Pay-as-you-go User Feedback for Dataspace Systems

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 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 on both real and synthetic datasets 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 utilizes this decision-theoretic framework to guide a dataspace in soliciting user feedback in a pay-as-you-go manner.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: System

General Terms

Algorithms, Experimentation

Keywords

Dataspace, Data Integration, Decision Theory, User Feedback

*This work was done in part while the author was an intern at Google Inc.

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 [32], sensor deployments [33], and an increasingly structured Web [19], there is a growing need to provide unified access to these heterogeneous data sources. Dataspaces [12] 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 [9]. 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. Typically, a dataspace system employs a set of mechanisms for semantic integration, such as schema matching [25] and entity resolution [7], 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* [31], in order to better understand sets of data [11, 22, 35].

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 such 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*) [26]. 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 entities in the domain. In this regard, our work is distinguished from previous research in soliciting user feedback for data integration tasks (e.g., [27, 37, 8]) that are tightly integrated with a mechanism (i.e., schema matching or entity resolution).

At the core of VPI is a *utility function* that quantifies the desirability of a given state of a dataspace; to make a decision, we need to compare the utility of the dataspace *before* the user confirms a match and the utility of the dataspace after. 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 dataspace system does not know the correctness of the candidate matches, we develop a set of approximations that allow the system to efficiently estimate the utility of a dataspace.

We describe a detailed experimental evaluation on both real and synthetic datasets showing that the ordering of user feedback produced by our VPI-based approach yields a dataspace with a significantly higher utility than a variety of other ordering strategies. We also illustrate experimentally the benefit of considering multiple schema integration mechanisms uniformly: we show that an ordering approach that treats each class of mechanisms separately for user-feedback 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. We conclude in Section 8.

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.

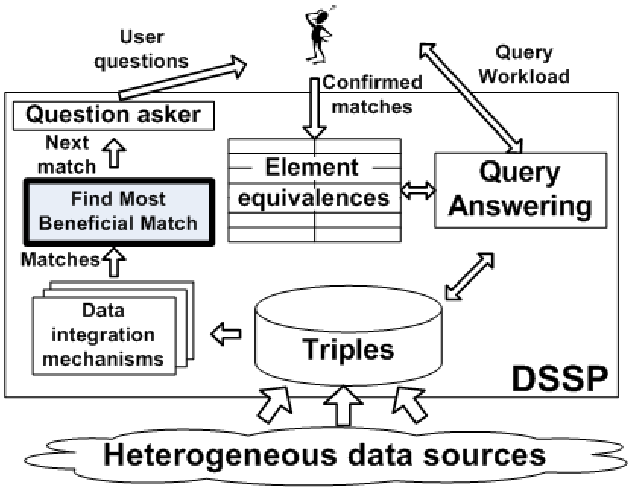

Figure 1: Architecture for incorporating user feedback in a dataspace system.

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.

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

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 disparate sources, they display a high degree of heterogeneity. For example, the triples in a 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 entities 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 [25], and there are techniques for entity resolution (also referred to as object de-duplication) that predict whether two object or value references are about the same real-world entity [7]. We assume that the mechanisms employ techniques for dealing with large amounts of data (e.g., [1]).

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 such a question.

Ideally, the system should ask users 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 [12], 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 [35]. 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 artists 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. In this paper, we assume that the real world can be partitioned as such using only strings; in some cases, however, it may be necessary to contextualize the strings with additional metadata to avoid linking unrelated concepts through a common string (e.g., the strings "Cal" and "City of Berkeley" may be incorrectly deemed equivalent if the matches ("Cal","Berkeley", c_1) and ("Berkeley", "City of Berkeley", c_2) are confirmed). 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 (potentially through the use of a concordance table as discussed above). 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 system 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 dataspace system contains basic statistics on occurrences of elements in triples in the dataspace. In particular, we assume the system 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 system stores the number of triples in D that have d in their first position, denoted $|D_d^1|$. For conjunctive queries, the system may either maintain multi-column statistics to determine the result sizes of conjunctive queries or use standard techniques in the literature to estimate such cardinalities (e.g., [15, 4]).

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*) [26]. 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 dataspace system 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 system 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. In Section 4, we show how to handle cases where such an approximation does not hold. 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:

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

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, $|Q(D)| = |D_d^1|$, 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 $|Q(D^{M_1})| = |D_d^1| + |D_{d_1}^1| + \ldots + |D_{d_m}^1|$, 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. There may, for instance, be one-to-one mappings between two data sources such that a disconfirmed match between two elements would increase the confidence of all other matches in which those elements participate; such considerations can be layered on top of the techniques presented here. 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

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

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 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:

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

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:

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

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 $w_2 = 0.5$, 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 on both real-world and synthetic datasets of the VPI-based approach presented in the previous section.

