Stop Word and Related Problems in Web Interface Integration

Eduard Dragut, Fang Fang, Prasad Sistla, Clement Yu
Computer Science Department
University of Illinois at Chicago
{edragut,ffang,sistla,yu}@cs.uic.edu

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

ABSTRACT

The goal of recent research projects on integrating Web databases has been to enable uniform access to the large amount of data behind query interfaces. Among the tasks addressed are: source discovery, query interface extraction, schema matching, etc. There are also a number of tasks that are commonly ignored or assumed to be apriori solved either manually or by some oracle. These tasks include (1) finding the set of stop words and (2) handling occurrences of "semantic enrichment words" within labels. These two subproblems have a direct impact on determining the synonymy and hyponymy relationships between labels. In (1), a word like "from" is a stop word in general but it is a content word in domains such as Airline and Real Estate. We formulate the *stop word problem*, prove its complexity and provide an approximation algorithm. In (2), we study the impact of words like AND and OR on establishing semantic relationships between labels (e.g. "departure date and time" is a hypernym of "departure date"). In addition, we develop a theoretical framework to differentiate synonymy relationship from hyponymy relationship among labels involving multiple words. We scrutinize its strength and limitations both analytically and experimentally. We use real data from the Web in our experiments. We analyze over 2300 labels of 220 user interfaces in 9 distinct domains.

1. INTRODUCTION

Enabling uniform access to the large amount of data behind query interfaces has been the goal of recent research projects on integrating Web databases. Among the tasks addressed so far in the literature are: source discovery, schema matching, consistent naming of attributes in integrated user interfaces etc. *The problem of consistent naming* is to assign labels to the attributes of an integrated query interface, represented as a schema tree, such that (1) no two sibling nodes (fields) have synonymous or hypernymous labels and (2) the label of a node must be a hypernym of the label of any descendant node [14]. Many techniques use the names and/or the labels of the attributes to infer that two attributes are synonyms/hypernyms. These techniques need to address issues like: removal of the so-called *stop words*, handling of "semantic enrichment words", defining a suitable similarity measure between the strings of the labels, etc. An imperfect solution to any of these problems may greatly affect the effectiveness of the matching result.

VLDB '09, August 24-28, 2009, Lyon, France

The Stop Word Problem

Consider the labels Where do you want to go? and When do you want to travel?. The words do, to, when, where and you are commonly regarded (e.g. according to [1]) as not conveying any significant semantics to the texts or phrases they appear in. Consequently, they are discarded. This kind of words are called *stop words*. The former label becomes want go and the latter label becomes want travel after the removal of the stop words. Given that go and travel are synonyms by WordNet[16], one infers that want go is semantically equivalent to want travel. Thus, one wrongly deduces that the original labels are semantically equivalent as well. The example illustrates that there are instances when stop words express important semantic information and their removal may lead to erroneous logic inferences. For instance, the removal of the words where and when along with the other stop words from the above labels yielded the wrong semantic relationship between the two labels.

The central problem addressed in this paper is finding the set of stop words in a given application domain. This problem is complicated because a stop word in one domain may not be a stop word in another domain. The example above illustrates that the word where is a content word in the Airline domain. The same word is a stop word in the Credit Card domain because its removal from the label Cell phone [where] we may call you does not affect the meaning of the label. A quick note: throughout the paper we refer to the data set used for the experimental study, the description of the data is postponed to Section 6.

To the best of our knowledge, there is no high quality mechanism to determine stop words for information integration problems. Nevertheless, we have noticed two strategies to handle stop words. First, before a task like schema matching in an application domain is undertaken, an *expert* manually analyzes all the labels or names of the schemas and compiles a set of stop words. This approach is time consuming, error prone and far from being trivial, especially for integration solutions that consider large sets of schemas at a time (e.g., integration of deep Web sources, Web directories

Figure 1: Example of a user interface.

or catalogs). For example, in our Credit Card domain, which has 20 forms, there are 818 labels (made of 2157 non-distinct words, out of which 373 are distinct) and it would take substantial time and effort to properly classify the words occurring in this domain into the two categories. Furthermore, since the number of application domains is very large and growing (a recent study estimated millions of such sources [25]), the manual operation is not feasible. Second, a precompiled generic list of stop words is considered—e.g., by searching such a list on the Web. Such a list is domain independent; unfortunately, a stop word in one domain is not necessarily a stop word in a different domain.

Information Retrieval has theoretical mechanisms to classify a set of words into stop words and content words. This is usually accomplished by defining a so-called *language model* [2]. A language model is a probability model that assigns probabilities/weights to words to reflect the degree of significance. The weight of a word is derived from the frequency of the word in the database of documents. Intuitively, the more documents having the word, the less useful the word is in distinguishing the documents having it from those not having it. Unfortunately, this strategy cannot be applied to the information integration problem. For example, in the Airline domain, the word what, which is a stop word, has a very low occurrence frequency, while the word passenger has a high occurrence frequency. Hence, this model would wrongly determine what to be a content word and passenger to be a stop word.

Semantic Enrichment Words

The labels of attributes may contain semantic enrichment words such as AND and OR. Pick-up date and time and From city or airport code are examples of such labels. The question is: what is the semantic relationship between Pick-up date and time and Pick-up date, or the relationship between Pick-up date and Date and time? A human being may argue that the semantics of Pick-up date and time subsumes the semantics of Pick-up date and the semantics of Date and time subsumes that of Pick-up date and time. But in order to make computer more intelligent, we need a systematic approach to handle such phrases. Words and phrases acquire meaning from the way they are used in society, blogs, internet web pages, etc. To understand the meaning and usage of these words, we manually investigated all the labels of the 220 interfaces containing them. We noticed a consistency of their use both lexically and topologically. The latter refers to their placements within query interfaces, i.e., whether these words occur consistently either in labels of fields or in labels denoting sections on Web query interfaces. For example, in the interface in Figure 1 OR appears only in the labels of the leaves (fields), e.g., the label From city or airport.

Evaluation

The problem of determining semantic relationships (synonymy and hypernymy) between the labels of the internal

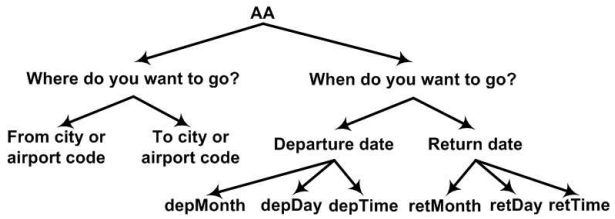

Figure 2: Example of a schema tree.

nodes of query interfaces is used to evaluate the impact of the problem of correctly identifying the stop words and the role of semantic enrichment words. Two algorithms are implemented: a naive algorithm and an improved one. Our experiments (see Section 6) show that without properly handling stop words, the accuracy (F-score) may decrease by almost 30% when the naive algorithm is applied (in domains such as Car Rental, Credit Card) and by almost 40% when the improved algorithm is used (e.g., Car Rental).

Usage

Automatically naming the attributes of an integrated query interface [14], as part of Web interface integration, requires that the semantic relationships (i.e. synonyms and hyponyms) among labels of attributes are determined. Furthermore, synonymy relationships should be distinguished from hyponymy relationships. For example, the labels of different fields of the integrated interface should not be synonyms (homonymy problem [28]). When the labels of fields consist of one word only it may be possible to check if they are synonyms in an electronic dictionary, such as WordNet. The problem is however increasingly harder when the labels are complex phrases. For example, no electronic dictionary would be able to say that Area of study is a synonym of Work area. Therefore, techniques that establish semantic relationships between multi-word phrases beyond those in electronic dictionary are of practical use. Another example is the integration of hierarchies (e.g., Web directories/categories). It is important to distinguish between synonymy and hyponymy as the former usually results in collapsing the synonym concepts into one concept, whereas the latter requires the insertion of the more specific concept as a descendant of the less specific one in the integrated hierarchy.

