An Interactive Clustering-based Approach to Integrating Source Query Interfaces on the Deep Web

Wensheng Wu
Computer Science Dept.
University of Illinois at Urbana-Champaign
wwu2@uiuc.edu

Clement Yu
Computer Science Dept.
University of Illinois at Chicago
yu@cs.uic.edu

AnHai Doan
Computer Science Dept.
University of Illinois at Urbana-Champaign
anhai@cs.uiuc.edu

Weiyi Meng
Computer Science Dept.
SUNY at Binghamton
meng@cs.binghamton.edu

ABSTRACT

An increasing number of data sources now become available on the Web, but often their contents are only accessible through query interfaces. For a domain of interest, there often exist many such sources with varied coverage or querying capabilities. As an important step to the integration of these sources, we consider the integration of their query interfaces. More specifically, we focus on the crucial step of the integration: accurately matching the interfaces. While the integration of query interfaces has received more attentions recently, current approaches are not sufficiently general: (a) they all model interfaces with flat schemas; (b) most of them only consider 1:1 mappings of fields over the interfaces; (c) they all perform the integration in a blackbox-like fashion and the whole process has to be restarted from scratch if anything goes wrong; and (d) they often require laborious parameter tuning. In this paper, we propose an interactive, clustering-based approach to matching query interfaces. The hierarchical nature of interfaces is captured with ordered trees. Varied types of complex mappings of fields are examined and several approaches are proposed to effectively identify these mappings. We put the human integrator back in the loop and propose several novel approaches to the interactive learning of parameters and the resolution of uncertain mappings. Extensive experiments are conducted and results show that our approach is highly effective.

1. INTRODUCTION

With the increasing number of data sources available over the Web, the integration of these sources is clearly an important problem. It has been observed that a large number of data sources on the Web are actually hidden behind query interfaces [3, 15]. For a domain of interest, there often exist numerous data sources on this "deep" Web, which provide similar contents but with varied coverage or querying capabilities. And it is often the case that the information a user desires is distributed over many different sources. Consequently, the user has to access each of these sources individually via its query interface, in order to find the desired information. Thus, an important first step to the integration of these hidden sources is the integration of their query interfaces.

SIGMOD 2004 June 13-18, 2004, Paris, France.

Given a set of query interfaces in a domain of interest, we aim to provide the users with a unified query interface which combines sufficiently similar fields in these interfaces, retains fields specific to some interface, and has a layout which preserves the ordering of the fields and the structure of the source query interfaces as much as possible.

The integration of source query interfaces can be divided into two steps [11, 29]. At the first step, semantic mappings of fields over different interfaces are identified; and at the second step, the interfaces are integrated based on the identified mappings of fields at the first step. Clearly, the accuracy of field mappings as the output from the first step is of crucial importance to a successful integration of the interfaces. In this paper, we will focus on the first step, that is, to accurately identify the mappings of fields.

Although schema integration is a well-studied problem [6, 7, 14, 16, 18, 19, 25, 29], the integration of query interfaces on the Web has just received more attentions recently [10, 11]. Unfortunately, current solutions suffer from the following limitations. First, *non-hierarchical modeling:* All current solutions model interfaces with *flat* schemas. In fact, we show that interfaces have much richer structure. Second, *1:1 mapping assumption:* Most of the current solutions only consider 1:1 mappings of fields over the interfaces. In fact, we show that more complex mappings are pervasive. Third, *blackbox operation:* All current solutions perform the matching and integration of interfaces in a blackbox-like fashion, in which the integrator is typically only an observer once the integration process starts. And the whole process has to be restarted if anything is found wrong. Fourth, *laborious parameter tuning:* Most of the current "automatic" solutions typically have a large set of parameters to be tuned in order to yield good performance for a specific domain. Furthermore, when the system is applied to another domain, these parameters frequently need to be re-tuned. The tuning process is often done in a trial-and-error fashion, without any principled guidance.

In this paper, we propose an interactive, clustering-based approach to address these limitations. The major contributions of our approach are:

- **Hierarchical modeling:** We show that a hierarchical ordered schema, such as an ordered tree, can capture the semantics of an interface much better. Furthermore, we show that the structure of interfaces can be exploited to help identify mappings of fields;

- **Clustering:** We examine the "bridging" effect in the process of matching fields from a large set of query interfaces [10]. The bridging effect is similar to the idea of reusing past identified mappings [25]. For example, two semantically similar fields, e_1 and e_2, might not be judged as similar based on their own evidences. Nevertheless, both of them might be similar to a third field e_3 in different aspects. Thus, by considering matching all three fields together, e_3 can be effectively used to suggest the mapping between e_1 and e_2. We will show how the clustering algorithm exploits this observation to help identify 1:1 mappings of fields;

- **Complex mapping:** We show that complex mappings, such as 1:m mappings, frequently occur among fields in different interfaces. We propose several approaches to finding complex mappings by exploiting the hierarchical nature of interfaces as well as the characteristics of fields. We further show that how the proposed clustering algorithm can be extended to effectively identify complex mappings among fields over *all* interfaces;

- **User interaction and parameter learning:** We put the human integrator back in the loop. We present a novel approach to learning the threshold parameter in the integration system by *selectively* asking the integrator certain questions. We further propose several approaches to determining the situations where interactions between the system and the integrator are necessary to resolve uncertain mappings, with the goal of reducing the amount of user interaction to the minimum. To the best of our knowledge, this is the first paper on the active learning of parameters in a schema matching algorithm.

The rest of the paper is organized as follows. Section 2 describes the hierarchical modeling of interfaces. Section 3 discusses the challenges in interface matching. The clustering algorithm for finding 1:1 mappings of fields is presented in Section 4.2. Section 4.3 extends the clustering process to handle complex mappings. User interaction and parameter learning are presented in Section 5. Section 6 reports results of our experiments. Section 7 contrasts our work with related work. The paper is concluded in Section 8.

2. HIERARCHICAL MODELING OF QUERY INTERFACES

In this paper, we start by considering query interfaces in HTML forms. Most of the query interfaces to hidden sources use HTML forms. We use *field* to refer to the building block of query interfaces. Typically, a field is used to request one piece of information from the user. A field can be in varied formats. In HTML forms, possible formats of a field are: text input box, selection list, radio button, and check box. A text input box is typically rendered as an empty box where the user can enter a suitable string. In a selection list, a list of choices is provided to the user to select from. There are two types of selection lists. In a single selection list, the user may only select one of the choices at the same time; while in a multiple selection list (or box), more than one choices may be selected at once. Radio buttons and checkboxes are used to explicitly display the choices to the user to facilitate the selection. The difference is that when used in a group, choices are exclusive in a radio button group, while multiple check boxes may be selected at the same time in a check box group. Thus, the former can be regarded as a variant of a single selection list, while the latter as a variant of a multiple selection list. In summary, there are two broad types of fields: one without pre-specified values, such as text input boxes; and the other with a set of values given to facilitate the input, such as fields in all other formats in HTML forms.

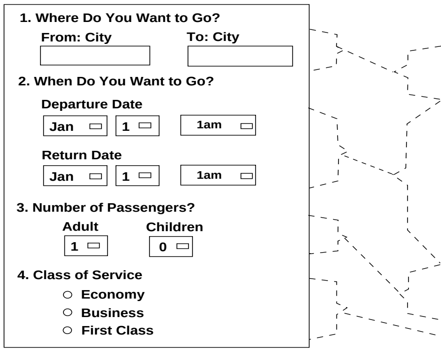
Each field in the interface typically has a label attached, describing to the user what the field is about. But it is possible that some fields do not have their own labels, rather they share a group label with other related fields. In these cases, the meaning of the field is conveyed to the user by the group label and the values in the field. Each field is also assigned with a name in the script for identification purpose (such a name is known as the internal name of the field). Related fields are usually placed close to each other in the interface, forming a group, and several related groups can further form a super-group, resulting in a hierarchical structure of the interface.