4.1 Google Base Experiments

The first set of experiments we present use real-world datasets derived from Google Base [13].

Experimental Setup

Google Base. Google Base is an online repository of semi-structured, user-provided data. The primary unit of data in Google Base is an *item*. Items consist of one or more attribute/value pairs (e.g., ⟨"make"= "Toyota", "model" = "Prius"⟩). Items are organized into *item types*, which are essentially domains or verticals, such as vehicles or recipes. For each item type, Google Base provides a set of recommended attributes (i.e., schema) and popular values for each attribute. Users can, however, make up their own attributes and values for their items. Since each user is free to use their own terminology when uploading items, there are many cases where different strings for attributes or values in two different items refer to the same concept in reality. Examples of such correspondences are: ("address" ↔ "location"), ("door_count" ↔ "doors"), and ("tan" ↔ "beige").

Characteristic	*Restricted*	*Full*
Total items	8000	16000
Total triples	60716	148050
Unique triples	52773	113489
Total attributes	333	708
Total values	6900	16924
Total elements	7233	17632
Candidate matches	839	4853
Percent correct matches	0.33	0.23
Avg matches per element	2.8	4.9

Table 2: Statistics for the Google Base datasets. *Total items* denotes the total number of Google Base items in each dataset. Each of these items may contain multiple triples and thus *total triples* is the total number of triples contained in those items. Many of these triples are identical: *unique triples* is the number of unique triples in each dataset. In these datasets, the elements of interest are attributes and values; this table shows the counts for each of these types of elements and the total number of elements. Additionally, we show statistics for the matches: the total number of matches produced by the mechanisms (*Candidate matches*), the fraction of correct matches (*Percent correct matches*), and the average number of matches in which each element participates (*Avg matches per element*).

Google Base datasets. To collect the data used for our experiments, we sampled Google Base to get 1000 elements from each of the 16 standard item types recommended by Google Base. These item types encompass most of the data in Google Base across a wide range of domains. The item types we sample from are as follows: business locations, course schedules, events and activities, housing, jobs, mobile, news and articles, personals, products, recipes, reference articles, reviews, services, travel packages, vehicles, wanted ads.

We partition this sample into two datasets (characteristics for these datasets are shown in Table 2):

• *Full*: This dataset contains all 16 item types, and thus represents the full range of semantic heterogeneity that exists in Google Base. Due to this heterogeneity, correct correspondences are challenging for the mechanisms to determine and thus many of the matches are incorrect; only 23% of the matches in this dataset are correct matches.

• *Restricted*: This dataset contains a relatively small amount of semantic heterogeneity: there are fewer cases where two strings refer to the same entity. Thus, the mechanisms produce a smaller number of matches and a higher percentage of correct matches (see Table 2). This dataset was created by selecting items from item types with less heterogeneity as follows: business locations, housing, news and articles, products, recipes, reference articles, reviews, wanted ads.

Ground truth. In order to evaluate our techniques against ground-truth, we manually annotated the dataset with correspondences between attributes and values in each of the datasets.

Mechanisms. We employ two different types of mechanisms, each applied to both attribute and value matching.

For the first mechanism, we use Google Base's internal attribute and value matching, termed *adjustments*, designed to convert attributes and values to canonical strings. For example, the attribute "Address" is converted to "location" and the value "F" is converted to "female". The Google Base data we collected includes for each adjustment both the original string and the adjusted string. Each original and adjusted string pair represents a candidate match. Note that matches produced by this mechanism are particularly challenging to use in VPI calculations as the matches *do not* have an associated confidence. We discuss how to address this challenge below.

The second mechanism we use is the SecondString library for approximate string matching [29]. This mechanism also operates on both attributes and values. From this library, we use the *SoftTFIDF* approach [7], a hybrid approach combining the Jaro-Winkler metric [36] (a measure based on how many characters the strings have in common and their ordering) with TF-IDF similarity. This mechanism was shown to have the best string matching accuracy in a variety of experiments [7]. For a match's confidence, this mechanism produces a "score", a number between 0 and 1, with 1 indicating the highest degree of similarity. While this score is loosely correlated with the probability the match is correct, it is inaccurate. To limit the number of candidate matches, we only consider matches with a score greater than 0.5 for confirmation.

Candidate matches. Using these two mechanisms, we produce a set of candidate matches for each dataset. We annotate these matches as correct or incorrect as determined by the manual annotation.[1]

Converting confidences to probabilities. Since our VPI-based approach uses the confidence numbers produced by mechanisms as probabilities to drive its VPI calculations, a major challenge presented by these candidate matches is to convert the match confidences into probabilities.