There are already techniques developed for computing *semantic similarities* between words and phrases [6, 7, 8, 9, 10, 20, 21, 24, 29]. Semantic similarity only suggests a likely relationship between phrases. It does not differentiate the synonymy relationship from the hyponymy relationship. It may even suggest "wrong" semantic relationships like finding that "true" and "false" have a similar semantics [9]. This is particularly the case of similarities computed from the number of hits returned by search engines.

The contributions of this paper are:

- A study of the stop word problem within the Web interface integration problem. We believe our study is the first to tackle this problem in the literature.
- A study of extending the definition of semantic relationships from one-word phrases to multi-word phrases. It also includes a comprehensive analysis of the role of the semantic enrichment words on determining semantic relationships between multi-word phrases.
- An extensive experimental study of our algorithms. The

average precision, recall and F-score of our algorithm for computing the stop words are 92.4%, 99.5% and 95.6%, respectively. We present a naive algorithm for computing semantic relationships between labels of internal nodes. This algorithm achieves an average precision of 76.1%, an average recall of 75.4%, and an average F-score of 74.9%. Our contribution is an algorithm that improves the average precision to 95%, the average recall to 90.4% and the average F-score to 92.6%.

The rest of the paper is organized as follows. Section 2 briefly introduces the problem of integrating Web query interfaces. Section 3 covers the stop word problem. Section 4 discusses the issues of computing semantic relationships between multi-word labels. Section 5 contrasts two algorithms for computing semantic relationships. We describe the experimental study in Section 6. Our work is contrasted with others in Section 7. The paper concludes in Section 8.

2. BACKGROUND

The most advocated approach to integrate Deep Web sources is to undergo integration *domain-wise*. First, relevant sources are discovered [3]. Second, query interfaces are identified, extracted from the relevant Web pages [12] and clustered on subject domains [4]. Third, fields of different query interfaces in the same domain are matched [19, 30, 31]. This is the well-known schema matching problem. Fourth, different user interfaces are merged to form a unified interface [13]. Fifth, the unified interface needs to be consistently labeled [14]. Sixth, a user's query on the unified interface is translated to queries against interfaces of specific sources [32]. Last, returned data by individual sources needs to be correctly extracted and the results ranked in descending order of desirability (e.g., price). The problems addressed in this paper are relevant to the schema matching (step 3), merging (step 4) and naming (step 5) problems.

Representation of Query Interfaces

Data in searchable databases are accessible through form-based search interfaces (mostly HTML forms). The basic building blocks for these forms are: text input boxes, selection lists, radio buttons, and check boxes. We will generically call them *fields*. The structure of a query interface can be captured using a *schema tree* [31]. It is an ordered tree of elements so that leaves correspond to the fields in the interface, internal nodes correspond to groups or super-groups of fields in the interface, and the order among the sibling nodes within the tree resembles the order of fields in the interface (usually this is top-down and left-to-right). Figure 1 shows a typical example of a query interface in the airline domain and its corresponding hierarchical representation is depicted in Figure 2.

3. COMPUTING THE SET OF STOP WORDS

For a set of Web query interfaces in a given application domain, the problem to be addressed is to determine the stop words within the labels of the attributes (fields and internal nodes) of the interfaces.

Our solution to the stop word problem is based on the heuristic observation that the set of stop words from an information integration perspective is a *subset* of the set of stop words from an information retrieval perspective. For example, the word last is in general a stop word [1], but not a stop word in the label Last name. We propose to tackle the stop word problem as follows: take an arbitrary general purpose dictionary of stop words and find its maximal subset that satisfies constraints specific to the information integration problem. In our experiments we performed a search on Google with the query "stop words" and picked a link [1] that pointed to such a list. We consider three constraints, called *stop word constraints*. After the removal of incorrect stop words, the following situations arise:

1. **Empty label:** A non-empty label becomes empty after the removal. The empty label cannot be used to convey necessary information to users. Also, it cannot be used to derive any knowledge.
2. **Homonymy:** Two sibling nodes in a hierarchy have synonym labels (the well-known homonymy problem [28]).
3. **Hyponymy:** Two sibling nodes in a hierarchy have a hyponymy relationship between them.

Before these constraints are illustrated we draw attention to a possible logic trap. The stop words have to be removed from labels before semantic relationships could be established between labels. But, in order to remove the stop words a definition for these relationships is necessary. The problem seems circular. To avoid this issue, *we regard the labels as sets of words*. A label A is a hyponym of label B if the set of words of B is strictly contained in the set of words of A. A label A is a synonym of label B if they have identical sets of words. The intuition of these rather simple definitions is that the labels of sibling nodes within a query interface share many of their words and the designer uses some key distinct words to emphasize the semantic meanings of the sibling fields. Observe that the sibling leaves of the internal node Where do you want to go? (Figure 2) have the same set of words except for the words from and to, which we actually aim to find.

Each of the stop word constraints is now motivated in turn. Consider the field with label From in the interface in Figure 3. The word from is a stop word according to most stop word dictionaries. However, by removing it from the label, we are left with an empty label which is practically useless for any knowledge inference. Thus, an important indicator of a word not being a stop word is when the removal of the word leads to an empty label. The homonymy condition is illustrated by the example of the labels From city or airport code and To city or airport code in Figure 1. By discarding both From and To, the two labels become identical, which is an obvious undesirable consequence. We use the labels of the interface in Figure 3, Where do you travel? and When do you travel?, to motivate the third criterion. The obvious stop words are do and you. Their removal does not impact the logic inferences between the two labels. However, the removal of where from the label [Where do you] travel? makes this label a hypernym of the label When [do you] travel? (because $\{\text{travel}\} \subset \{\text{when}, \text{travel}\}$).

It is erroneous to enforce the homonymy and hyponymy constraints over an entire schema tree, because the same label may be used to denote distinct concepts in different branches of the tree. As a case in point, the label Address is used to denote both home address and company address in interfaces in the Credit Card domain. Since the label appears in two distinct branches of the tree (which branches, corre-

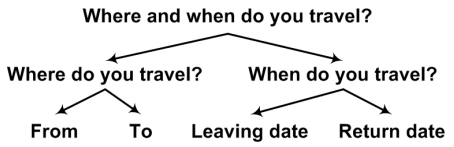

Figure 3: Another example of schema tree.

spond to different sections of the user interface), one branch describes the home information and the other branch describes employment information, there is no danger of misinterpreting the meaning of the label.

However desirable these constrains are, the stop word problem is intractable. Worse, we will show that regardless of the subset of constraints chosen the problem remains "equally" hard.

Here we show that the **stop word problem** is NP-complete. We define a label to be a set of words. The labels of a set of sibling nodes in a schema tree are called a *set of sibling labels*, denoted by SL. SL also includes the label of the root, if exists. Let $\mathcal{S}$ be the set of all sets of sibling labels from all the schema trees in an application domain. For each set of sibling labels $SL \in \mathcal{S}$, we need to check the stop-word constraints. Thus the stop-word problem is defined as follows. Given a set $\mathcal{S}$ of sets of sibling labels and a set T of words, called *potential stop words*, find the largest subset T' of T such that by deleting all words of T' from every label in SL, $SL \in \mathcal{S}$, the resulting multi set $\mathcal{S}'$ satisfies the following property: no label in SL', $SL' \in \mathcal{S}'$, is a super set of another label in it. Note that each set of sibling labels SL' satisfying the above property also satisfies the following conditions: the empty set does not belong to SL' (i.e., empty label condition) and no two members of it are identical (i.e., homonymy condition).

EXAMPLE 1. *We use only one schema tree, namely the one in Figure 3, to exemplify the stop word problem. The set of labels* {*Where do you travel?, When do you travel?*} *is a set of sibling labels. The set of all sibling labels induced by this schema tree is* $\mathcal{S}$ = { {*From, To*}, {*Where do you travel?, When do you travel?*}, {*Leaving date, Return date*}, {*Where and when do you travel?*}}. *Suppose the set of potential stop words is* T = {*do, from, to, when, where, you*}. *The words from and to are not stop words because their removal leaves some labels empty. The removal of where from the label [Where] do you travel? makes the label a hypernym of its sibling label When do you travel?, because the former is a proper subset of the latter. The word when behaves similarly to the word where. Hence, the largest subset* T' *of* T *satisfying the stop word constraints is* {*do, you*}.