Note that since labels are visible to the user but names are not, as a consequence, words in the labels are usually ordinary words which can be understood semantically, while words in the names are often concatenated or abbreviated. Nevertheless, in our experiments we find that, in many cases, the name of a field can be very informative and especially useful when the label of the field is absent. See, for example, fields f_3–f_8 in Example 1 below. To capture the semantics of fields, we define the following properties for each field.

Definition 1: (Properties of Field) A field f has three properties: name(f), label(f), and dom(f). name(f) is the name of f; label(f) is the label attached to f in the interface or empty string if no label is associated with f; and dom(f) is the domain of f, which contains a set of valid values which f may take. ■

Example 1: The left portion of Figure 1 shows a query interface in the airfare domain. It expects several categories of information from the user including the location, the date, and the service type for the trip. The right portion of Figure 1 tabulates the fields in the interface. Fields are numbered in the order of their appearances in the interface. For each field, its name, label, and instances are shown. For example, the name of f_1 is origin, its label is From: City, and it accepts an arbitrary string. Note that some fields (e.g., f_3) do not have labels themselves, rather they share a group label with other fields (e.g., f_4 and f_5) and each field might have a list of values suggesting what the field is about. ■

To capture the structure of a query interface, we model the interface with a hierarchical schema which is essentially an ordered tree of elements. A leaf element in the tree corresponds to a field in the interface. An internal element corresponds to a group or a super-group of fields in the interface. Elements with the same parent are sibling elements. Sibling elements are ordered by the sequence of their corresponding fields or groups of fields (if they are internal elements) appearing in the interface. Thus, the order semantics and the nested grouping of the fields in an interface are captured in



Field	Name	Label	Domain
f1	origin	From: City	{s | s is any string}
f2	destination	To: City	{s | s is any string}
f3	departureMonth	""	{Jan, Feb, ..., Dec}
f4	departureDay	""	{1, 2, ..., 31}
f5	departureTime	""	{1am, 2am, ..., 12pm}
f6	returnMonth	""	{Jan, Feb, ..., Dec}
f7	returnDay	""	{1, 2, ..., 31}
f8	returnTime	""	{1am, 2am, ..., 12pm}
f9	numAdultPassengers	Adult	{1, 2, ..., 6}
f10	numChildPassengers	Children	{0, 1, ..., 5}
f11	cabinClass	Class of Services	{Economy, Business, First Class}

Figure 1: A query interface and its fields

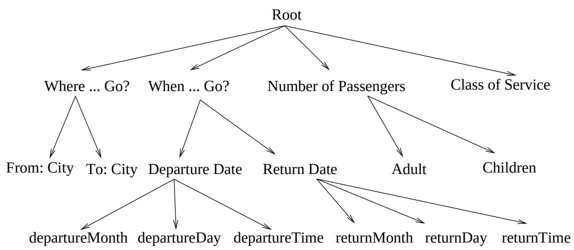


Figure 2: A schema tree

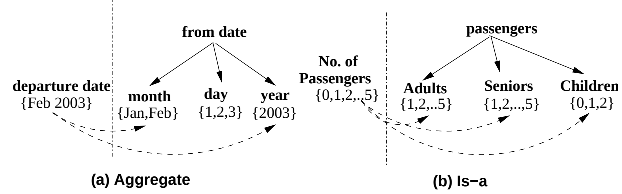


Figure 3: 1:m mappings

the structure of the schema tree corresponding to the interface.

Example 2: Figure 2 shows the schema tree corresponding to the query interface in Figure 1. The schema tree has four levels. The first level contains a generic root element that has four children, the first of which corresponds to the first category in the search form, i.e., the origin and destination cities. Each level except for the first level refines the element(s) at the level above. Note that each node in the tree except for the root is annotated with the label of its corresponding field or group of fields (for a field without a label, its name is shown instead). ■

3. INTERFACE MATCHING, CHALLENGES AND OUR APPROACH

We define interface matching as a problem of identifying semantically similar fields over different query interfaces. From the set of query interfaces we collected (see Section 6 for more details), we observe two broad types of semantic correspondence between fields in different interfaces: *simple* (or 1:1) mappings and *complex* (such as 1:m) mappings. In this section, we examine each type of mappings, discuss the challenges in identifying these mappings, and give an overview of our approach.

3.1 Simple Mappings

A simple mapping is a *1:1* semantic correspondence between two fields in different interfaces. For example, two interfaces in the automobile domain can both have a field for the make of automobiles.

The major challenge to the identification of 1:1 mappings is the *label mismatch* problem. Label mismatch occurs when similar fields in different interfaces are attached with different labels. For instance, the following are some example labels from several real-world query interfaces in the airfare domain, used to annotate fields for the service class: class of service, class of ticket, class, cabin, preferred cabin, and flight class. Note that the label could be just a single word such as class and cabin; a phrase such as class of service and preferred cabin; or a sentence. The labels for semantically similar fields often share common words (e.g., class appears in four of the six labels above) or they might be synonyms (e.g., cabin and class). But note that a general-purpose semantic lexicon such as WordNet [8] does not help much in the identification of domain-specific synonyms. For example, it is difficult to infer from WordNet that cabin is synonymous to class in the context of airline ticket reservation. Furthermore, domain-specific lexicons are not generally available and they can be expensive to build.

3.2 Complex Mappings

While 1:1 mappings account for the majority of the mappings of fields, we found that 1:m mappings occur in every domain we studied and very frequently in some domains. A *1:m* mapping refers to that one field in an interface semantically corresponds to multiple fields in another interface. Most of the current solutions to interface matching [10, 11] only consider 1:1 mappings, thus largely simplify the problem.

We observe that there exist two types of 1:m mappings between fields: *aggregate* and *is-a*. For both types, fields on the many side of the mapping refine the field on the one side. The difference is: in the aggregate type, the content of a field on the many side is *part of* the content of the field on the one side; while in the is-a type, the content of the field on the one side is typically the *union* (or sum) of the content of fields on the many side. Figure 3 shows examples of 1:m mappings of both types with interfaces in the airfare domain. Note that for the fields on the many side, we also show their parent node. (The reason for this will become clear soon.)

Finding 1:m mappings is even more challenging than finding 1:1 mappings. Note that domain-specific concept hierarchies might help but again they are rarely available and also expensive to construct manually.

3.3 Toward a Highly Accurate Field Matching

Handling 1:1 mappings: To cope with the label mismatch, we exploit the "*bridging*" effect achieved by considering matching fields of all interfaces *at once* rather than *pairwise* [10]. This can be illustrated with the following example. Note that the bridging effect is similar to the idea of reusing existing mappings [25]. (See our related work section for more details.) In Section 4.2, we propose a clustering algorithm which exploits exactly this observation to accurately identify 1:1 mappings of the fields.

Example 3: Consider three fields, e, f, and g, from three different interfaces in the computer hardware domain. Suppose label(e) = cpu, dom(e) = {celeron, pentium, duron}; label(f) = processor, dom(f) contains any strings; and label(g) = processors, dom(g) = {athlon, celeron, pentium, xeon}. For the simplicity, we assume their names are all dissimilar to each other. We note that neither the labels nor the domains of e and f are similar. But the domain of e is similar to the domain of g while the label of f is similar to the label of g. Thus, g may serve as a potential "bridge" to relate e and f, making them similar. ■

Handling 1:m mappings: To cope with the lack of domain-specific concept hierarchy, we exploit the following observations to help identify 1:m mappings of fields: (1) *Value correspondence:* The characteristics of values in the fields involved in a 1:m mapping as described above can be utilized to suggest the mapping. (2) *Field proximity*: Fields on the many side are typically very close to each other in the interface, which can be exploited to reduce the search space. (3) *Label similarity:* The label of the field on the one side often bears similarity with the *parent* label of fields on the many side, which can be used to further improve the matching accuracy. In Section 4.3, we will present our approach to finding 1:m mappings and describe how the clustering algorithm can be extended to handle 1:m mappings.

User interactions: The automatic solution for the field matching proposed above has several parameters to be manually set as is typical of many other similar automatic schema matching approaches [11, 18]. In Section 5, we first propose an approach to learning the parameters with a small amount of user interaction. We then examine the errors made by the field matching algorithm, determine the uncertainties during the matching process which might cause these errors, and propose several approaches to resolving these uncertainties with user's help. Our goal is to achieve a highly accurate field matching with a minimum amount of user interaction.