To address this challenge, we use a histogram-based learning approach. As the system confirms matches, it compiles how many times each mechanism produces correct or incorrect matches and the corresponding input confidence (or lack thereof) for each match. Using these observed probabilities, the system maintains a histogram for each mechanism that maps the confidence value for matches produced by that mechanism to a probability: each bucket corresponds to an input confidence range and the bucket contains the computed probability for that input confidence. Upon each confirmed match (or after a batch of confirmed matches), the system recomputes the probabilities for each mechanism and then applies the computed probabilities to the remaining candidate matches based on their confidences. Note that for mechanisms without any confidence value (e.g., the Google Base adjustments in this scenario), this approach assigns a single confidence value based on the percentage of correct matches produced (and confirmed as correct) by that mechanism (i.e., the histogram only has one bucket).

Queries. We use a query generator to generate a set of queries. Each generated 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 each query a weight w using a distribution to represent the frequency of queries on this element. Since the distribution of query-term frequencies on Web search engines typically follows a long-tailed distribution [30], for w in our experiments we use values selected from a Pareto distribution [2]. We evaluate other query workloads below.

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. The intuition behind this strategy is that important queries should have their relevant matches confirmed earlier.

• $NumTriples$: $score(m_j) = |D_{e_1}| + |D_{e_2}|$. This strategy scores each candidate match by number of triples in which the two elements in the match appear. The rationale behind this strategy is that matches containing elements appearing in many triples are more important since queries involving the elements in these matches will miss more data if the match is correct but not confirmed.

• *GreedyOracle*: For each unconfirmed candidate match, this strategy runs the entire query workload W and measures the actual increase in utility resulting from confirming that match. To calculate the actual utility, this strategy uses the manual annotation for all matches to determine if they are correct or not in reality. The match with the highest resulting utility is chosen as the next confirmation. Note that this strategy is not a realistic ordering approach as it relies on knowing the correctness of all matches as well as running the entire workload for all unconfirmed matches for each match confirmation. It is 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

Using each of the above strategies, we score all candidate matches 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 determined by the manual annotation. 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 query workload W over the dataspace and measure the utility using the utility function defined in Equation 2. We report the percent of improvement in utility over a dataspace with no confirmed matches.

[1]All identified correspondences are captured in the candidate matches.

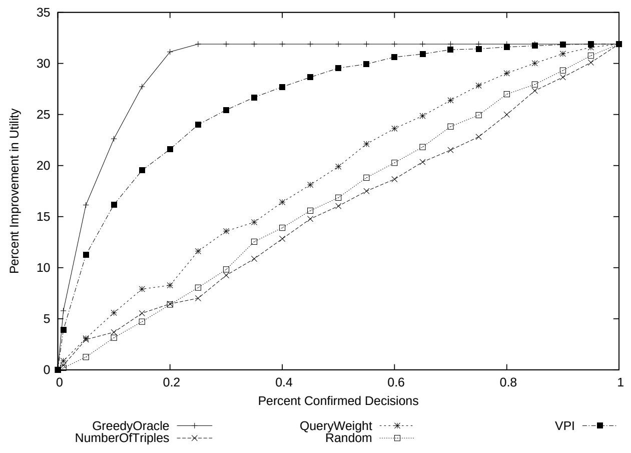

Figure 2: Basic test comparing a VPI-based approach for ordering user feedback to other approaches run over the *Full* dataset.

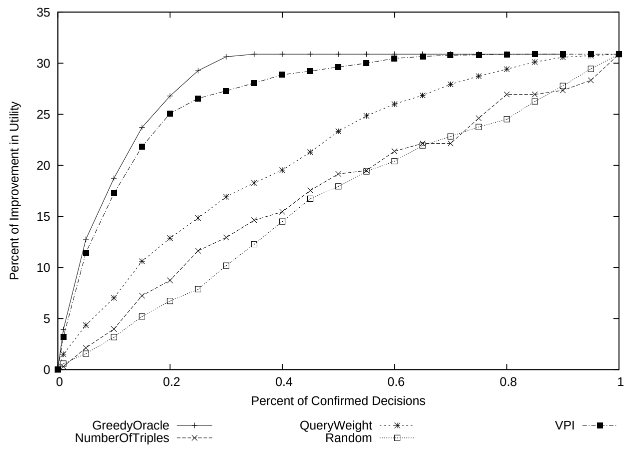

Figure 3: Basic test comparing a VPI-based approach for ordering user feedback to other approaches run over the *Restricted* dataset.