THEOREM 1. *The* ***Stop word problem*** *is NP-complete. Sketch of Proof: Reduction from the set covering problem*

It turns out that even if a single schema tree and its set of sibling nodes are considered, the stop word problem is NP-complete. Or, in the above formulation, the problem remains "hard" even when the cardinality of $\mathcal{S}$ is 1.

A first reaction to this result is to try to solve the stop word problem by eliminating one or more of the conditions stated above. For instance, we could attempt to solve the problem by keeping the "empty label" and "homonymy" constraints, and by dropping the "hyponymy" constraint. Unfortunately, this problem is also NP-complete. The proof

StopwordsAlgorithm
Input: an arbitrary stop word dictionary T,
the set of schema trees $\mathcal{QI}$
Output: a maximal set of stop words T'
1. $T' \leftarrow \emptyset$
2. W = the set of all words in the domain
3. $D = T \cap W$
4. **pick** word $w \in D$
5. **for each** interface $q \in \mathcal{QI}$
6. remove w from the labels of interface q
7. check the stop word constraints for the labels of sibling nodes
8. **if** no stop word constraint is violated **then** $T' \leftarrow T' \cup \{w\}$;
else remove antonyms of w appearing in D from T'
9. **goto 4**
10. **return** T';

Figure 4: Stop word algorithm

is by reduction from the *set covering problem*. Another attempt would be to keep only the "empty label" constraint. This version of the stop word problem is also NP-complete. The proof is by reduction from the *independent set problem*. By now it should be pretty clear that the problem is hard regardless of the combination of constraints chosen. The common practice to handle NP-complete problems is to devise approximation solutions that come as close as possible to the ideal solution. Next we present an approximation algorithm to the stop word problem.

The approximation solution relies on the following observations. First, we assume that the source query interfaces satisfy the following constraint: Source interfaces do not have sibling nodes whose labels are either homonymys or hyponyms/hypernyms. The attributes whose labels are initially empty are ignored. Second, with this assumption, it is trivial to show that the empty set of stop words obeys the stop word constraints. Since this set satisfies the desired properties, it becomes the initial list of stop words in the given application domain. Third, the main idea is to traverse a given general purpose dictionary of stop words and for each word in the dictionary to check if the stop word constraints are satisfied. If they are satisfied, the word is a stop word in the given application domain. These steps actually characterize the algorithm given in Figure 4.

PROPOSITION 1. *The algorithm* StopwordsAlgorithm *returns a maximal set of stop words with respect to the stop word constraints.*

Two points need to be clarified about the algorithm. First, the generic list of stop words is intersected with the set of words used in the labels. We search for the stop words only in this new set. Second, we observe that some stop words come as antonym pairs (e.g., "last", "first") in general purpose stop word dictionaries [1]. Hence, we also employ the rule that if a word is not a stop word then its antonym is also not a stop word, to derive more content words (in step 8). The assumption is that for each potential stop word, if it is determined to be a non-stop word, then its antonyms, if they exist in the potential list of stop words, will also be non-stop words.

4. SEMANTIC RELATIONSHIPS AMONG LABELS

The main objects manipulated in the integration of Web query interfaces are the labels of the source interfaces. In order to establish semantic relationships (e.g., hypernymy) between labels across user interfaces we represent them as sets of content words (we discard the stop words). For example, {provide, financial, information} corresponds to Please provide us with some financial information. The semantic relationships we focus on are: synonymy and hypernymy. They are computed based on the relationships among the tokens of the labels as given by WordNet.

DEFINITION 1. *Given two labels A and B along with their set of content words representation $A_{cw} = \{a_1, ..., a_n\}$ and $B_{cw} = \{b_1, ..., b_m\}$, respectively, $m, n \geq 1$, we define the following relationships between A and B:*

- *A synonym B, if there exists a bijective function $f: A_{cw} \rightarrow B_{cw}$ with $f(a) = b$ if a is either the same as or synonym with b, where synonymy between a and b is given by WordNet. (E.g. Area of Study synonym Field of Work since area is a synonym of field and study is a synonym of work, by WordNet.)*
- *A hypernym B, if there exists an injective function $f: A_{cw} \rightarrow B_{cw}$ with $f(a) = b$ if a rel b, where rel is either identity, synonymy or hypernymy, by WordNet. (E.g. Financial Information hypernym Household Financial Information.) If $n = m$ then at least one rel is hypernymy by WordNet. (E.g. Employment Information hypernym Job Information, because Employment is a hypernym of Job.)*
- *A hyponym B if B hypernym A*

The intuition behind this definition is that a label B is hyponym of label A (or, the meaning of B is subsumed by that of A) if the set of content words of A is "contained" in the set of content words of B (containment means that the elements of A can be mapped into B using a 1-to-1 function). The reason is that additional words usually restrict the meaning of a phrase.

The synonymy and hyponymy relationships are defined in terms of bijective and injective functions between sets of labels (Definition 1). In general, between two sets A and B, with A and B having n and m elements ($n \leq m$), respectively, there can be $P(m, n)$ injective functions (P stands for permutation) and $m!$ bijective functions, if $n = m$. One way to test for hyponymy is to generate all combinations of mapping the elements of A into those of B and test if there exists one mapping that satisfies Definition 1. This approach takes exponential time. In the rest of the section, we show that the problem of computing the semantic relationships according to Definition 1 can be reduced to bipartite matching problems with well-known polynomial algorithms.

4.1 The Senses of a Word in a Domain

In order to acquire a better understanding of the meaning of the content words with respect to a dictionary (WordNet), we constructed inverted lists of content words for each of the 9 domains on which we perform experiments. Associated with each content word, we created an inverted list of labels containing the word in a domain. Table 1 shows an example of such lists. In total, the inverted lists have 735 content words and 2,319 labels. Note that the same word may appear in multiple lists as it may be used across domains. For example, Table 1 shows the word Type in the Real Estate and Credit Card domains along with some of the labels it appears in. For each entry in the list we manually check the number of meanings the word is used within a given application domain. For example, the usage of the word Address in the Credit Card domain shown in Table 1 is consistent with only one sense in WordNet: "the place where a person or organization can be found or communicated with". Among the 735 words, we found only one content word that has multiple senses in the same domain. This is the word Area and it is shown in the table. The word has two senses in the Real Estate domain. Its meaning in the label Select an area is "a particular geographical region of indefinite boundary". Whereas its meaning in the label minimum floor area is "the extent of a 2-dimensional surface enclosed within a boundary". Thus we conclude that in a given application domain the occurrence of words with multiple meanings is extremely rare and we state this in the following assumption:

Table 1: Example of inverted list of content words

Domain	Word	Labels
Credit Card	Address	Home address, Company address, Email address
Credit Card	Type	3rd party credit card type, Major credit card type
Real Estate	Type	Property type, Parcel type, Type of use
Real Estate	Area	Select an area, Minimum floor area

ASSUMPTION 1. *We assume that each content word used across a set of query interfaces in a given domain of discourse (e.g. airline, car rental) has a unique sense.*

The assumption is also supported by real life facts. First, it is very unlikely for a well-designed query interface to have the same content word used to denote different concepts within an application domain. For one, this may have the risk to confuse the user. Second, it has been observed in practice [19] that the majority of deep Web sources in the same domain are consistent on some "core" content words (e.g., title) and others are variations of the core ones (e.g., title of book). Thus, even though title has 12 senses by the electronic dictionary WordNet, its use across the interfaces in the Book domain is consistent with only one of these senses (i.e., "the name of a work of art on literary composition"). The assumption helps us to build a formal framework that facilitates a systematic way to reason about the labels. Nonetheless, it is not trivial in practice to determine the actual sense of a word (or phrase) in a given context. Our finding is consistent with the observation made in [8] that domain specific text tends to greatly constrain which senses of a word will appear.