4. FIELD MATCHING VIA CLUSTERING

4.1 Field Similarity Function

As proposed earlier, each field is characterized by three properties: name, label, and domain. The semantic similarity of two fields is evaluated on the similarity of their properties. More specifically, we compute an aggregate similarity of two fields, e and f, denoted as $\mathcal{AS}(e, f)$, based on two component similarities, the linguistic similarity and the domain similarity, as follows:

$$\lambda_{ls} * \text{lingSim}(e, f) + \lambda_{ds} * \text{domSim}(e, f), \qquad (1)$$

where λ_{ls} and λ_{ds} are two weight coefficients, reflecting the relative importance of the component similarities.

4.1.1 The Linguistic Similarity

The name and the label of a field can be regarded as two description-level properties of the field. Two fields are linguistically similar if they have similar names or labels.

Fundamental to the computation of linguistic similarities is the measure of the similarity of two strings of words. For this, we employ the Cosine [4, 27] function in Information Retrieval which we now briefly describe. Suppose s and p are two strings of words. First, stop words (or non-content words) such as "the", "a", and "an", are removed from each string. Suppose the number of distinct content words (or terms) in the resulted strings is m. We then represent each string as an m-dimensional vector of terms with weights, e.g., $\vec{s} = (w_1, w_2, \ldots, w_m)$, where w_i is the number of occurrences of the i-th term t_i in the string s. Then, the similarity of the two strings, s and p, is computed as: $\text{Cosine}(\vec{s}, \vec{p}) = \vec{s} \cdot \vec{p} / (\|\vec{s}\| * \|\vec{p}\|)$. In the following, to simplify notations, we will also denote $\text{Cosine}(\vec{s}, \vec{p})$ as $\text{Cos}(s, p)$. In the experiments, words in strings are also stemmed [22].

The linguistic similarity of two fields, e and f, denoted as lingSim(e, f), is a weighted average of the similarities of their names, their labels, and their name vs. label:

$$\lambda_n * \text{nSim}(e, f) + \lambda_l * \text{lSim}(e, f) + \lambda_{nl} * \text{nlSim}(e, f), \qquad (2)$$

where nSim(e, f) is the name similarity of two fields, computed as Cos(name(e), name(f)); lSim(e, f) is their label similarity, similarly computed as Cos(label(e), label(f)); and nlSim(e, f) is their name vs. label similarity, given by: $\max\{\text{Cos}(\text{name}(e), \text{label}(f)), \text{Cos}(\text{label}(e), \text{name}(f))\}$. Note that if a field does not have a label itself, the label of its parent, if available, will be used instead.

Normalization: As mentioned earlier, names often contain concatenated words and abbreviations. Thus, they first need to be normalized before they are used to compute linguistic similarities. We apply the following normalizations [18, 30].

(a) *Tokenization* is used to cope with concatenated words. First, delimiter and case change in letters are used to suggest the breakdown of concatenated words. For example, "departCity" into "depart City", and "first_name" into "first name"; Second, we utilize words appearing in the labels to suggest the breakdown. A domain dictionary is constructed automatically for this purpose, which contains all the words appearing in the labels of fields in a set of interfaces from the same domain. For example, "deptcity" will be split into "dept" and "city" if "city" is a word in some label.

(b) *Transformation* is used to expand the abbreviations. For example, "dept" into "departure". We again utilize the domain dictionary constructed above to suggest the expansion. To avoid false expansions, we require that word to be expanded is not in the dictionary, with at least three letters, and having the same first letter with the expanding word.

4.1.2 The Domain Similarity

The domain similarity of two fields, e and f, is the similarity of their domains: dom(e) and dom(f). For the simplicity, we also denote the similarity of two domains, d and d', as domSim(d, d').

Simple Domain: We first consider simple domains where each value in the domain contains only one component. The simple domains can be of varied types. In this paper, we consider the following simple domain types: *time*, *money*, *area*, *calendar month*, *int*, *real*, and *string*. Usually, the domain type of a field is not specified in the interface. But it can often be inferred from the values in the domain. The inference is carried out via pattern matching: for each simple domain type, a regular expression is defined, which specifies the pattern of the values in the domain. For example, the regular expression for the *time* type can be defined as "[0-9]{2}:[0-9]{2}" which recognizes values of the form "03:15" as of the *time* type.

For some fields, especially those which do not have predetermined values, the label of the field might contain some information on the type of the values expected by the field. For example, a field whose label contains "$" or keyword "USD" expects an input of the monetary type. For all fields which we are not able to infer their domain types, we assume their domains are of *string* type with an infinite cardinality.

Similarity of Two Simple Domains: The similarity of two simple domains, d and d', denoted as $\text{domSim}(d, d')$, is judged on both the type of the domain and the values in the domain as follows:

$$\lambda_t * \text{typeSim}(d, d') + \lambda_v * \text{valueSim}(d, d'). \qquad (3)$$

For two domains of the same type, typeSim is 1 and we further evaluate their value similarity; otherwise, their similarity is defined to be zero. First, let's consider two character domains, d and d'. Suppose the set of values in d is $\{u_1, u_2, \ldots, u_m\}$ and similarly, $d' = \{v_1, v_2, \ldots, v_n\}$. All u_i's and v_j's are strings. With a desired threshold τ, we determine all pairs of similar values, one from each domain, by the following *Best-Match* procedure: (1) We compute the pairwise Cosine similarity for every pair of values, one from d and the other from d'. (2) The pair with the maximum similarity among all pairs is chosen and the corresponding two values are deleted from d and d'. (3) Repeat step 2 on all remaining values in the domains until no pair of values has a similarity greater than τ. Let the pairs of values chosen be C. The valueSim(d, d') is then computed with the Dice's function [5, 6] as $\frac{2*|C|}{|d|+|d'|}$.

For two numeric domains, we measure their value similarity by the percentage of the overlapping range of values in the domains. More specifically, the $\text{valueSim}(h,\ h')$ of two such domains, h and h', is evaluated as follows:

$$\frac{\min\{\max(h), \max(h')\} - \max\{\min(h), \min(h')\}}{\max\{\max(h), \max(h')\} - \min\{\min(h), \min(h')\}},$$

where $\min(x)$ and $\max(x)$ give the minimum and the maximum of the values in the domain x, respectively. Note that the numerator is the range where the two domains overlap and the denominator is the outer span of the two domains. For two identical domains, the similarity is 1. The similarity is defined to be zero if two domains do not overlap. For discrete numeric domains such as *int*, the above formula might underestimate their value similarity. For these domains, we adjust the formula by adding constant 1 to both the numerator and the denominator but only when the numerator is greater than zero.

For a field whose domain type is *string* with an infinite cardinality, we assume that its domain is dissimilar to the domain of any other field, be it finite or infinite.

4.2 Finding 1:1 Mappings

We employ a hierarchical agglomerative clustering algorithm [13] to identify 1:1 mappings of fields. The clustering algorithm is shown in Figure 4. It expects three inputs: a set of interfaces $\mathcal{S}$, the similarity matrix M of fields in $\mathcal{S}$, and a stopping threshold $\tau_c \geq 0$. The result of the clustering process is a partition over fields such that similar fields are in the same partition while fields in different partitions are dissimilar.

Cluster($\mathcal{S}$, M, τ_c) $\rightarrow P$:
(1) place each field in $\mathcal{S}$ in a cluster by itself.
(2) while there are two clusters with similarity $> \tau_c$,
  (a) choose two clusters, c_i and c_j, whose similarity is the largest over all pairs of clusters.
  (b) resolve the ties if necessary.
  (c) merge c_i and c_j into a new cluster c_k, and remove clusters c_i and c_j.
  (d) remove all rows and columns associated with c_i and c_j in M, and add a new row and column for c_k.
  (e) compute similarities of c_k with other clusters using Formula 4.
(3) return the clusters of fields.