Basic Tests

To study the basic efficacy of our VPI-based approach, the first experiments we present investigate the performance of different ordering strategies on both of the Google Base datasets. We use the basic setup as described above with each dataset.

The resulting utility produced by confirming matches ordered by each strategy for the *Full* dataset 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 for the early confirmations. As it confirms more matches, the curve flattens out as these matches provide less benefit to the dataspace. Finally, it converges to the utility of the perfect dataspace, i.e., the dataspace where all element equivalences are known.

As can be seen, the VPI strategy performs comparatively well: it tracks the *GreedyOracle* strategy much more closely than any of the other strategies. The VPI strategy performs well despite the fact that this dataset is particularly challenging for any non-oracle strategy as there is a large degree of semantic heterogeneity. First, this dataset contains many incorrect matches, which provide no benefit to the dataspace's utility when confirmed. More challenging, however, is that many of the incorrect matches are given similar confidences. As an example of the challenges in this dataset, consider the following matches created by the SecondString mechanism in the vehicles item type: the candidate match (*"AM/FM Stereo Cassette/Cd"*, *"AM/FM Stereo Cassette & CD Player"*) is correct in reality with a confidence of 0.91 while the match (*"AM/FM Stereo Cassette/Cd"*, *"AM/FM Stereo Cassette"*) is incorrect in reality with a similar confidence of 0.92. The VPI strategy is unable to discern between these two matches and thus occasionally incorrectly orders them. Despite these challenges, however, the VPI strategy substantially outperforms all other non-oracle strategies.

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 when selecting candidate matches for confirmation: $NumTriples$ performs poorly because it fails to consider the workload. 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.

The results for the basic tests on the *Restricted* dataset are shown in Figure 3. Here, the VPI-strategy does particularly well; it tracks the *GreedyOracle* curve closely. Since this dataset contains less semantic heterogeneity than the *Full* dataset, the VPI strategy is able to easily discern the best matches to confirm.

While these graphs provide a holistic view of how each strategy performs, we present in Table 3 two alternative views of this data (for the *Restricted* dataset) to better illustrate the effect of match ordering strategy on dataspace utility. First, since a large-scale dataspace system can only request feedback for a very small number of candidate matches, we report the improvement after a small fraction of confirmed matches (here, 10%). Second, 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 fraction of the utility of the perfect dataspace (here, 0.95).

These numbers further emphasize the effectiveness of a VPI-based ordering approach. With only a small percentage of confirmations using the VPI strategy, the utility of the dataspace closely approaches the utility of the perfect dataspace. For instance, with only 20% of the confirmations, the VPI strategy is able to produce a dataspace that is 95% of the perfect dataspace, equivalent to the oracle strategy and over twice as fast as the next-best strategy.

Strategy	10% matches confirmed	0.95 of perfect dataspace
VPI	17.2	0.20
$QueryWeight$	7.0	0.55
$NumTriples$	3.9	0.75
$Random$	3.2	0.80
$GreedyOracle$	18.7	0.20

Table 3: Two measures of candidate match ordering effectiveness (shown for the *Restricted* dataset). The first column shows the resulting percent of improvement after confirming 10% percent of the matches. The second column shows the fraction of confirmed matches required to reach a dataspace whose utility is 0.95 of the utility of the perfect dataspace.

Partitioned Ordering

To study the need for a single unifying means of reasoning about user feedback in a dataspace, we compare the VPI approach to an ordering algorithm that treats candidate matches produced by different mechanisms separately. The general idea with this strategy is that the output of each mechanism is ordered separately in a *partition* for each mechanism's matches, and then these partitions are ordered. We term this ordering strategy $Partitioned$, where $score(m_j)$ is calculated as follows. The algorithm separates its matches into partitions corresponding to the output of the two mechanisms (GoogleBase adjustments and SecondString). These partitions are roughly ordered based on the relative performance of each mechanism. For this experiment, $Partitioned$ orders Google Base adjustments first, followed by SecondString matches. To provide a fair comparison, within each partition the matches are ordered by their VPI score. This strategy represents the case where individual mechanisms each perform their own ordering; there is no global ordering beyond deciding how to order the partitions.

For this experiment, we compare the $Partitioned$ strategy to the VPI strategy using the experimental setup as in the basic tests using the *Restricted* dataset. We also include the curve for the *Random* strategy for reference. The results are shown in Figure 4.

Here we can see the distinct phases of match confirmations in the $Partitioned$ curve: in the early confirmations (prior to point A), the algorithm confirms matches from Google Base adjustments, after which (after point B) it confirms SecondString matches.