PROPOSITION 2. *The relations introduced above have the following properties:*

1. *Hypernymy is transitive, asymmetric and irreflexive.*
2. *Synonymy is an equivalence relation on labels.*

The unique sense assumption is key in proving the above statements. The result of the proposition is later used to establish the relationship between the bipartite matching problems and the problem of computing the semantic relations. The reduction of the latter problem to the former problem is discussed next.

4.2 Conversion to Bipartite Matching

We first introduce several graph theory definitions. A bipartite graph $G = (V = (A, B), E)$ is a graph whose vertices

can be divided into two disjoint sets A and B such that every edge in E connects a vertex in A to one in B. A *matching* M in G is a set of edges with no two edges sharing a common vertex. A *maximum bipartite matching* is a matching that contains the largest possible number of edges. A *perfect matching* is a matching which covers all vertices of the graph. That is, every vertex of the graph is incident to exactly one edge of the matching. A *maximum weighted bipartite matching* in a weighted bipartite graph is defined as a perfect matching where the sum of the values of the edges in the matching has the maximum value.

Let A and B be two labels. Denote their sets of content words by $A_{cw} = \{a_1, ..., a_n\}$ and $B_{cw} = \{b_1, ..., b_m\}$, respectively, $m, n \geq 1$. A bipartite graph $G = (V, E)$ is constructed as follows: the set of vertices $V = A_{cw} \cup B_{cw}$ and an edge $e = (a_i, b_j) \in E$, $1 \leq i \leq n$, $1 \leq j \leq m$, if a_i is either equal to, a synonym of or a hypernym of b_j by WordNet. According to Definition 1, label A is a synonym of label B if there exists a bijective function from A_{cw} to B_{cw}. Since the bijective function corresponds to a perfect matching in the associated bipartite graph, then the problem of finding a synonymy relationship between two labels becomes the problem of finding a *perfect matching* in the associated bipartite graph that consists *only* of synonym edges. In Figure 5, we give a simple example of a perfect matching corresponding to a synonym relation between labels Area of study and Field of work.

Finding hypernymy relationships is somewhat more complicated because two cases have to be considered: 1) when the number of content words in one set is smaller than the number of content words in the other set and 2) when the numbers of content words of the two sets are equal. **Case 1:** suppose $|A_{cw}| < |B_{cw}|$, then label A is a hypernym of label B if there exists an injective function from A_{cw} to B_{cw}. In the bipartite graph formulation, the problem of finding a hypernymy relationship is reduced to finding a *maximum bipartite matching* that consists of synonym or hypernym edges and the cardinality of the matching is equal to $|A_{cw}|$. **Case 2:** $|A_{cw}| = |B_{cw}|$, since a bijective function needs to be determined, this case is similar to finding a synonymy relationship, but in this case at least one relation must be a hypernymy relationship. Hence, we have to solve the problem of finding a *perfect matching* in the associated bipartite graph that consists of synonym edges and at least one hypernym edge.

When $|A_{cw}| = |B_{cw}|$, we need a way to distinguish between a matching corresponding to a synonymy relationship and a matching corresponding to a hypernymy relationship. The problem boils down to determining if at least one hypernym edge is present in the matching. It is inefficient to generate all possible matchings and test if the above conditions are met. This instead can be realized by assigning different weights to synonym and hypernym edges. A weight of 1 is assigned to a synonym edge and a weight of 2 is assigned to a hypernym edge. Then the problem of capturing the semantic relationships between the labels becomes a *maximum bipartite weighted matching* problem (recall that maximum bipartite weighted matching is also a perfect matching). The trick is to observe that a maximum weighted bipartite matching for a synonymy relationship has the sum of the values of its edges equal to $|A_{cw}|$. Whereas the maximum weighted matching for a hypernymy relationship has the sum strictly larger than $|A_{cw}|$. The next

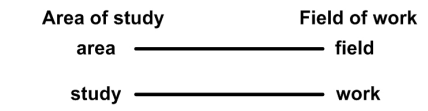

Figure 5: Example of bipartite matching.

proposition summarizes this in a concise formal manner.

PROPOSITION 3. *Let A and B be two labels with their set of content words representation $A_{cw} = \{a_1, ..., a_n\}$ and $B_{cw} = \{b_1, ..., b_m\}$, respectively, $m, n \geq 1$. Let $G = (A_{cw} \cup B_{cw}, E)$ be the bipartite graph associated with the two labels.*

1. *A is a synonym of B iff $m = n$ and the sum of the values of the edges in the maximum weighted bipartite matching is equal to m.*
2. *A is a hypernym of B iff*
   (a) *either $n = m$ and the sum of the values of the edges in the maximum weighted bipartite matching is larger than n;*
   (b) *or $n < m$ and the cardinality of the maximum bipartite matching is equal to n.*

The maximum bipartite matching problem can be solved in polynomial time. We implemented the Edmonds-Karp algorithm [15]. The maximum bipartite weighted matching problem has also a polynomial solution and we implemented the Hungarian algorithm [27]. It can be proven that it is impossible to have both synonymy and hypernymy relationships between two labels.

5. DICTIONARY VERSUS CONTEXT

We argue that, even though a content word may have unique sense in a dictionary, the task of establishing relationships between the labels of schema trees cannot be accurately carried out unless additional features are considered. To illustrate this point, we first present an algorithm that relates the labels of internal nodes across schema trees employing only Definition 1 and ignoring any other structural information. Conversely, we devise another algorithm that combines the structural information of schema trees and Definition 1. In this section, an analytical argument is laid out to explain why the latter is better than the former. The experimental section shows the dramatic accuracy improvement of the latter algorithm over the former algorithm.

5.1 The Naive Algorithm

The naive algorithm takes each label of an internal node of a given schema tree and compares it, using Definition 1, with all the labels of the internal nodes in the other schema trees. This is a trivial solution to semantically relate the labels of internal nodes across schema trees in an application domain. This algorithm achieves on average a precision of 76.1%, a recall of 75.4%, and a F-score of 74.9% over the data set employed in our experiments (see Section 6).

5.2 The Context-based Algorithm

The naive approach has several problems. We only discuss its main drawback. The same linguistic expression may be used to denote several unrelated concepts. Based on the *context* the label Address occurs within a schema tree, in the Credit Card domain, it denotes home address, company address, a relative's address, previous address, etc. The naive

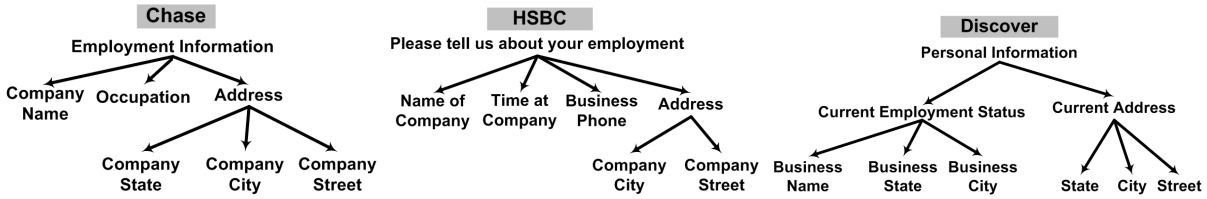

Figure 6: Example of user interfaces.

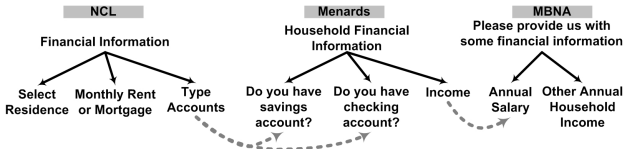

Figure 7: An instance of recursiveness.

technique does not take into account the context of a label and it will wrongly suggest that Address of interfaces Chase and HSBC is a *hypernym* of Current Address of Discover (Figure 6).