Figure 4: The clustering algorithm

The similarity matrix M is obtained as follows. Suppose the total number of fields over all interfaces in $\mathcal{S}$ is m. We compute the aggregate similarity for every pair of fields in different interfaces, which results in an $m \times m$ symmetric matrix M, whose entry $M[i, j]$ gives the aggregate similarity between the i-th field and the j-th field. Since two fields, say the k-th and the l-th fields, from the same interface should not be merged, such an entry $M[k, l]$ is set to zero.

The algorithm starts by placing each field in a cluster by itself. Then, it repeatedly selects two clusters which have the largest similarity to merge until none of the remaining pairs of clusters has a similarity greater than the given threshold. In the following, we first discuss the greedy property of the algorithm and its implication to the field matching. We then give the formula for the cluster similarity. Finally, we discuss the tie resolution technique employed at step (2b) of the algorithm.

Greedy Matching: Typically, for each field e, there are a large number of candidate mappings from the interfaces other than the one which e belongs to, and we have to decide which choices are the "best". The well-studied matching problem in graph theory [17] provides a rich set of possible criteria such as: maximum cardinality, maximum total weight, and stable marriage. A matching has the maximum cardinality if it has the largest number of mappings; a matching has the maximum total weight if the sum of weights of its mappings is the largest; and the stable marriage property requires that there are no two mappings (x, y) and (x', y') such that x prefers y' to y and y' prefers x to x'. Empirical studies in [19] show that the *perfectionist egalitarian polygamy* selection metric (that is, no male or female is willing to accept any partner(s) but the best) produces best results in a variety of schema matching tasks. The *greedy choice* step of the clustering process for the identification of 1:1 mappings can be regarded as the monogamy version of this metric.

More specifically, the clustering process involves in the repeated application of the following two procedures: (i) *Greedy choice*: Two clusters, c_i and c_j, with the *maximum* similarity are chosen to merge into a new cluster c_k. All fields in c_k are considered to be 1:1 mapped to each other. (ii) *Constraint enforcement:* Consider two fields, f_i and f_j, in c_k and suppose that f_i belongs to interface S_u and f_j to interface S_v. Since all we consider here are 1:1 mappings, clearly f_i can not be mapped to any field in S_v other than f_j; similarly, f_j can not be mapped to any field in S_u other

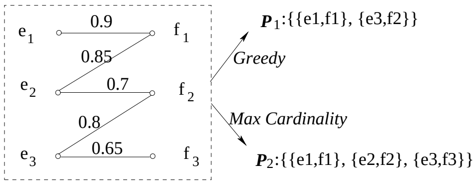


Figure 5: A matching example

than f_i. Procedure (i) corresponds to steps (2a–c) in Figure 4 and procedure (ii) corresponds to steps (2d–e), where Formula 4 is to be introduced below. It can be shown that the matching obtained by the greedy strategy is stable. The following example further illustrates the greedy strategy by contrasting it with the maximum cardinality criterion.

Example 4: Consider two interfaces, E and F, in Figure 5. E has three fields, e_1, e_2, and e_3; F also has three fields, f_1, f_2, and f_3. Suppose their aggregate similarities are as shown in Figure 5 in the form of the similarity graph and the clustering threshold $\tau_c = .6$. When the clustering begins, each field is in one cluster itself. For example, f_2 is in cluster $\{f_2\}$. The greedy strategy first chooses to merge $\{e_1\}$ and $\{f_1\}$ since they have the maximum similarity. Then, $\{e_2\}$ can not be merged with the resulted cluster $\{e_1, f_1\}$ by the constraint enforcement. This is achieved by removing the edges associating e_1 with all f_j's and f_1 with all e_i's.

Accordingly, $\{e_3\}$ and $\{f_2\}$ will be merged next since they have the maximum similarity among the remaining clusters. The final partition of fields resulted from the clustering process is P_1. It can be verified that the mappings in P_1 are stable. In contrast, the maximum cardinality criterion yields matching P_2 whose cardinality is 3, the largest size of matching possible for this example. Clearly, P_1 does not have the maximum cardinality. ■

The Cluster Similarity: Consider two clusters, c_i and c_j, each of which contains a set of fields. Suppose there are m fields: $\{c_{i_1}, c_{i_2}, \ldots, c_{i_m}\}$ in c_i, and n fields: $\{c_{j_1}, c_{j_2}, \ldots, c_{j_n}\}$ in c_j. The similarity between c_i and c_j, denoted as $\mathrm{Sim}(c_i, c_j)$, is then given as follows:

$$\max_{1 \le u \le m, 1 \le v \le n} \{\mathcal{AS}(c_{i_u}, c_{j_v})\}. \quad (4)$$

If a field in c_i and a field in c_j belong to the same interface, then $\mathrm{Sim}(c_i, c_j) = 0$. Formula 4 yields a single-link algorithm.

Ordering-based Tie Resolution: Tie occurs when there are more than one pairs of clusters with the same maximum similarity and there are at least one clusters involved in two or more such pairs. Intuitively, this situation occurs when the aggregate similarities alone are not sufficient for the proper identification of equivalent pairs of fields. Tie frequently occurs in some domains (see our experiment section for more details), thus a proper resolution of the ties is very important to the accuracy of field matching.

We resolve the ties by exploiting the order semantics of fields in the involved clusters. Consider a cluster c which has the maximum similarity with a set s of other clusters. First, we decide on two interfaces to be used as our reference to resolve the ties. The first of such interfaces, S_1, is chosen to be any interface whose fields appear in at least one of the clusters in s. We select any interface in cluster c to be our second reference interface, S_2. Next, we identify all pairs of fields having the maximum similarity, with each pair

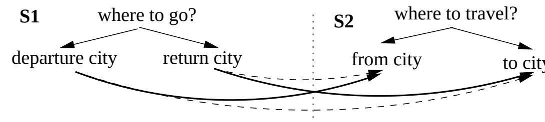


Figure 6: Tie resolution

containing a field from each of the two reference interfaces. We then select a reference field pair (e, f), where e from S_1 and f from S_2, such that no other field appearing before f in S_2 has the maximum similarity with e and vice versa (in other words, e and f are each other's first best choice). If such a pair is found, we merge the cluster where e appears with the cluster where f appears. Otherwise, we randomly choose two clusters with the maximum similarity to merge.

Example 5: Consider two such reference interfaces in the airfare domain as shown in Figure 6, where four pairs of fields have the maximum similarity. (departure city, from city) will then be chosen as the reference field pair. Note that this corresponds to the intuition that normally the departure information appears before the information on the returning trip in the interface. ■

4.3 Finding Complex Mappings

In this section, we extend the clustering process to handle 1:m mappings of the fields. Two additional phases are introduced: a *preliminary-1-m-matching* phase before the clustering process and a *final-1-m-matching* phase after the clustering process. In the preliminary-1-m-matching phase, we exploit the properties of the fields and the structure of the interfaces to identify an initial set of 1:m mappings. At the end of this phase, all fields which are involved in the one side of at least one 1:m mappings will be removed before the clustering algorithm is applied. After the clustering process is completed, the result from the preliminary-1-m-matching phase will be combined with the clustering result to obtain the final set of 1:m mappings. Figure 7 shows the complete field matching algorithm which we now describe in detail. We start with a necessary definition.

Definition 2: (Composite Domain & Field) A composite domain d of arity k is a set of ordered k-tuples, where the i-th component of the tuple is a value from the i-th subdomain of d, denoted as d_i. Each d_i is a simple domain. The arity of domain d is denoted as $\phi(d)$. A field is composite if its domain is composite. ■

The domain type of a composite domain is also composite, consisting of an ordered list of simple domain types, each defined for one of its sub-domains. A special composite type, *date*, is pre-defined.

