSSP, however, *getNextMatch*() is called over time as the system runs; thus, a particular ordering of matches derived at one point of time using one

[1]Here, we use F-measure with precision and recall as equally important, sometimes referred to as F1-measure.

[2]The name "Roomba" alludes to the vacuuming robot of the same name [13]. Just as the robot discovers what needs to be cleaned your room, the Roomba system aids a DSSP in determining what needs to be cleaned in the dataspace.

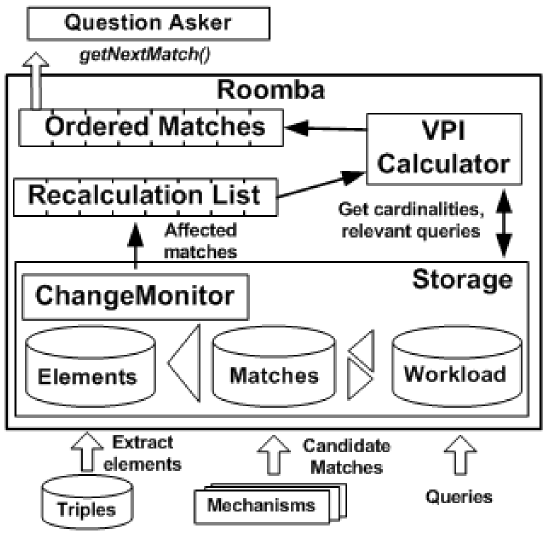

Figure 7: Roomba architecture

state of the dataspace may become invalid as the characteristics of the dataspace change. A match's VPI score depends on the queries for which it is relevant, the cardinalities of the elements involved in the match, and the confidence of the match. Furthermore, over time confirmed matches may be fed back to the data integration mechanisms which, as a result, may alter some match predictions.

The key to efficiently supporting $getNextMatch()$ under changing conditions is to limit the number of VPI scores that need to be recomputed when some aspect of the dataspace changes. Here we briefly outline the techniques employed by Roomba to efficiently compute the next best match as the characteristics of the dataspace change.

Roomba operates in three phases: initialization, update monitoring, and $getNextMatch()$.

Initialization: The initialization phase creates all the data structures used by Roomba and produces an initial ordering of matches. At this point, Roomba also calculates the element cardinality statistics over the dataspace. In order to facilitate efficient VPI recomputation, Roomba builds indexes that map from each aspect of the data that factors into the VPI calculation to matches that would be affected by a change in that data. Finally Roomba calculates the initial VPI score for each match as defined in Equation 3 and stores them in an ordered list. Note that these VPI computations can be done in parallel.

Update Monitoring: While the system runs, the dataspace's conditions will continuously change, potentially causing an invalidation of the ordering derived during initialization. When such a change occurs, a $ChangeMonitor$ notes the type of change (i.e., element cardinality, query workload, or match confidence) and utilizes the indexes built during the initialization phase to find the matches that are affected by the particular change. Only these matches are flagged for recomputation in a recalculation list to be processed on the next call to $getNextMatch()$.

$getNextMatch()$**:** On a call to $getNextMatch()$, Roomba sends the recalculation list to the VPI calculator to recompute and reorder any matches whose VPI score may have changed. Note that here, too, the VPI calculations can be done in parallel. Roomba then returns the top match off the list for user confirmation.

7. RELATED WORK

While we have based our decision-theoretic framework on formalisms used in the AI community [21], decision theory and the value of perfect information are well-known concepts in many fields such as economics [18, 16] and health care [2]. Within the data management community, there has been work on applying expected utility to query optimization [4].

Previous work on soliciting user feedback in data integration systems has focused on the output of a single mechanism. The work in [7] and [29] addresses incorporating user feedback into schema matching tasks. Similarly, [22] introduces an active-learning based approach to entity resolution that requests user feedback to help train classifiers. These approaches are closely tied to a single type of data integration task and select candidate matches for feedback based closely on the type of classifier it is using. Furthermore, their overall goal is to reduce the uncertainty in the produced matches without regard to how important those matches are to queries in the dataspace. Rather than reasoning about user feedback for each mechanism separately, a primary benefit of our framework is that it treats multiple mechanisms uniformly and judiciously balances between them with the goal of providing better query results for the dataspace.

The MOBS [17] approach to building data integration systems outlines a framework for learning the correctness of matches by soliciting feedback from many users and combining the responses to converge to the correct answer. While MOBS does unify multiple mechanisms in one framework, it do not address how to select which question to ask the user. Our approach naturally fits within this framework: when the MOBS system decides to solicit user feedback, Roomba can provide the best match to confirm, the results of which can be fed back into the MOBS system for learning.

8. CONCLUSIONS AND FUTURE WORK

This paper proposed a decision-theoretic approach to ordering user feedback in a dataspace. As part of this framework, we developed a utility function that captures the usefulness of a given dataspace state in terms of query result quality. Importantly, our framework enables reasoning about the benefit of confirming matches from multiple data integration mechanisms in a uniform fashion. While we mostly considered entity resolution and schema matching, other data integration mechanisms to which we can apply these techniques are information extraction, relationship and entity extraction, and schema clustering. We then presented a means of select matches for confirmation based on their value of perfect information: the expected increase in dataspace utility upon requesting user feedback for the match. We described a set of experiments that validated the benefits of our framework.