To avoid this problem a label should be regarded as a tuple $\langle l, C_l \rangle$, where l is the actual label and C_l is the context of the label in a given schema tree. The context of the label l within a schema tree is the set of descendant leaves of the internal node with the label l. For example, $\langle$Address, {Company State, Company City, Company Street}$\rangle$, $\langle$Address, {Company City, Company Street}$\rangle$ and $\langle$Current Address, {State, City, Street}$\rangle$ are the tuple representations of the three occurrences of Address which appear in the interfaces Chase, HSBC and Discover, respectively. Two labels are compared using Definition 1 only if the intersection of their sets of descendant leaves is nonempty. A leaf in a user interface is the "same" as that in another user interface if it has been determined that they are equivalent in the attribute matching phase. In our example, the labels Address of Chase and Address of HSBC are compared with each other because the intersection of their contexts is nonempty. But neither of them is compared with the label Current Address of Discover because their contexts do not overlap with the context of Current Address.

It should be noted that to determine the intersection of the context of an internal node v in a schema tree with that of an internal node w in another schema tree we may need to involve the ancestors of v and w. For instance, the interface Discover has a leaf State and the interface Chase (Figure 6) has a leaf Company State. Algorithms such as [11, 26] can determine that State and Company State do not match by utilizing the labels of their ancestors.

We employ the example in Figure 7 to reveal another subtlety of computing semantic relationships using the contexts of labels. The figure shows the internal nodes in three distinct user interfaces that represent financial information. The dashed lines show the semantically equivalent fields in the three interfaces. Suppose the order in which the labels of the three interfaces are compared is: NCL, MBNA and Menards. First, no semantic relationship is established between the labels Financial Information and Please provide us with financial information because their sets of descendant leaves do not intersect. Second, Financial Information is compared to Household Financial Information because the intersection of their sets of descendant leaves is nonempty. Third, even though the intersection of the descendant leaves

AlgorithmWithContexts
Input: the schema trees, $\mathcal{QI}$
Output: the list of semantic relationships $\mathcal{L}$
1. **do**
2. **for** i=1 **to** $|\mathcal{QI}| - 1$
3. **for each** internal node of U_i with label l
4. **for** j=i+1 **to** $|\mathcal{QI}|$
5. **for each** internal node of U_j with label p
6. **if** $C_l \cap C_p \neq \emptyset$ **then**
//C_l = set of descendent leaves of the node with label l
7. **if** l synonym p **then**
add l synonym p to $\mathcal{L}$;
$C_l, C_p \leftarrow C_l \cup C_p$; //update contexts
8. **else if** l hypernym p **then**
add l hypernym p to $\mathcal{L}$;
$C_l \leftarrow C_l \cup C_p$; //update context
9. **else if** p hypernym l **then**
add p hypernym l to $\mathcal{L}$;
$C_p \leftarrow C_l \cup C_p$; //update context
10. **while** no label's context changes
11. **return** $\mathcal{L}$

Figure 8: The algorithm using context of labels

of Household Financial Information and Please provide us with financial information is nonempty no semantic relationship between them can be determined using Definition 1. The set representations of the two labels are {household, financial, information} and {provide, financial, information}, respectively, and no relationship can be established between them because provide and household cannot be semantically related using WordNet. At the end of this iteration, no relationship has been established between Financial Information and Please provide us with some financial information. This example points out two issues. First, whenever a semantic relationship is established between two labels their contexts have to be updated. For example, after interfaces NCL and Menards are compared, the context of Financial Information should include the leaf Income. Second, multiple iterations may be needed in order to compute all semantic relationships across the labels of internal nodes of all the schema trees. In our example, a second iteration is needed to find that the context of Financial Information intersects that of Please provide us with some financial information, and therefore to find that the former is a hypernym of the latter.

The algorithm for computing semantic relationships between the labels of the internal nodes of query interface considering the context of labels is sketched in Figure 8. As illustrated in the example in Figure 7, the algorithm needs to iterate over the set of query interfaces until the context of each label does not change (Lines 1 & 10). Two labels from two distinct schema trees (Lines 2-5) are compared using Definition 1 if their contexts intersect (Line 6). If they

Table 2: Occurrences of AND and OR

	and	or	/
section title (internal node)	21	2	11
field (leaf)	2	47	52

are synonyms then their contexts are updated (Line 7) and this relationship is appended to $\mathcal{L}$, where $\mathcal{L}$ is the list of all semantic relationships found by the algorithm. If one of the labels is a hypernym of the other then only the context of the former label is updated (Lines 8, 9). Their relationship is added to $\mathcal{L}$ as well.

5.3 Handling AND and OR

6% of the labels in our data set have either AND or OR. We call these words *semantic enrichment words*. Electronic dictionaries have no entries for these words. Their proper handling directly impacts the ability to discover additional relations. It is desirable to understand if there is any semantic relation between the labels Departure date and time and Leaving date. Applying directly Definition 1 would report that the latter label is a hypernym of the former label. This is a wrong relation. A correct understanding of the way these words are used within Web query interfaces would increase the chances to find semantic relationships between even more complicated phrases. There are three main issues. First, how to compare two phrases one with and the other without semantic enrichment words. Second, how to compare two phrases that both have the same semantic enrichment word (e.g., City or airport code and City, point of interest or airport code). Third, should two labels, one containing AND and the other containing OR, be compared? And, if yes, what should the expected outcome be of such a comparison?

Table 2 shows the distribution of these words among the labels of leaves and internal nodes in our data set. There are 21 occurrences of AND and only 2 occurrences of OR among the labels of internal nodes. AND appears only 10 times among the labels assigned to leaves. Out of these, 8 appear in the comments attached to the actual label (e.g., Seniors (age 65 and over)) and do not have a direct bearing on the semantics of the actual label. Thus, we infer that AND is frequently used (91.3%) in the labels of internal nodes and not in the labels of leaves. When a label of an internal node (section) combines two concepts with AND it conveys to users that the section of the interface has fields pertaining to the meaning of the two concepts. In the section delimited by the label Where and when do you travel?, in the interface shown in Figure 3, there are fields pertaining to where and fields pertaining to when. In this sense, the semantic of AND is that of a union. In general, a label having n concepts joined with AND conveys the meaning of the union of the n concepts. An example of a label of a field containing AND is: Search one day before and after. The meaning of AND is still that of union.

OR occurs 47 times in the labels of leaves and only twice in the labels of internal nodes. Hence, it is frequently used (96%) among the labels of fields (leaves) and not among those of internal nodes. The label From city or airport code is a label of a field in the interface in Figure 1. Whenever OR is used in the label of a field, it expresses that the field plays multiple semantic roles. In our example, the field accepts both values denoting cities as well as airport code values.

AndOrAlgorithm
Input: label A and label B
Output: the semantic relation between them
EW = the set of enrichment words
1. **if** $EW \cap A = \emptyset$ and $EW \cap B = \emptyset$ **then**
   return Definition 1(A, B);
2. **if** $EW \cap A \neq \emptyset$ and $EW \cap B \neq \emptyset$ **then**
   remove EW from A and B;
   return Definition 1(A, B);
3. **if** $EW \cap A = \emptyset$ and $EW \cap B \neq \emptyset$ **then**
   $B' = B - EW$;
   if A synonym B' **then return** A synonym B;
   if B' hypernym A **then return** B hypernym A;
   if mapSynHypo(A, B') **then return** A hyponym B;
4. **if** $EW \cap A \neq \emptyset$ and $EW \cap B = \emptyset$ **then**
   Same as Case 3, but the roles of A and B are reversed.
5. **else return** no relationship;

Figure 9: Algorithm for computing semantic relationships

So, OR has the meaning of union, but with the following caveat: the field can only accept one type of input at a time (either city or airport code, but not both). When OR appears in the label of an internal node it has the same behavior. If the address above is a P.O. Box, a residential or business address is required is an example of such a label in the Credit Card domain. The user is required to fill in either the information about his/her address or some business address, but not both.