To determine if a field is composite, we adopt a simplified *structure extraction* process as employed in [26] for data cleaning. In particular, we exploit the delimiters in the values to suggest the structure of the domain, where the delimiters we consider include punctuation characters, white spaces, and special words such as "to". For a field to be composite, we require that the overwhelming majority of the values of the field can be consistently decomposed into the same number of components. For the fields which do not have instances, we also try to exploit the format information which might appear in the label of the field to help determine if the field has a composite domain and the type of its sub-domains. For example, "mm/dd/yyyy" or "mm/dd/yy" are commonly used to annotate a field which

FieldMatch(S) $\rightarrow P$ and Q'
(1) /* **Preliminary-1-m-matching phase**: */
$Q \leftarrow$ IdentifyInitialOneToManyMappings(S)
(2) /* **Clustering phase**: */
(a) /* compute pairwise aggregate similarities of fields */
$M \leftarrow$ ComputeAggregateSimilarities(S)
(b) /* identify 1:1 mappings via clustering */
$P \leftarrow$ Cluster(S,M,τ_c)
(3) /* **Final-1-m-matching phase**:
combine P and Q to obtain final 1:m mappings */
$Q' \leftarrow$ ObtainFinalOneToManyMapping(P, Q)

Figure 7: The field matching algorithm

expects an input of the *date* type in the format: month, day, and year.

Similarity of Composite vs. Simple/Composite Domains: Consider two domains, d and d', at least one of which is a composite domain. In other words, either $\phi(d) > 1$, or $\phi(d') > 1$, or both. The similarity of two such domains (each of which can be a composite domain) is then evaluated based on the extent by which their sub-domains are similar. Since sub-domains are all simple domains, their similarity is evaluated by Formula 3 and they are determined to be similar if their similarity value, domSim, exceeds a threshold τ'. We employ the *Best-Match* procedure to determine the set of pairs of similar sub-domains: $\{(d_i, d'_j)|\text{domSim}(d_i, d'_j) > \tau'\}$, denoted as C'. The similarity of d and d' is then evaluated also via the Dice's function as: $\frac{2*|C'|}{\phi(d)+\phi(d')}$.

4.3.1 Identify a Preliminary Set of 1:m Mappings

Aggregate Type: To identify the initial set of aggregate 1:m mappings of fields, we proceed as follows. Consider all fields over all interfaces. For each field e in interface S, we first check if it is a composite field as described above. If e is composite, then in every interface other than S, denoted as X, we look for a set of fields $\mathbf{f} = \{f_1, f_2, \ldots, f_n\}$, where $n > 1$, such that the following conditions are satisfied:

1. f_i's are siblings, that is, they share the same parent p but the set of f_i's might be a proper subset of the set of all children of p.
2. The label of the parent of f_i's is highly similar to the label of e.
3. There is a subset s of sub-domains of domain of e such that there is a 1:1 correspondence between each sub-domain in s and the domain of some field f_j (or sub-domain if f_j is composite) in $\mathbf{f}$ in the sense that they have high similarity (according to Formula 3).

If there exists such a $\mathbf{f}$ in interface X, a 1:m mapping of aggregate type is then identified between e and fields in $\mathbf{f}$, denoted as $e \leftrightarrow \{f_1, f_2, \ldots, f_n\}$.

Note that the *field proximity* observation as discussed in Section 3.3 is exploited in condition (1): since the fields on the many side are closely related, they are typically placed close to each other in the interface, forming a group, and the fields of the same group in the interface are siblings in the schema tree. Note also that it is possible that the field on the one side only matches with some of fields in the same group. See Figure 3(a) for such an example where day field does not participate in the 1:m mapping shown in the figure. Further note that condition (3) essentially captures the *part-of* relationship between the content of a field on the many side and that of the field on the one side in a 1:m mapping of the *aggregate* type.

Is-a Type: The identification of aggregate 1:m mappings of fields relies on the detection of composite fields and then the discovery of the corresponding sub-fields in another interface. Typically, the domains of the sub-fields may be different. In contrast, the identification of is-a 1:m mappings of fields requires that the domain of each corresponding sub-field is similar to that of the general field. More precisely, for each non-composite field e', we check if there exists a set of fields $\mathbf{f} = \{f_1, f_2, \ldots, f_n\}$, $n > 1$, in another interface X, which meets the following conditions:

1. All f_i's are siblings and their parent does *not* have any children other than f_i's.
2. The label of the parent of f_i's is highly similar to the label of e.
3. The domain of *each* f_i is highly similar to the domain of e.

If yes, a 1:m mapping of is-a type is then identified between e' and fields in $\mathbf{f}$, denoted as $e' \leftrightarrow \{f_1, f_2, \ldots, f_n\}$.

Dealing with Infinite Domains: As described above, there are some fields which we are not able to infer their domain types and assume that they have domains of string type with an infinite cardinality. Since the similarity of an infinite string domain with any other domain is zero, the above procedures can not be employed.

To cope with this situation, we introduce an additional approach which utilizes the label information extensively to identify fields which map to several other fields in another interface. In detail, we consider all fields which are not involved in any 1:m mappings of both types identified above. For each such field g, we seek a set of *sibling* fields $\mathbf{f} = \{f_1, f_2, \ldots, f_n\}$, $n > 1$, such that *one* of the following two conditions is satisfied. (1) f_i's are the only children of their parent, p, and the label of g is *identical* to the label of p. (2) The label of g can be decomposed into several component terms with ',', '/', 'or' as delimiters, and the label of each f_i is one of the component terms in the label of g.

4.3.2 Obtain the Final 1:m Mappings of Fields

Our experiments show that the mappings identified in the preliminary-1-m-matching phase are quite accurate (i.e. of high precision). But there are cases where direct evidences between the fields involved in a 1:m mapping might not be sufficient to meet the required conditions as described above. As a result, these mappings fail to be identified, reducing the recall. To cope with this, in the *final-1-m-matching* phase, an inference process is carried out where the 1:m mappings identified in the premilinary-1-m-matching phase are combined with the 1:1 mappings identified in the clustering process to infer additional 1:m mappings. We also require that the fields on the many side of new 1:m mappings are siblings. We use the following example to illustrate this inference process.

Example 6: Suppose the preliminary-1-m-matching phase is able to identify a 1:m mapping $a \leftrightarrow \{b_1, b_2\}$, where a comes from interface A, and both b_1 and b_2 from interface B. Suppose further that the clustering process discovers two 1:1 mappings: $b_1 \leftrightarrow c_1$ and $b_2 \leftrightarrow c_2$, where c_1 and c_2 are from interface C. Then, a new 1:m mapping $a \leftrightarrow \{c_1, c_2\}$ will be inferred by the final-1-m-matching phase, given that c_1 and c_2 are siblings in interface C. ■

5. USER INTERACTIONS

Our experiments show that the automatic field matching algorithm proposed above can achieve high accuracy over different domains. As typical of many schema matching algorithms, the algorithm first requires a set of parameters to be manually set, then it proceeds to the end without human's intervention. Since these parameters are often domain-specific or even field-specific, when the system is applied to a different domain, a different set of parameters may need to be used. Usually, these parameters are tuned in a trial-and-error fashion with no principled guidance. Furthermore, since there might not exist a *best* set of parameters or a perfect similarity function, errors might still occur: (1) some mappings might fail to be identified (i.e. *false negatives*); and (2) some identified mappings are not correct (i.e. *false positives*).

In this section, we make our field matching algorithm interactive by putting the human integrator back in the loop. We first propose a novel approach to learning the parameters by selectively asking the user (that is, the integrator) some questions. We then propose several approaches to reducing errors in both 1:1 and 1:m mappings with user's help. For a question on 1:1 mapping, we present the corresponding pair of fields to the user, showing both their labels and instances, and the user only needs to give "yes"/"no" responses. For a question on 1:m mapping, all fields on the many side of the suggested mapping are shown to the user. The empirical evaluation of the user interactions will be given in Section 6.

5.1 Parameter Learning

First, we observe that the field similarity, denoted as fs, is actually a linear combination of the component similarities: cs_i's. That is, $\mathrm{fs} = a_1 * \mathrm{cs}_1 + a_2 * \mathrm{cs}_2 + \cdots + a_n * \mathrm{cs}_n$, where a_i's are weight coefficients reflecting the relative importance of different component similarities. And the field matching algorithm can be regarded as a thresholding function: $\mathrm{fs} > \tau$, or $a_1 * \mathrm{cs}_1 + a_2 * \mathrm{cs}_2 + \cdots + a_n * \mathrm{cs}_n > \tau$. Two fields are judged to be similar if their field similarity $\mathrm{fs} > \tau$; and not otherwise.