This decision-theoretic framework provides a rich basis for future work. Here, we outline some of the avenues we are exploring.

The first direction is to extend our techniques beyond the *myopic* value of perfect information. Currently, our approach computes the expected benefit of confirming the next match and greedily selects the best one for which to request user feedback. There may, however, be a series of multiple matches that, when confirmed, would produce a dataspace with higher utility than with the myopic approach. Conceptually, we can apply non-myopic decision making to our setting to look ahead to multiple possible confirmations to find such sets of matches. The challenge is to balance the distance of the look-ahead with its cost and with the fact that as the dataspace changes, some of the characteristics on which the VPI calculations were based may change.

A second direction is to explore other kinds of user feedback. In this paper, we explored how to efficiently involve users in resolving uncertainty through *explicit* user feedback. The DSSP can also leverage the wealth of research on *implicit* feedback (e.g., [14, 5, 19]) to improve the certainty of candidate matches. For instance, the click-through rate of query results supply an indicator of the correctness of the matches employed during query answering: a click on a particular result may indicate that the matches used to compute that result are correct, causing the DSSP to increase the confidence of those matches. A DSSP can also use information from subsequent queries, or query chains [19], to reason about the correctness of matches not employed during query processing. If, for instance, a user searches for "red" and then subsequently searches for "cardinal", then the DSSP can increase the confidence of the candidate match ("red", "cardinal", c).

Another promising area of future work is exploring the interaction of decision theory and query answering using unconfirmed matches. Rather than use a static threshold for query answering, a DSSP can utilize our decision-theoretic framework to determine what uncertain matches to use for a query based on the principle of *maximum expected utility* (*MEU*) [21]. Intuitively, employing MEU for the decision to utilize or disregard a match for a particular query involves choosing the action with the highest the expected utility.

Finally, we intend on building and deploying Roomba within a large-scale, real-world dataspace to apply the principles presented in this paper to large-scale data integration environment.

9. REFERENCES

[1] G. Casella and R. Berger. *Statistical Inference.* Duxbury, 2002.

[2] G. B. E. Chapman and F. A. E. Sonnenberg. *Decision Making in Health Care: Theory, Psychology, and Applications.* Cambridge University Press; New Ed edition (September 1, 2003), 2003.

[3] S. Chaudhuri, G. Das, and U. Srivastava. Effective use of block-level sampling in statistics estimation. In *SIGMOD '04*, 2004.

[4] F. Chu, J. Y. Halpern, and P. Seshadri. Least expected cost query optimization: an exercise in utility. In *PODS '99*, 1999.

[5] M. Claypool, P. Le, M. Wased, and D. Brown. Implicit interest indicators. In *Intelligent User Interfaces*, pages 33–40, 2001.

[6] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks, 2003.

[7] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *SIGMOD '01*, 2001.

[8] A. Doan and A. Y. Halevy. Semantic-integration research in the database community. *AI Mag.*, 26(1):83–94, 2005.

[9] Flickr. http://www.flickr.com.

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

[11] E. Horvitz, C. Kadie, T. Paek, and D. Hovel. Models of attention in computing and communication: from principles to applications. *Commun. ACM*, 46(3):52–59, 2003.

[12] Y. E. Ioannidis. The history of histograms (abridged). In *VLDB*, pages 19–30, 2003.

[13] iRobot Roomba. http://www.irobot.com/sp.cfm?pageid=122.

[14] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR '05*, 2005.

[15] J. Madhavan, A. Y. Halevy, S. Cohen, X. L. Dong, S. R. Jeffery, D. Ko, and C. Yu. Structured data meets the web: A few observations. *IEEE Data Eng. Bull.*, 29(4):19–26, 2006.

[16] A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic Theory.* Oxford, 1995.

[17] R. McCann, A. Doan, V. Varadaran, A. Kramnik, and C. Zhai. Building data integration systems: A mass collaboration approach. In *WebDB*, 2003.

[18] O. Morgenstern and J. Von Neumann. *Theory of Games and Economic Behavior.* Princeton University Press, 1944.

[19] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback, 2005.

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

[21] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[22] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD '02*, 2002.

[23] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz. Analysis of a very large altavista query log. Technical Report 1998-014, Digital SRC, 1998. http://gatekeeper.dec.com/pub/DEC/SRC/technical-notes/abstracts/src-tn-1998-014.html.

[24] J. Surowiecki. *The wisdom of crowds.* Doubleday, 2004.

[25] The Large Hadron Collider. http://lhc.web.cern.ch/lhc/.

[26] G. Tolle, J. Polastre, R. Szewczyk, D. E. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, and W. Hong. A Macroscope in the Redwoods. In *SenSys*, pages 51–63, 2005.

[27] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition.* Dept. of Computer Science, University of Glasgow, 1979.

[28] L. von Ahn and L. Dabbish. Labeling Images with a Computer Game. In *ACM CHI*, 2004.

[29] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD '04*, 2004.