We also analyze the meanings of the character "/" within labels. In summary, this character has the same meaning as that of AND whenever it appears in the labels of internal nodes (e.g., Departure date/time) and it behaves the same as OR when showing up in the labels of leaves (e.g., Unit/Apt #). One last observation: No label has both AND and OR.

The question left to be answered is: should two labels, one containing AND and the other containing OR, be compared? First, each comparison is done between the label of one internal node and that of another internal node. Second, the number of internal nodes involving OR is negligible. Third, the semantics of AND and that of OR are so close that they can be considered to be equivalent. Thus, our analysis is based on the assumption that AND is the only enrichment word, if exists. Armed with these observations we extend Definition 1 to handle comparisons between labels with and without semantic enrichment words.

The general procedure to establish semantic relationships between phrases is given in Figure 9. In case 1, if labels A and B do not contain an enrichment word, Definition 1 is applied. If both labels A and B contain enrichment words, then remove these words from the labels and apply Definition 1 (Case 2)—e.g., Departure date and time is a synonym of Leaving date and time. If label A does not have an enrichment word and label B has an enrichment word, then the enrichment word is removed from B yielding B' (Case 3). There are three subcases based on the relationship between A and B':

- if A is a synonym of B' ($|A| = |B'|$), then the original labels are synonyms (e.g., City, state, country and City, state, and country). The intuition is that the two labels have the "same" set of words, but one of them has an

explicit enrichment word and the other does not.

- if B' is a hypernym of A ($|A| \geq |B'|$), then B is a hypernym of A. This result can be argued using transitivity. That is, since B' is a hyponym of B (because B' is the same as B, except for the enrichment words) and A is a hyponym of B' then, by transitivity, A is a hyponym of B. E.g., B = Address and housing information is a hypernym of A = Residence address information because housing is a hypernym of residence and the remaining content words in the two labels are the same.
- if the words of A can be mapped into those of B' ($|A| \leq |B'|$) using either synonymy or hyponymy relationship and at least one of them is a hyponymy relationship (the procedure mapSynHypo does the mapping) then A is a hyponym of B. Suppose A = Pick-up date and B = Pick-up date and time. Since the concepts of B are at least as general as those of A and B has also an enrichment word (which means union), then B is a hypernym of A. In other words, the intuition that additional words restrict the meaning of a phrase is no longer true when an enrichment word is present. Such words do not restrict, but rather enrich the meaning of a phrase. Note that this is contradictory to Definition 1.

This concludes case 3. Case 4 is the same as case 3 except that the roles of A and B are reversed. If none of these conditions is met then no relationship is established between the labels.

The new algorithm of computing semantic relationships that uses the context and the observations made about AND and OR attain on average a precision of 97.3%, a recall of 81.2% and an F-score of 88.3%.

5.4 Context Characterization of Labels

Recall that a label, thereafter called *tuple-label*, is regarded as a tuple $\langle l, C_l \rangle$, where l is the label and C_l is the context of the label. Recall also that every time a relationship is found between two labels, their contexts are updated (algorithm in Figure 8). The context of a label is computed by iterative application of the procedure in Figure 9. It provides a semantic characterization of the label in terms of the fields/leaves across the entire domain. So, this context can be used to establish semantic relationships among the labels of the internal nodes from different query interfaces. Given two tuple-labels $\langle l, C_l \rangle$ and $\langle p, C_p \rangle$, if $C_l = C_p$, then l is synonym of p; if $C_l \subset C_p$, then l is a hyponym of p. Using this definition instead of Definition 1, additional relationships among labels can be found. (It can be proved that the new definition is more general than Definition 1.) This definition is not restrained by the existence of bijection/injection between the content words of the labels. For example, who is going in this trip? and How many people are going? cannot be related by Definition 1 but are found to be synonyms. The new definition attains on average a precision of 95%, a recall of 90.4% and an F-score of 92.6%. This algorithm is referred thereafter as *the improved algorithm*. It uses all the features discussed so far.

6. EXPERIMENTS

In this section, we present an experimental evaluation of the approximation algorithm for computing the dictionary of stop words. Another experiment is conducted to assess the ability of the proposed method to establish semantic relationships. The impact of stop words on determining semantic relationships is also studied.

Table 3: Characteristics of interfaces per domain.

Domain	Domain Char. (avg)			Integrated Query Interface			
	Leaves	Nodes	Depth	Leaves	Groups	Depth	Nodes
Airline (20)	10.7	5.1	3.6	24	8	6	13
Auto (20)	5.1	1.7	2.4	18	5	3	7
Book (20)	5.4	1.3	2.3	19	5	2	6
Real Estate (20)	6.7	2.4	2.7	28	8	4	9
Job (20)	4.6	1.1	2.1	19	1	2	2
Car Rental (20)	10.4	2.4	2.5	34	9	4	16
Hotels (30)	7.6	2.4	2.3	26	8	5	15
Credit Card (20)	50.15	20.25	3.6	180	59	6	64
Alliances (50)	15.3	8.32	3.58	31	8	5	12

6.1 Experiment Setup

The testing data set consists of 220 sources over 9 real-world domains on the Web. Each domain has 20 query interfaces, except Hotels that has 30 and Alliances that has 50. Table 3 summarizes the characteristics of the source data sets per domain. The average numbers of fields and internal nodes on the individual interfaces are given in columns 2 & 3. Column 4 shows the average depth of individual interfaces. The characteristics of the resulting global interfaces for the 9 domains are illustrated in columns 5-8.

The general purpose dictionary of stop words contains 319 entries. The dictionary was the first list of stop words extracted from the Web through a query submitted to the Google search engine.

6.2 Evaluating Stop Words

The evaluation of our algorithm for computing the set of stop words with respect to an application domain is performed for each of the 9 domains. The standard F-score [19], which combines precision and recall, is used to measure the effectiveness of finding the correct stop words. For each domain, the intersection of the stop word dictionary and the set of all words appearing in the labels is computed. For example, in the Credit Card domain there are 53 words in the intersection of the general purpose dictionary of stop words and the set of words in the labels of this domain.

The golden standard of stop words for an application domain has been manually generated following the intuition: a word in the intersection list is not a stop word if there is a label whose meaning changes so "drastically" after the removal of the word from the label that the new label does not resemble in any way the original meaning of the label. The remaining words in the list are the golden standard stop words. For example, in the Credit Card domain, the word yourself is not a stop word because there is a label Please tell us about yourself whose meaning completely changes once the word yourself is discarded.

Figure 10 presents the outcome of the experiment for the 9 domains. In Auto, Credit Card, Hotel, Job and Real Estate domains, the algorithm achieves a nearly perfect score. Table 4 shows the only non-stop word missed, yourself. The word appears in the label Please tell us about yourself. The word cannot be discovered as a non-stop word because of the following problems. The only non-stop word with respect to the general purpose dictionary is tell in the label. The label of its sibling is Please give us some financial information. Note that the words Please and us can be removed from the two labels as the stop word constraints are satisfied. The labels

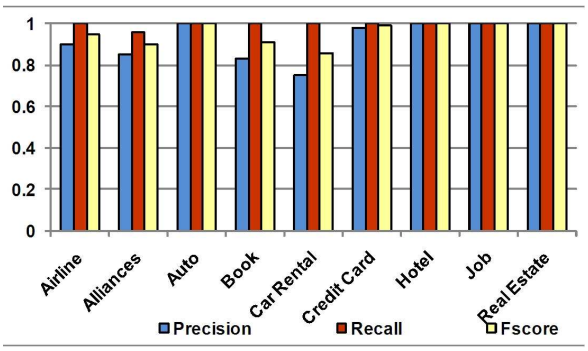

Figure 10: Stop word algorithm evaluation (from left to right Precision, Recall, F-score)

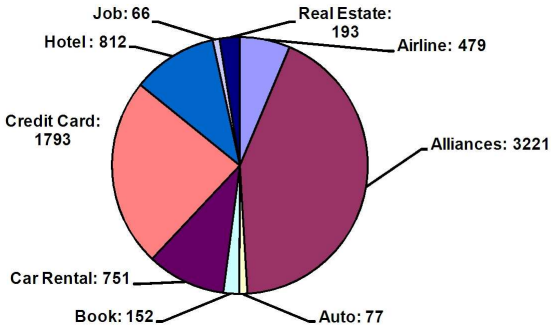

Figure 11: Domain-wise distribution of relationships in the golden standard.