Of note in this figure is the tail end of the Google Base confirmation phase (point A) and the start of the SecondString confirmation phase (point B). The highly-ranked SecondString confirmations provide more benefit than the lower-ranked Google Base confirmations, but since the two types of candidate matches are ordered separately, these non-beneficial Google Base matches are confirmed first and thus the overall utility of the dataspace suffers: at 10% confirmations, the percent improvement for $Partitioned$ is 9.4%, whereas with the VPI strategy it is 17.2%.

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.

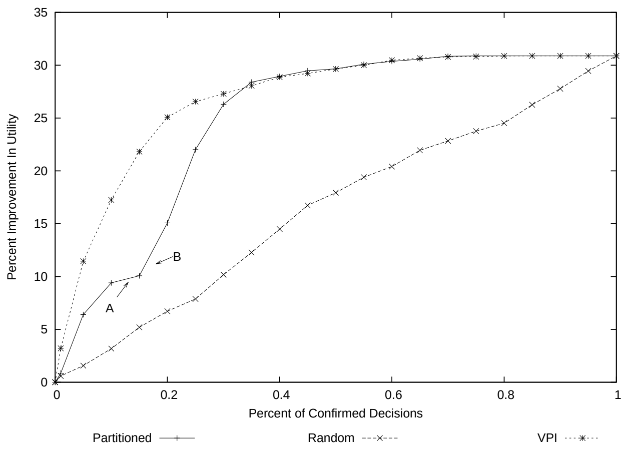

Figure 4: Experiment comparing a strategy that separately orders candidate matches from disparate mechanisms to the VPI strategy, run on the *Restricted* dataset.

The problem is a result of multiple independent mechanisms using different, incomparable means of ordering 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 globally ordered based on the overall benefit to the dataspace and not based on their ranking relative to matches produced 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.

Different Query Workloads

Here, we explore the effect of different query workloads on the VPI strategy. We generate workloads for the *Full* dataset containing query weights using different types of distributions: $Pareto$ (modified to produce values between 0 and 1) as was used above, a Normal distribution with a mean of 0.5 and a standard deviation of 0.25 ($Normal(0.5, 0.25)$), a Uniform distribution between 0 to 1 ($Uniform(0, 1)$), and a Uniform distribution between 0.5 to 1 ($Uniform(0.5, 1)$). We generate a curve for each query weight distribution using the VPI ordering strategy. Additionally, we show the curve for the *Random* strategy to provide a baseline. Changing the query weights used to generate the workload affects the overall utility of the dataspace; thus, in order to present all curves on the same graph, we report the percent of potential improvement in utility: i.e., the ratio of improvement between a dataspace with no confirmations and the perfect dataspace. The results are shown in Figure 5.

Observe that while the VPI strategy performs best with the highly-skewed $Pareto$ query workload, its effectiveness on all other workloads is close to that of the $Pareto$ workload. Thus, across a range of query workloads in this scenario, the VPI strategy effectively orders candidate matches for confirmation.

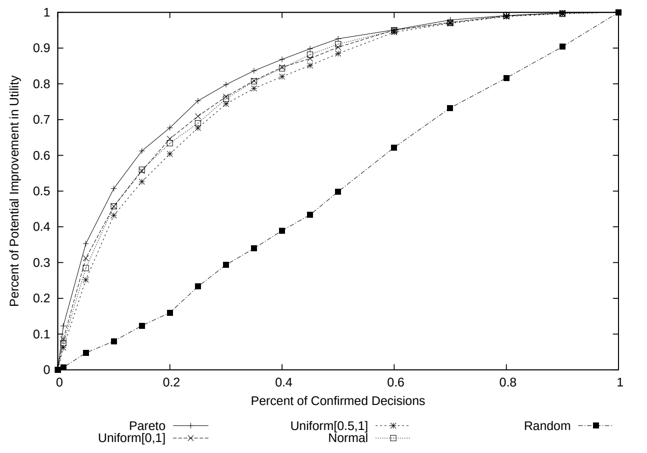

Figure 5: The effect of different query workloads on the performance of the VPI strategy run over the *Full* dataset.

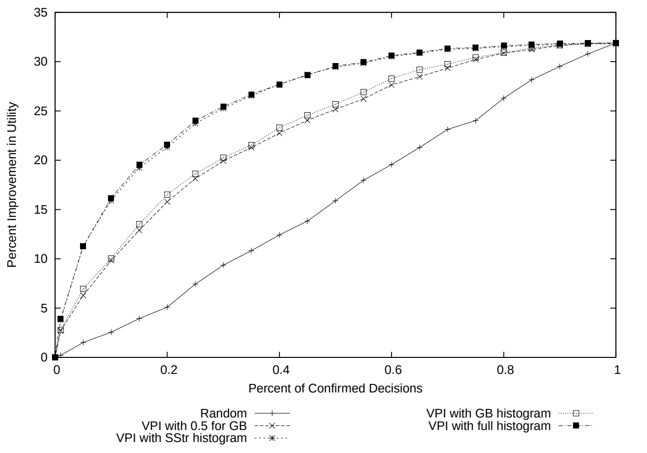

Figure 6: The performance of the VPI strategy using different means of assigning probabilities to matches, run over the *Full* dataset.