Consider a simple case where the field similarity fs has only two component similarities, cs_1 and cs_2. Figure 8 plots the distribution of field similarities in two dimensions, one for each component similarity. A point is shown in '+' sign if two fields with the corresponding component similarities indeed match; and in '-' sign otherwise. If the component similarity functions are reasonably accurate in capturing the similarity of fields, we expect a typical distribution as shown in the figure. That is, matching fields typically have at least one large component similarities and non-matching fields normally would have low values in both of their component similarities. Clearly, a good thresholding function should be such a dividing line that the majority of positive points lie *above* it and the majority of negative points lie *below* it. There are different ways of learning a thresholding function. Here, we propose an approach to learning the threshold τ, while a_i's are set to some *domain-independent* empirical values (see Section 6 on how this is done in our experiments).

Learning the Threshold: We further observe that there are only two possibilities for each field in an interface: either it matches with *some* field in some other interface, or it does not match with *any* field in any other interface. Assuming that our similarity function is reasonably accurate, we will have relatively larger similarity values for the fields in the former case; and relatively smaller similarity values for the fields in the latter case. In other words, there will be a gap between these two types of similarity values and a good threshold can be set to any value within this gap.

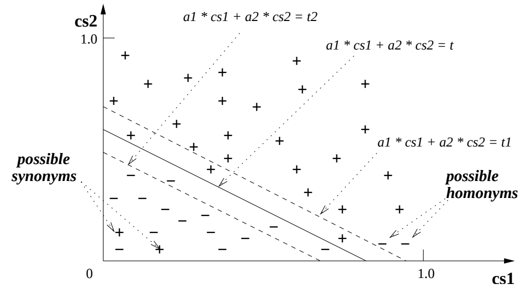


Figure 8: Thresholding function

Based on this observation, we propose an approach to determining a boundary for a good threshold. We first set the boundary to some reasonable range [a, b]. Then, we apply the following process on all interfaces. For each interface under consideration, we obtain, for each field in the interface, the *maximum* similarity of this field with other fields in all other interfaces. We arrange these similarities into a list by the descending order of their values. Next, starting from the first value in the list which is within the current boundary, and for each such value v, we examine if v is significantly lower, say by percent p, than the previous value in the list. If yes, we ask the user to determine if the pair of fields corresponding to v is matching. If the answer is yes, we lower the upper bound to v and continue on the list; if the answer is no, we increase the lower bound to v and stop further processing down the list. The result of the process is an updated boundary [a', b'].

If the obtained boundary is still too rough, a refinement process is carried out. For this process, we reconsider the lists of maximum similarities obtained above, one for each interface, and select the one with the largest number of values within [a', b']. Instead of asking the user to determine if all these values correspond to correct matches, we adopt a bisection-like strategy to reduce user interactions: given a list of values, the first question is on the middle, and depending on the answer, the remaining questions will be restricted to either the first or the second half of the list.

5.2 Resolving the Uncertainties

The analysis of our experimental results reveals that most of the errors occur are: (1) false positive 1:1 mappings due to homonyms; (2) false negative 1:1 mappings; and (3) false negative 1:m mappings. To reduce these errors, in this section, we propose several methods to determine the uncertainties arising in the mapping process and resolve them with user's interactions.

Determine Possible Homonyms: Homonyms are two words pronounced or spelled the same but with different meanings. Here, we use homonyms to refer to two fields which have very large linguistic similarity but rather small domain similarity. For example, type of job might mean the duration of the job such as part time and full time, but it might also mean the specialty of the job such as accountant, clerk, and lawyer. Intuitively, the domain of a field reflects its extensional semantics, while the label/name of a field conveys its intensional semantics. Two fields with highly similar names/labels but very different domains resemble

Domain	Leaf Nodes			Internal Nodes			Depth			% of Fields w/ Inst	Distribution of Simple Domain Types							Comp. Types	
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg		Int	Real	String	Money	Time	Area	CalMon	Date	Other
Airfare	5	15	10.7	1	7	5.1	2	5	3.6	71.9	76	0	26	0	20	0	26	6	9
Automobile	2	10	5.1	1	4	1.7	2	3	2.4	61.4	12	0	37	2	0	0	0	0	1
Book	2	10	5.4	1	2	1.3	2	3	2.3	25.4	0	0	24	0	0	0	0	4	18
Job	3	7	4.6	1	2	1.1	2	3	2.1	70.0	2	0	53	1	0	0	0	0	5
Real Estate	3	14	6.7	1	6	2.4	2	4	2.7	67.8	20	0	39	21	0	12	0	0	7

Table 1: Domain and characteristics of interfaces for our experiments

homonyms in linguistics.

Again, we use Figure 8 to illustrate homonym fields and their relationship with the thresholding function. Consider cs_1 as the linguistic similarity, and cs_2 as the domain similarity. Note that the lower right area of the figure is where potential homonyms could occur, since this is the area where the linguistic similarity is high but the domain similarity is low. Further note that homonym fields could have a similarity value large enough to be placed above the thresholding line.

To resolve homonym fields, user is asked to confirm when the system discovers two fields with rather high linguistic similarity but very low domain similarity. Since homonym fields could potentially confuse the process of learning the clustering threshold, they are resolved first before the learning starts. If two fields are determined to be homonyms, they will not be utilized during the process.

Determine Possible Synonyms: Possible synonyms refer to two fields which could still be semantically similar even if neither their linguistic similarity nor their domain similarity is very high. Possible reasons for the low similarity of two such fields are: (1) their labels/names do not have any common words; and (2) their domains are actually semantically similar but might not contain a sufficient number of common values so that their domain similarity will be large enough. Examples of such fields are those positive points located below the thresholding line in Figure 8.

To determine potential synonyms, an additional *Check-Ask-Merge* procedure is introduced right after the end of step 2 of the clustering process in Figure 4. The procedure is a repeated application of: (1) *check* if there are two clusters which contain fields with some common instances and if yes, we choose two such fields with the largest number of common instances; (2) *ask* the user if two chosen fields match; and (3) if they match, *merge* the corresponding two clusters.

Determine Possible 1:m Mappings: Although the procedure given in Section 4.3 is quite accurate (see our experimental results) in identifying 1:m mappings, there are some potential 1:m mappings which do not satisfy the rules which are designed to find only the 1:m mappings the system is highly confident of.

Intuitively, field e could potentially map to fields f and g if: (1) the similarity between e and f is very close to the similarity between e and g; (2) f and g are very close to each other in the interface; and (3) there is no other field in the interface containing e, which also satisfies conditions (1) and (2). To reduce the number of questions asked, we further require that f and g are adjacent to each other in the interface. Note that condition (3) is necessary since otherwise there might as well be multiple 1:1 mappings instead of one 1:m mapping.

We apply similar rules to find potential 1:m mappings with more than two fields on the many side. The resolution of uncertain 1:m mappings is carried out in the preliminary-1-m-matching phase but after all other automatic means of identifying 1:m mappings are completed.

6. EXPERIMENTS

To evaluate our approach, we have conducted extensive experiments over several domains of sources on the Web. Our goal was to evaluate the matching accuracy and the contribution of different components.

Data Set: We consider query interfaces to the sources on the "deep" Web in five domains: airfare, automobile, book, job, and real estate. For each domain, 20 query interfaces were collected by utilizing two online directories. First, we searched listed sources in invisibleweb.com (now profusion.com) which maintains a directory of hidden sources along with their query interfaces. We also utilized the Web directory maintained by yahoo.com. Since yahoo.com does not focus on listing hidden sources, for the sources in the domain of our interest, we examine if they are hidden sources and if yes, we identify their query interfaces. After query interfaces were collected, they were manually transformed into schema trees. Note that it is possible to utilize the techniques developed in [24] to facilitate the transformation.

Table 1 shows the characteristics of interfaces used in our experiments. For each domain, the table shows the minimum, the maximum, and the average of the number of leaf nodes and internal nodes, and of the depth of the schema trees representing the interfaces. For leaf nodes, the table also shows the percentage of their corresponding fields which contain instances. The last two portions of the table show the distribution of simple and composite domain types of the fields.