Table 4: Example of non-stop words commonly regarded as stop words

Domain	Found Non-stop words	Missed Non-stop words
Airline	first, last, from, to, when, and, or	where, who
Alliances	from, to, on, yourself, no, for, there, and, or	where, when, who, by
Auto	first, last, from, to, within, or	
Book	first, last, before, or	after
Car Rental	to, and, or	from, last
Credit Card	first, last, per, and, or	yourself
Real Estate	to, from, or	

become tell about yourself and give some financial information. Similarly the words about and some can be removed yielding tell yourself and give financial information, respectively. At this stage there is no restriction precluding the removal of yourself from the label because there is no semantic relationship between what is left from the label and give financial information. For the other domains, the F-score is over 90% except for the Car Rental domain. In this domain we missed two non-stop words from and last (see Table 4). The word last appears in the label Last name and in all the schemas having this label there is no neighboring node with labels pertaining to name. Hence, no stop word constraint is violated when last is removed from Last name.

Efficiency/scalability discussion: The average time to run the algorithm to produce the set of stop words for an application domain is about 30 seconds; the maximum is about 120 seconds. The time complexity of the algorithm is $O(|D| \times |QI| \times NS \times mSL^2 \times mL)$, where D is the set of potential stop words, QI is the set of interfaces in the domain, NS is the number of sets of sibling nodes in an interface, mSL is the maximum number of sibling nodes in an interface and mL is the maximum number of words in a label. The algorithm is scalable, because it is linear in the number of user interfaces and is performed off-line.

6.3 Evaluating Semantic Relationships

Golden standard: For each pair of query interfaces in each of the 9 domains we manually mapped the labels of the internal nodes with the appropriate semantic relationship type. For example, there are 812 relationships in the Hotel domain. In total, the golden standard contains 7,544 relationships. Out of them 4,103 (54.4%) are synonymy relationships and 3,441 (45.6%) are hypernymy/hyponymy relationships. The pie chart in Figure 11 gives the overall distribution of the relationships per domain. A couple of domains stick out: Credit Card and Alliances which together account for more than 66% of all relationships. This is not surprising because Credit Card domain has interfaces 4 to 10 times larger than the interfaces in any other domain (see Table 3). The Alliances domain consists of 50 interfaces, which is twice as large as the number of interfaces in all other domains but Hotel.

Naive Algorithm: Recall that the naive algorithm employs only the senses of the content words present in WordNet. The outcome of the algorithm is evaluated again using Precision, Recall and F-score. The graph in Figure 12 depicts the performance of the algorithm for each of the 9 domains. Notice that the F-score ranges from 39% to 97.3%. The question arises as to why the accuracy of the algorithm is so poor and ranges over such a large interval. There are a couple of explanations. First, as we argued in Section 5, this algorithm blindly compares two labels without taking into consideration where within the interfaces the labels are placed. Second, the vocabulary of the majority of Deep Web sources in the same domain is confined to a limited number of content words. As a case in point, it has been shown that the vocabulary of Deep Web sources in the same domain converges at a very steep rate [19]. Moreover, the labels in the same domain are consistent on some core content words. For example, the word Address is a core term and Employer address, Company address are variations of it in the interfaces shown in Figure 6. Hence, in general, this algorithm wrongly establishes semantic relationships between labels that share some words. Another observation drawn from the graph is that for domains with small interfaces, such as Auto, Book, the performance is significantly better than the performance for domains with larger (complex) interfaces, such as Car Rental, Credit Card. This can be explained by the fact that smaller interfaces have fewer internal nodes, and thus a smaller number of labels, whereas larger interfaces have more internal nodes, and thus many labels. Consequently, in larger interfaces the same words or their synonyms tend to be reused. For instance, there are interfaces where Address appears in both Home Address and Company Address.

The Improved Algorithm: We implemented an algorithm for computing the synonymy and hypernymy relationships that combines the context of labels and semantic enrichment words. The performance of the algorithm is given in Figure 13 in terms of precision, recall and F-score. A brief visual inspection of the graph of this algorithm and that of the naive algorithm shows that the new algorithm has a more stable behavior from one domain to another. Its F-score ranges from 82.1% to 99.3%, with the mean at 92.6%

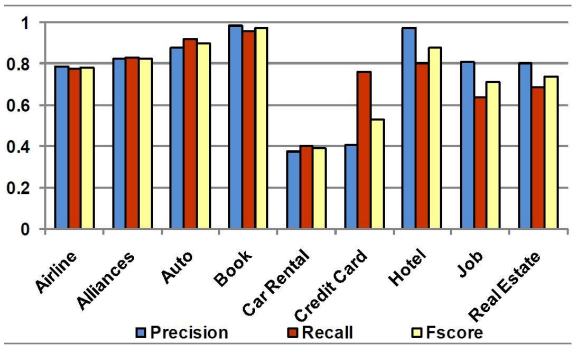

Figure 12: Naive algorithm accuracy (from left to right Precision, Recall, F-score)

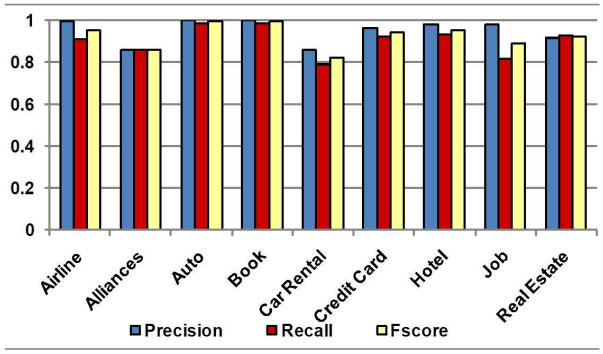

Figure 13: The improved algorithm accuracy (from left to right Precision, Recall, F-score)

and a standard deviation of 5.9%. The naive algorithm has a mean F-score of 74.9% and a standard deviation of 18.5%. Moreover, overall the algorithm improves the average precision to 95%, the average recall to 90.4% and the average F-score to 92.6%. The algorithm has the poorest performance in Car Rental and Alliances. The problem mainly stems from the fact that the designers of these domains use words and phrases that are commonly perceived as synonyms in these domains but not in WordNet. For example, drop-off and return are synonyms in the Car Rental domain but not by WordNet. Notice that while the precision is constantly over 90%, the recall for some domains is around 80%. In addition to the problem we just mentioned, another problem is that many labels are complex sentences, such as So, what do you do for a living?. These labels elude the algorithm.

In Table 5 we randomly selected some examples of pairs of labels from different domains that cannot be related by the improved algorithm. More sophisticated linguistic techniques beyond electronic dictionary abilities are attempted to enable the finding of semantic relationships between such labels. We used the Normalized Google Distance (NGD) [9] for the randomly picked pairs of labels. NGD is a similarity distance to measure the semantic similarity of two words or phrases using the number of hits returned by the search engine Google. We are not able to get consistent results on our labels. As an example, in the Car Rental domain we expect to find that drop-off is a synonym of return. Our algorithm fails to find this relationship between them. So we tried NGD: NGD(drop-off, return) = 1.02. And we also tried NGD(drop-off, leaving) = 0.63. According to NGD, drop-off is more similar to leaving than to return. Similar experiments were performed for more complex cases. One possible explanation for not obtaining consistent results is that to get meaningful hits from Web search engines the queries need to be noun phrases. Many of the relationships we miss are between labels that are not noun phrases (e.g., How flexible are you?). We also tried to use the kernel function for measuring the semantic similarity between pairs of short text snippets reported in [29]. For drop-off and return, the similarity score is very low because most of the top 20 documents returned by Google for the former term refer to varied drop-off locations whereas the documents for the latter refer to return on an investment and the return statement in programming languages.