Different Means of Assigning Confidence

Recall that Google Base does not assign confidences to its adjustments and the confidences for the matches produced by SecondString are not accurate as probabilities. In the experiments above, we used the histogram-based approach for converting confidences to probabilities as discussed in Section 4.1. Here, we investigate the effectiveness of this approach on the VPI-based strategy's performance.

We compare the histogram-based technique as used in the previous experiments (labeled here as *VPI with full histogram*) to three other VPI-based approaches and the *Random* strategy (shown in Figure 6). In *VPI with 0.5 for GB*, the confidences for the matches produced by Google Base set to 0.5, while the confidences for the SecondString matches are left as is. This strategy is the baseline strategy. In *VPI with GB histogram*, the Google Base matches are converted to probabilities using the histogram technique while the SecondString matches are left as is. In *VPI with SStr histogram*, the SecondString matches are converted using the histogram technique while the confidence for the Google Base matches are left at 0.5. We also experimented with different bucket sizes for the histogram technique. As expected, the effectiveness is higher at smaller bucket sizes, though there was not a marked difference. For all experiments with the histogram approach, we use a bucket size of 0.01.

The results of this experiment are presented in Figure 6. First, observe that the curve for *VPI with 0.5 for GB* provides the least benefit for ordering confirmations. This performance is due to the inaccurate confidences produced by SecondString and the lack of confidences for Google Base matches. A slight improvement occurs when using the histogram approach to convert the confidence for the Google Base matches (*VPI with GB histogram*). The improvement is small because the conversion is very coarse-grained: all Google Base matches have no input confidence and thus get assigned to the same histogram bucket resulting in all Google Base matches getting the same output probability. When the SecondString match confidences are converted to probabilities, we see a large jump in the VPI strategy's effectiveness (*VPI with SStr histogram*). Here, the histogram approach is able to map input confidences to probabilities at a very fine resolution. Finally, when confidences for both SecondString and Google Base matches are converted (*VPI with full histogram*), the VPI strategy has the best performance, though its performance is only slightly better than just converting the SecondString matches due to the issues with the Google Base matches mentioned above. This experiment shows that the histogram-based technique for converting confidences to probabilities can provide a substantial increase in effectiveness for the VPI strategy, but in order to realize its full potential it is better to have initial confidences assigned per match.

4.2 Synthetic Data Experiments

The experiments above show that the VPI-based strategy is effective in two real-world dataspace scenarios derived from Google Base data. In order to validate our VPI-based strategy in an even wider range of scenarios, we built a dataspace generator capable of creating dataspaces with different characteristics.

Results on Synthetic Datasets

Using the dataspace generator, we ran a series of tests that evaluated the VPI strategy under a variety of dataspace environments. We highlight some of the findings from these experiments here.

Basic tests. We generated a dataspace that recreates a realistic large-scale data integration environment using realistic values or distributions for different characteristics: e.g., 100000 elements and a zipfian distribution for the query weights and items per element. In this dataspace, the VPI strategy is able to produce a dataspace whose utility remains within 5% of the utility produced by the oracle strategy.

Robustness tests. The VPI strategy is robust to variations in the dataspace characteristics: manipulating one characteristic of the dataspace (e.g., matches per element, items per element) while leaving the others at the realistic values used in the basic tests above has very little effect on the efficacy of the VPI strategy. For instance, to test the effect of differing degrees of heterogeneity, we generate dataspaces with different distributions for the number of matches in which an element participates. Regardless of the distribution used in this experiment, the utilities produced by the VPI strategy are within 10% of the percent of potential improvement in utility (as described in the query workload experiment in the previous section). We varied other parameters of the dataspace and ran similar experiments; all experiments produced results similar to the first experiment. In particular, we varied the distributions for the number items in which each element appears, introduced errors into element cardinality statistics, used different mechanism accuracies, and used different distributions for the query weights (as investigated above with the Google Base datasets).

Parameter exploration tests. By setting all parameters but one to trivial constants (e.g., 1 item per element, 0.5 query weight), we can study the effect that one parameter has on the VPI strategy's effectiveness. We can then determine in what environments it is particularly important to employ an intelligent ordering mechanism for user feedback.