Performance Metrics: Similar to [10, 19], we measure the performance of field matching via three metrics: precision, recall, and F-measure [31]. Precision is the percentage of correct mappings over all mappings identified by the system, while recall is the percentage of correct mappings identified by the system over all mappings as given by domain experts. F-measure incorporates both precision and recall. We use the F-measure where precision P and recall R are equally weighted: $F = 2PR/(R + P)$. Note that a 1:m mapping is counted as m 1:1 mappings, each of which corresponds to one of the m mapping elements. For example, a 1:m mapping from element e_i to e_j and e_k are considered as two 1:1 mappings: $e_i \leftrightarrow e_j$ and $e_i \leftrightarrow e_k$.

Experiments: For each domain, we perform three sets of experiments. First, we measure the accuracy of our automatic field matching algorithm. Second, we examine the effectiveness of user interactions in improving the accuracy. Third, we evaluate the contribution of different components.

For all the experiments we conduct, the weight coefficients for the component similarities are set as follows: (1) $\lambda_{ls} = .6$

Domain	Prec.	Rec.	F
Airfare	92.0	90.7	91.4
Auto	92.8	92.3	92.6
Book	93.5	92.5	93.0
Job	81.8	83.5	82.6
Real Est.	81.0	96.7	88.1
Average	88.2	91.1	89.5

Table 2: The automatic field matching accuracy

Domain	Prec.	Rec.	F
Airfare	94.1	90.5	92.3
Auto	96.3	91.4	93.8
Book	97.8	92.5	95.1
Job	90.0	71.8	79.9
Real Est.	97.6	93.6	95.6
Average	95.2	88.0	91.3

Table 3: The accuracy with learned thresholds

Domain	Prec.	Rec.	F
Airfare	94.1	90.6	92.3
Auto	96.5	97.6	97.0
Book	98.5	97.1	97.8
Job	95.0	86.4	90.5
Real Est.	95.6	97.6	96.6
Average	96.0	94.0	94.8

Table 4: The accuracy with all user interactions

Domain	Thres.	Hom.	1:m	Syn.	Total
Airfare	4	0/0	1/2	0/0	7
Auto	2	0/1	3/1	0/0	7
Book	2	0/1	5/0	0/1	9
Job	5	3/3	0/3	3/14	31
Real Est.	5	0/2	4/4	2/0	17

Table 5: Distribution of different types of questions

and $\lambda_{ds} = .4$. This reflects the observation that both the description-level and the instance-level properties of a field are very important evidences in identifying the semantics of the field, and further that labels are typically more informative than instances; (2) $\lambda_n = 1/6$, $\lambda_l = 3/6$, and $\lambda_{nl} = 2/6$. This reflects the observation that the label of a field is more informative than the name of a field which often contains acronym and abbreviation; (3) For the domains (such as money, time, and area) whose types convey significant semantic information, we set $\lambda_t = .8$ and $\lambda_v = .2$. For other domains (such as int, real, and string), we set $\lambda_t = 0$ and $\lambda_v = 1$.

6.1 Automatic Field Matching Accuracy

For all the experiments with the automatic field matching algorithm, the clustering threshold is set to zero for all domains (so that as long as two fields have some non-zero similarities, they will be matched). Table 2 shows the accuracy of our automatic field matching algorithm. Columns 2–4 show precision, recall, and F-measure, respectively. We observe that precisions range from 81% to 93.5%, recalls from 83.5% to as high as 96.7% over five domains, and that it achieves about 90% in F-measure on average. These indicate the effectiveness of our automatic field matching algorithm.

6.2 Results on User Interactions

In the user interaction experiments, we want to observe whether the proposed methods for the interactive learning of thresholds and the resolution of uncertainties are effective. Table 3 shows the field matching accuracy with learned thresholds. That is, the only user interaction is to determine the threshold. Table 4 shows the accuracy of the interactive field matching algorithm which incorporates both the threshold learning and the resolution of uncertainties.

Table 5 shows, for each domain, the number of questions asked for each type of questions. The second column shows the number of questions asked to determine the threshold. The third column shows the number of questions asked to determine homonyms (x/y means that x questions were asked with a "yes" response while y questions were asked with a "no" response). For example, for the book domain, one homonym question was raised but the user responded with a "no" answer. The fourth column shows the number of 1:m questions asked. The fifth column shows the number of questions asked at the end of the clustering algorithm to determine potential 1:1 mappings. The last column shows the total number of questions asked.

Threshold learning: Compare Table 3 with Table 2, we observe that precisions increase significantly and consistently over all five domains while recalls are all around 90% except for the job domain. The reason for the lower recall for the job domain is that one of questions raised to the user during the threshold learning process happened to be homonym fields with relatively large similarity, driving up the threshold. This indicates the importance of detecting homonyms, particularly before the threshold learning process. Note that, in general, a larger threshold will lead to higher precision but lower recall. Thus, in order to improve over the automatic field matching algorithm, it is critical that the learned threshold does not lead to a dramatic decrease in the recall while improving the precision significantly. The above results indicate that our threshold learning process is very effective. Moreover, this is achieved with a small amount of user interaction: on average less than four questions were asked for each domain.

All user interactions: Compare Table 4 with Table 3, we observe that recalls improve consistently over all domains, with nearly 15% for the job domain. Detailed analysis on the results for the job domain reveals that the increase in recall is largely due to the resolution of homonyms. In fact, from Table 5 we observe that six homonyms questions were asked for the job domain and three of them were confirmed by the user. We also observe that the most common type of questions is 1:m mapping question and at least one proposed 1:m mappings were confirmed by the user for all domains except for the job domain. The resolution of potential 1:1 mappings is most effective in the real estate domain. Overall, the total number of questions asked ranges from 7 for the airfare and automobile domains, to 31 for the job domain.

The overall improvement due to the user interactions can be observed by contrasting Table 4 with Table 2. We note that the average precision increases by 7.8%, the average recall by 2.9%, and the average F-measure by 5.3%. These indicate the effectiveness of the user interactions.

6.3 Studies on Component Contribution

Table 6 shows the contribution of different components in the automatic field matching algorithm to the overall performance. We examine three important components: (1) handling of 1:m mappings; (2) utilization of instance information; and (3) tie resolution. For each component, we show the accuracy of the automatic field matching algorithm if the component is removed. To ease the comparison, we reproduce the results for the complete automatic field matching algorithm in the last three columns.

Domain	None			No 1:m Handling			No Instances			No Tie Res.			All		
	Prec	Rec	F	Prec	Rec	F	Prec	Rec	F	Prec	Rec	F	Prec	Rec	F
Airfare	81.0	66.9	73.3	93.0	81.8	87.0	82.2	83.4	82.8	84.9	87.4	86.1	92.0	90.7	91.4
Automobile	90.1	88.8	89.5	92.7	91.2	92.0	88.9	88.5	88.7	92.8	92.3	92.6	92.8	92.3	92.6
Book	97.7	86.8	91.9	93.5	92.0	92.8	97.7	87.2	92.1	93.5	92.5	93.0	93.5	92.5	93.0
Job	79.1	74.7	76.8	81.6	81.0	81.3	79.7	77.2	78.4	81.8	83.5	82.6	81.8	83.5	82.6
Real Estate	77.8	75.4	76.6	79.8	81.3	80.6	79.6	92.2	85.5	80.1	96.0	87.3	81.0	96.7	88.1
Average	85.1	78.5	81.6	88.1	85.5	86.7	85.6	85.7	85.5	86.6	90.3	88.3	88.2	91.1	89.5

Table 6: Comparisons of contribution of different components

Handling of 1:m mappings: Columns 5–7 show the results with the component of handling 1:m mappings removed. In comparison with the results for the complete algorithm, we observe that 1:m mapping of fields occur in all five domains. With the handling of 1:m mappings, the recall increases over all domains, with the largest increase as much as 15.4% in the real estate domain. The precision either increases or remains the same for all domains except for the airfare domain. The slight decrease in precision in the airfare domain is due to the relatively worse performance (83.8% in precision) of 1:m matching for the domain.