Stop words and semantic relationships: The following experiments were conducted to understand the role of stop words when determining semantic relationships. For each domain, we run the improved algorithm for computing semantic relationships with the following four possible sets of stop word:

- S_1 is the set of stop words produced by our algorithm;
- S_2 is the gold standard of stop words;
- S_3 is the empty set;
- S_4 is a domain independent stop word set used by a typical IR system; e.g., reference [1].

The F-score of the computed semantic relationships among labels using S_1 is on average 17.6% better than that using S_3. The largest difference is 43% in Car Rental. This shows that it is essential in using a suitable set of stop words. The F-score of the computed semantic relationships among labels using S_2 is only 0.03% above that of using S_1. This shows that our algorithm produces a set of stop words which is close to the ideal set of stop words. The F-score using S_1 is on average 8% better than that using S_4. For the Car Rental domain, the former is 33% better than the latter. If S_4 is used, then a significant number of labels for the leaf nodes (fields) (labels such as from and to in the airline domain) will become empty and integration of query interfaces will not be possible. The graphs of these experiments are omitted due to space limitation.

Table 5: Example of challenging semantic relationships

Domain	Label	Rel	Label
Airline	Outbound	Syn	Origin date
Airline	How flexible are you?	Hyp	Search one day before and after
Car Rental	End	Hyp	Dropoff date
Car Rental	Pick-up	Syn	Start
Car Rental	Pickup Location	Syn	Rental location information
Credit Card	2nd card holder	Syn	Additional authorized user
Credit Card	So, what do you do for a living?	Syn	Employment information
Credit Card	Please tell us about yourself	Syn	Personal information
Real Estate	Size	Hyp	Square feet

7. RELATED WORK

Synonym and near-synonym relationships between short phrases have been recently studied [7, 29]. We are not aware, however, of works that find hypernymy relationships and distinguish hypernyms from synonyms. Nevertheless, there is a great deal of work to represent meaning of words in various areas of research: linguistics, computer science, cognitive phycology, etc. There are two lines of work to develop means to capture the similarity between words. The first one, led by projects like Cyc [23] and WordNet [16], aims to manually create semantic networks of words of such vast pro-

portions that knowledge about the real world spontaneously emerge. The second one aspires to devise generic methods to measure word similarity or word association. There are two directions: those that use word (phrase) frequencies in text corpora [6, 8, 20, 21, 24] and those that employ the Web search engine counts (hits) to identify lexico-syntactic patterns [7, 9, 10]. For the pairs of words we could not find to be semantically related in WordNet (e.g, (drop-off, return), (bed, bedroom)) we tried using the Normalized Google Distance [9]. We were not able to obtain consistent similarity scores between these pairs. NGD was reported to have good results when applied to matching ontologies in the music domain [18]. This work also uses a particular case of the hypernymy definition given in this paper.

There are a number of dictionary-based semantic matching techniques [5, 17, 22] developed for relational/XML schema and ontology alignment. They first annotate the name of each concept of a schema with the most appropriate semantic meaning from a dictionary. This process is semi-automatically carried out. Using this annotation they then use the dictionary to establish semantic correspondences across schemas. For complex phrases not present in the dictionary, they manually assign the most appropriate sense according to the dictionary. Consider the labels Please tell us about your job and Employment information. To determine that the two labels are synonyms, they need to be apriori annotated with the word employment. In this paper we developed a method to automatically compute the relationship between such phrases which is relevant to these works.

8. CONCLUSIONS

The main goal of this work is to address the stop word problem. Its proper handling is a key ingredient to developing fully automated integration solutions. We formulated the stop word problem in the context of Web query interface integration. The problem was shown to be NP-complete and an approximation solution was given. We feel that the formulation of the problem is generic enough to be applicable to other integration areas. Furthermore, we introduced a framework to compute concrete semantic relationships between phrases. Among others, we showed that the problem of computing such relationships can be regarded as finding maximum matchings in bipartite graphs. A comprehensive study of the semantic roles of the words AND and OR within labels/names was also carried out. Extensive experiments were performed to quantitatively assess the performance of our solutions.

Acknowledgements: This work is supported in part by the following NSF grants: IIS-0414939 and IIS-0414981. The authors would also like to express their gratitude to the anonymous reviewers for providing helpful suggestions.

9. REFERENCES

[1] dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words.
[2] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. John Wiley and Sons, Inc., 1999.
[3] L. Barbosa and J. Freire. Combining classifiers to identify online databases. In *WWW*, 2007.
[4] L. Barbosa, J. Freire, and A. Silva. Organizing hidden-web databases by clustering visible web documents. In *ICDE*, 2007.
[5] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. The momis approach to information integration. In *ICEIS*, 2001.
[6] M. Berland and E. Charniak. Finding parts in very large corpora. In *Proc. of ACL*, 1999.
[7] D. Bollegala, Y. Matsuo, and M. Ishizuka. Measuring semantic similarity between words using web search engines. In *WWW*, 2007.
[8] S. A. Caraballo. Automatic construction of a hypernym-labeled noun hierarchy from text. In *ACL*, 1999.
[9] R. L. Cilibrasi and P. M. Vitanyi. The google similarity distance. *TKDE*, 19(3), 2007.
[10] P. Cimiano and S. Staab. Learning by googling. *SIGKDD Explor. Newsl.*, 6(2):24–33, 2004.
[11] H. H. Do and E. Rahm. COMA - A System for Flexible Combination of Schema Matching Approaches. In *VLDB*, pages 610–621, 2002.
[12] E. Dragut, T. Kabisch, C. Yu, and U. Leser. A hierarchical approach to model web query interfaces for web source integration. In *VLDB*, 2009.
[13] E. Dragut, W. Wu, P. Sistla, C. Yu, and W. Meng. Merging source query interfaces on web databases. In *ICDE*, 2006.
[14] E. Dragut, C. Yu, and W. Meng. Meaningful labeling of integrated interfaces. In *VLDB*, 2006.
[15] J. Edmonds and R. M. Karp. Theorectical improvements in the algorithmic efficiency for network flow problems. *ACM*, 19:248–264, 1972.
[16] C. Fellbaum. Wordnet: An on-line lexical database and some of its applications. 1998.
[17] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-match: an algorithm and an implementation of semantic matching. In *SII*, 2005.
[18] R. Gligorov, W. ten Kate, Z. Aleksovski, and F. van Harmelen. Using google distance to weight approximate ontology matches. In *WWW*, 2007.
[19] B. He and K. Chang. Statistical schema matching across web query interfaces. In *SIGMOD*, 2003.
[20] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proc. of ICL*, 1992.
[21] J. J. Jiang and D. W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *ROCLING X*, 1998.
[22] K. Kotis and G. A. Vouros. The hcone approach to ontology merging. In *ESWS*, 2004.
[23] D. Lenat, R. Guha, K. Pittman, D. Pratt, and M. Shepherd. Cyc: Towards programs with common sense. *Communications of the ACM*, 33(9), May 1990.
[24] D. Lin. Automatic retrieval and clustering of similar words. In *Proc. of ICL*, 1998.
[25] J. Madhavan, S. R. Jeffery, S. Cohen, X. L. Dong, D. Ko, C. Yu, and A. Halevy. Web-scale data integration: You can only afford to pay as you go. In *CIDR*, 2007.
[26] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. In *ICDE*, 2002.
[27] J. Munkres. Algorithms for the assignment and transportation problems. *J. SIAM*, 1957.
[28] E. Rahm and P. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 2001.
[29] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW*, 2006.
[30] J. Wang, J.-R. Wen, F. H. Lochovsky, and W.-Y. Ma. Instance-based schema matching for web databases by domain-specific query probing. In *VLDB*, 2004.
[31] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD*, 2004.
[32] Z. Zhang, B. He, and K. Chang. Light-weight domain-based form assistant: querying web databases on the fly. In *VLDB*, pages 97–108, 2005.