Our experiments reveal that in cases where there is a wide range or high skew in the values for a particular parameter, the benefit provided by the VPI strategy is greater: it is able to effectively determine the matches that provide the most benefit and confirm them first.

For instance, when we manipulate the query weight distribution (and set all other parameters to trivial constants), the percent of potential improvement in the dataspace by the VPI strategy after confirming 10% of the matches in a dataspace with a zipfian query weight distribution is 0.35, whereas with a uniform distribution between 0 and 1 for the query weights, the percent of potential improvement is only 0.19. With smaller ranges in the query weights, the improvement is even less. Note that these results illustrate the potential benefit of employing an intelligent ordering strategy in environments such as Web search where some queries are orders of magnitude more frequent than others. On the other hand, in environments with more homogeneous queries, the selection method is less important.

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 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 a 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 [34][2], 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:

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

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

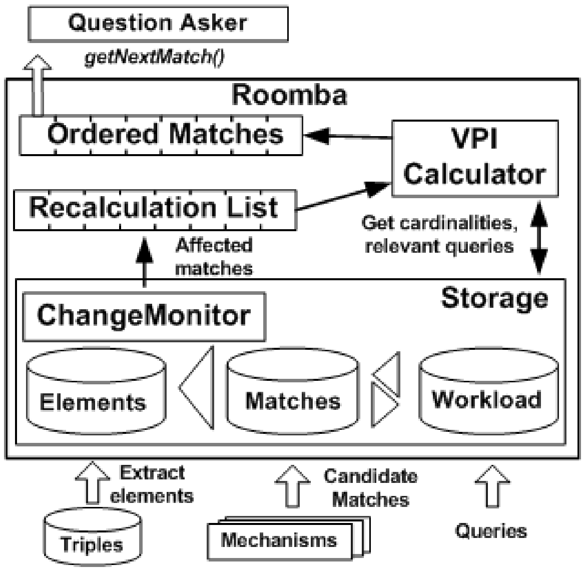

Figure 7: Roomba architecture

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

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

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.

6. ROOMBA

In this section, we outline the architecture of Roomba[3], a component of a dataspace system that incorporates the decision-theoretic framework presented in the previous sections to provide efficient, pay-as-you-go match ordering using VPI. The architecture of Roomba is shown in Figure 7.

To facilitate a pay-as-you-go mode of interaction, a dataspace system contains a user interaction module that determines the appropriate time to interrupt the user with confirmation request, such as in [14]. 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 $getNextMatch()$.

[3]The name "Roomba" alludes to the vacuuming robot of the same name [16]. Just as the robot discovers what needs to be cleaned in your room, the Roomba system aids a dataspace system in determining what needs to be cleaned in the dataspace.

In a pay-as-you-go dataspace, however, $getNextMatch()$ is called over time as the system runs; thus, a particular ordering of matches derived at one point of time using one 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's scores.

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

Decision theory. While we have based our decision-theoretic framework on formalisms used in the AI community [26], decision theory and the value of perfect information are well-known concepts in many fields such as economics [23, 21] and health care [3]. Within the data management community, there has been work on applying expected utility to query optimization [5].

Solicitng user feedback. Previous work on soliciting user feedback in data integration systems has focused on the output of a single mechanism. The work in [8] and [37] addresses incorporating user feedback into schema matching tasks. Similarly, [27] introduces an active-learning based approach to entity resolution that requests user feedback to help train classifiers. Selective supervision [18] combines decision theory with active learning. It uses a value of information approach for selecting unclassified cases for labeling. 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 [22] 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.

Data integration platforms. Recent research has proposed new architectures for dealing with large-scale heterogeneous data integration. The work presented here falls under the dataspaces vision [12], a new data integration paradigm for managing complex information spaces. As part of the dataspaces agenda, previous work has proposed the PAYGO [20] architecture for integrating structured and unstructured data at Web-scale. Similarly, the CIMPLE project [10] is developing a software architecture for integrating data from online communities. Roomba is designed to be a component within these systems to provide guidance for user feedback. In [28] the authors show how to bootstrap a dataspace system by completely automating the creation of a mediated schema and the semantic mappings to the mediated schema. Our work dovetails with [28] by showing how to subsequently resolve any additional semantic heterogeneity in a dataspace.

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 selecting 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 on real and synthetic datasets that validated the benefits of our framework.

Here, we outline some future research directions. 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 deal with *imperfect* user feedback. Here, we assumed that users answered correctly every time: a confirmation meant that the match was correct in reality. Users, however, are human and may not always be correct: matches may be ambiguous or challenging to answer correctly, or users may be malicious. To cope with uncertainty in user feedback, the dataspace could ask the same confirmation of multiple users and employ a majority voting scheme. More advanced approaches involve modeling user responses as probabilistically related to the true answer of the match and then adjusting the confidence of a match on confirmation [26].