Utilization of instances: Since some solutions to interface matching do not utilize the instance information, we want to observe the effectiveness of exploiting instances for the field matching. Columns 8–10 show the results without utilizing the instances. We observe significant and consistent increases in recall over all domains if the instance information is utilized, with the largest increase 7.3% in the airfare domain. This confirms the importance of the instance information to the field matching and also our "bridging" effect observation.

Tie resolution: Our tie resolution strategy was actually first motivated by some example interfaces from the airfare domain. The results shown in columns 11–13 indicate that the strategy is indeed effective, achieving over 7% increase in precision and over 3% increase in recall. The improvement can also be observed in the real estate domain.

The aggregate contribution of these three components can be observed by contrasting columns 2–4 with the last three columns. As expected, we observe dramatic increases in recall over all five domains, ranging from 3.5% in the automobile domain to as much as 23.8% in the airfare domain. Overall, the average recall over five domains increases by 12.6%.

7. RELATED WORK

We discuss our work with the related works from the following perspectives.

Schema and Interface Matching: There is a large body of works on schema matching and integration [6, 7, 9, 10, 11, 14, 16, 18, 19, 23, 29]. [25] gives a taxonomy of approaches to schema matching. Most of the current works on schema matching only consider 1:1 mappings of elements [6, 7]. [18, 19] also handle 1:m mappings but the techniques utilized are completely different from ours. The average accuracy of matching reported in [19] (52%) is substantially worse than our accuracy rate although the test data are different. And since [18] only reports the comparison of their approach with several other systems on two example schemas, it is not clear how their system performs on a large data set. The importance of instances in schema matching has also been observed in [7, 11]. Our utilization of instances in suggesting possible mappings resembles the value correspondence problem in [20].

Two recent works [10, 11] also study interface matching. While 1:m mappings frequently occur among fields in the interfaces as we observed, both of them only consider 1:1 mappings of fields. Furthermore, both of them model interfaces as flat schemas and do not utilize the ordering, the sibling, and the hierarchical structure of the interfaces, which are highly valuable in improving the matching accuracy as we have shown.

User Interaction and Parameter Learning: Thresholding function can be regarded as a linear classification function. Learning of classifiers is extensively studied in the machine learning literature [21]. Interactive learning of classifiers is explored in [28] in the context of deduplication. Our approach to the interactive learning of thresholds is similar to [28] in the goal of reducing the number of user interactions, but our application area, namely schema matching, is different. Our experiments show that our approach can effectively shrink the confusion region with a small amount of user interaction.

In [7], users provide feedback on the system identified mappings. The feedback is captured as constraints which are utilized in future matching tasks. In our approach, users interact to resolve the uncertainties arising *during* the matching process.

Bridging Effect vs. Mapping Reusing: Reusing past identified mappings to help identify new mappings is an effective way of improving matching accuracy [25]. Consider three elements, a, b, and c. It might be difficult to match a with c directly, but if b has been previously matched with c and a is very similar to b, we could use the mapping of b and c to suggest the mapping of a and c. The bridging effect is similar to the idea of mapping reusing.

Our bridging effect observation is also motivated in part by the works done in [10, 12, 16], in particular by the holistic approach to interface matching in [10]. The bridging effect for 1:1 mappings has been discussed in Section 3. The bridging effect for 1:m mappings can be observed in Section 4.3.2, where 1:1 mappings obtained from the clustering process serve as the "bridges" for the identification of new 1:m mappings.

We further note that [32] exploits a unlabeled corpus to connect new examples with labeled examples in the context of text classification, to achieve a similar bridging effect.

8. CONCLUSIONS & FUTURE WORK

We have presented an approach to interface matching which achieves high accuracy across different domains. Our approach captures the hierarchical nature of interfaces, handles both simple and complex mappings of fields, and incorporates user interactions to learn the parameters and to resolve the uncertainties in the matching process. Both the description-level and the instance-level information of fields are utilized. Our results indicate that our approach is highly effective.

While our work is done in the context of interface matching, we believe our approach contributes to the general schema matching problem from several aspects. First, our bridging effect observation further shows that rather than matching two schemas at a time, we can exploit the evidences from a large set of schemas at once to help identify mappings. Second, our approach shows that user interactions can be introduced *during* the matching process, thus complementing the approaches where the user feedback is provided at the end of the matching process. Third, our approach shows that it is possible to utilize both the structural and the instance-level information of schemas to help identify complex mappings. Fourth, our approach for the active learning of parameters constitutes an important step towards a systematic tuning of the parameters in schema matching algorithms.

Although our approach has achieved a remarkable accuracy, there is still some room for improvement. Currently, we are investigating the possibility of user interactions in resolving other uncertainties in the matching process. One interesting interaction would be to help break ties when the ordering-based strategy fails. Another direction we are working on is to incorporate an automatic interface modeling procedure into our approach and to further evaluate our approach on the automatically generated schema trees.

Additional information on our work, including further experimental evaluation of the "bridging" effect, can be found on our project's Web site [1]. The data set used in our experiments is available from the UIUC Web integration repository [2] to facilitate related research.

Acknowledgments: We thank the anonymous reviewers for the invaluable comments. This work is supported in part by the following grants: IIS-0208574 and IIS-0208434 from the National Science Foundation, and ARO-2-5-30267 from the Army Research Office.

9. REFERENCES

[1] IceQ project: http://hanoi.cs.uiuc.edu/iceq/.

[2] http://metaquerier.cs.uiuc.edu/repository/.

[3] M. Bergman. The Deep Web: Surfacing the hidden value. *BrightPlanet.com*, 2000.

[4] W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM TOIS*, 18(3), 2000.

[5] L. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3), 1945.

[6] H. Do and E. Rahm. Coma - A system for flexible combination of schema matching approaches. In *VLDB*, 2002.

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

[8] C. Fellbaum, editor. *WordNet: An On-Line Lexical Database and Some of its Applications*. MIT Press, Cambridge, MA, 1998.

[9] A. Halevy and J. Madhavan. Corpus-based knowledge representation. In *Int'l Joint Conf. on AI*, 2003.

[10] B. He and K. Chang. Statistical schema matching across Web query interfaces. In *SIGMOD*, 2003.

[11] H. He, W. Meng, C. Yu, and Z. Wu. Wise-integrator: An automatic integrator of Web search interfaces for e-commerce. In *VLDB*, 2003.

[12] A. Hess and N. Kushmerick. Automatically attaching semantic metadata to Web services. In *Int'l Joint Conf. on AI - Workshop on Information Integration on the Web*, 2003.

[13] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[14] J. Larson, S. Navathe, and R. Elmasri. A theory of attributed equivalence in databases with application to schema integration. *IEEE Trans. on Software Engineering*, 15(4), 1989.

[15] S. Lawrence and C. Giles. Accessibility of information on the Web. *Nature*, 400, 1999.

[16] W. Li and C. Clifton. Semint: A tool for identifying attribute correspondence in heterogeneous databases using neural networks. *Data & Knowledge Engineering*, 33(1), 2000.

[17] L. Lovasz and M. Plummer. *Matching Theory*. North-Holland, Amsterdam, 1986.

[18] J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *VLDB*, 2001.

[19] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, 2002.

[20] R. Miller, L. Haas, and M. Hernandez. Schema mapping as query discovery. In *VLDB*, 2000.

[21] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[22] M. Porter. An algorithm for suffix stripping. *Program*, 14(3), 1980.

[23] R. Pottinger and P. Bernstein. Merging models based on given correspondences. In *VLDB*, 2003.

[24] S. Raghavan and H. Garcia-Molina. Crawling the hidden Web. In *VLDB*, 2001.

[25] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4), 2001.

[26] V. Raman and J. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, 2001.

[27] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McCraw-Hill, New York, 1983.

[28] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Int'l Conf. on Knowledge Discovery & Data Mining*, 2002.

[29] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3), 1990.

[30] S. Tejada, C. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information Systems*, 26(8), 2001.

[31] C. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[32] S. Zelikovitz and H. Hirsh. Improving short-text classification using unlabeled background knowledge to assess document similarity. In *Int'l Conf. on Machine Learning*, 2000.