Another area of future work is to explore other types of user feedback. In this paper, we explored how to efficiently involve users in resolving uncertainty through *explicit* user feedback. The dataspace system can also leverage the wealth of research on *implicit* feedback (e.g., [17, 6, 24]) 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 dataspace system to increase the confidence of those matches. A system can also use information from subsequent queries, or query chains [24], 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 system 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 dataspace system 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*) [26]. Intuitively, employing MEU for the decision to utilize or disregard a match for a query involves choosing the action with the highest expected utility.

9. ACKNOWLEDGEMENTS

The authors would like thank Jayant Madhavan, Omar Benjelloun, the Google Base team, and the members of the Berkeley database research group for many discussions regarding this work. This work was funded in part by NSF grant IIS-0415175.

10. REFERENCES

[1] Omar Benjelloun, Hector Garcia-Molina, Hideki Kawai, Tait Eliott Larson, David Menestrina, Qi Su, Sutthipong Thavisomboon, and Jennifer Widom.

Generic entity resolution in the serf project. *IEEE Data Eng. Bull.*, 29(2):13–20, 2006.

[2] George Casella and Roger Berger. *Statistical Inference*. Duxbury, 2002.

[3] Gretchen B. (Editor) Chapman and Frank A. (Editor) Sonnenberg. *Decision Making in Health Care: Theory, Psychology, and Applications*. Cambridge University Press; New Ed edition (September 1, 2003), 2003.

[4] Surajit Chaudhuri, Gautam Das, and Utkarsh Srivastava. Effective use of block-level sampling in statistics estimation. In *SIGMOD '04*, 2004.

[5] Francis Chu, Joseph Y. Halpern, and Praveen Seshadri. Least expected cost query optimization: an exercise in utility. In *PODS '99*, 1999.

[6] Mark Claypool, Phong Le, Makoto Wased, and David Brown. Implicit interest indicators. In *Intelligent User Interfaces*, pages 33–40, 2001.

[7] W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI*, 2003.

[8] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *SIGMOD '01*, 2001.

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

[10] AnHai Doan, Raghu Ramakrishnan, Fei Chen, Pedro DeRose, Yoonkyong Lee, Robert McCann, Mayssam Sayyadian, and Warren Shen. Community information management. *IEEE Data Eng. Bull.*, 29(1):64–72, 2006.

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

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

[13] Google Base. http://base.google.com.

[14] Eric Horvitz, Carl Kadie, Tim Paek, and David Hovel. Models of attention in computing and communication: from principles to applications. *Commun. ACM*, 46(3):52–59, 2003.

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

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

[17] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR '05*, 2005.

[18] Ashish Kapoor, Eric Horvitz, and Sumit Basu. Selective supervision: Guiding supervised learning with decision-theoretic active learning. In *IJCAI*, pages 877–882, 2007.

[19] Jayant Madhavan, Alon Y. Halevy, Shirley Cohen, Xin Luna Dong, Shawn R. Jeffery, David Ko, and Cong Yu. Structured data meets the web: A few observations. *IEEE Data Eng. Bull.*, 29(4):19–26, 2006.

[20] Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin (Luna) Dong, David Ko, Cong Yu, and Alon Halevy. Web-scale data integration: You can only afford to pay as you go. In *CIDR*, 2007.

[21] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford, 1995.

[22] Robert McCann, AnHai Doan, Vanitha Varadaran, Alexander Kramnik, and ChengXiang Zhai. Building data integration systems: A mass collaboration approach. In *WebDB*, 2003.

[23] Oskar Morgenstern and John Von Neumann. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[24] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2005.

[25] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDBJ*, 10(4):334–350, 2001.

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

[27] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *KDD '02*, 2002.

[28] Anish Das Sarma, Luna Dong, and Alon Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD '08*, 2008.

[29] SecondString Project Page. http://secondstring.sourceforge.net/.

[30] Craig Silverstein, Monika Henzinger, Hannes Marais, and Michael 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.

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

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

[33] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David E. Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A Macroscope in the Redwoods. In *SenSys*, pages 51–63, 2005.

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

[35] Luis von Ahn and Laura Dabbish. Labeling Images with a Computer Game. In *ACM CHI*, 2004.

[36] William E. Winkler. The state of record linkage and current research problems. Technical Report Statistical Research Report Series RR99/04, U.S. Bureau of the Census, Washington, D.C., 1999.

[37] Wensheng Wu, Clement Yu, AnHai Doan, and Weiyi Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD '04*, 2004.