Automatic Ontology Matching using Application Semantics

Avigdor Gal*, Giovanni Modica†, Hasan Jamil‡, Ami Eyal§

Abstract

We propose the use of application semantics to enhance the process of semantic reconciliation. Application semantics involves those elements of business reasoning that affect the way concepts are presented to users: their layout, *etc.* In particular, we pursue in this paper the notion of *precedence*, in which temporal constraints determine the order in which concepts are presented to the user. Existing matching algorithms use either syntactic means (such as term matching and domain matching) or model semantic means, the use of structural information that is provided by the specific data model to enhance the matching process. The novelty of our approach lies in proposing a class of matching techniques that takes advantage of ontological structures and application semantics. As an example, the use of precedence to reflect business rules has not been applied elsewhere, to the best of our knowledge. We have tested the process for a variety of Web sites in domains such as car rentals and airline reservations, and share our experiences with precedence and its limitations.

1 Introduction and motivation

The ambiguous interpretation of concepts describing the meaning of data in data sources (*e.g.*, database schemata, XML DTDs, RDF schemata, and HTML form tags) is commonly known as *semantic heterogeneity*. Semantic heterogeneity, a well-known obstacle to data source integration [13], is resolved through a process of *semantic reconciliation*, which matches concepts from heterogeneous data sources. Traditionally, the complexity of semantic reconciliation[13] required that it be performed by a human observer (a designer, a DBA, or a user) [50, 27]. However, manual reconciliation (with or without computer-aided tools) tends to be slow and inefficient in dynamic environments, and for obvious reasons does not scale. Therefore, the introduction of the semantic Web vision [7] and the shift towards machine-understandable Web resources has made clear the importance of automatic semantic reconciliation.

As an example, consider the Web search, an information-seeking process conducted through an interactive interface. This interface may be as simple as a single input field (as in the case of a

*Technion, Israel Institute of Technology, Technion City, Haifa 32000, Israel
†Mississippi State University, Mississippi State University MS 39762, USA
‡Mississippi State University, Mississippi State University MS 39762, USA
§Technion, Israel Institute of Technology, Technion City, Haifa 32000, Israel

general-purpose search engine). Web interfaces may also be highly elaborate: consider a car rental or airline reservation interface containing multiple Web pages, with numerous input fields, that are sometimes content dependent (*e.g.*, when a rented car is to be returned at the point of origin, no input field is required for the return location). A Web search typically involves scanning and comparing Web resources, either directly or via some information portal – a process hampered by their heterogeneity. Following the semantic Web vision, semantic reconciliation should be inherent in the design of smart software agents for information seeking. Such agents can fill Web forms and rewrite user queries by performing semantic reconciliation among different HTML forms.

To date, many algorithms have been proposed to support either semi-automatic or fully automatic matching of heterogeneous concepts in data sources (see Section 1.1 for a detailed discussion). Existing matching algorithms make comparisons based on measures that are either syntactic in nature (such as term matching and domain matching) or based on model semantics. By model semantics, we mean the use of structural information that is provided by the specific data model to enhance the matching process. For example, XML provides a hierarchical structure that can be exploited in identifying links among concepts and thus allow a smooth Web search.

In this paper, we propose the use of application semantics to enhance the process of semantic reconciliation. Application semantics involves those elements of business reasoning that affect the way concepts are presented to users, such as layout. In particular, we pursue in this paper the notion of *precedence*, in which temporal constraints determine the order in which concepts are presented to the user.

We use ontologies as an interface conceptualization tool for representing model and application level semantics to improve the quality of the matching process. Four ontological constructs are used in this work, namely terms, values, composition, and precedence. The first two are syntactical comparison constructs, while the third falls into the category of model-specific semantics, and the fourth belongs to application semantics. These constructs are extracted from form entries and utilized in automatic conceptual modeling of a Web resource. Terms, values, and composition are borrowed from [10, 11]. Precedence, a unique feature of our model, represents the sequence in which terms are laid out within forms, imitating temporal constraints embedded in business rules (*e.g.*, a car rental reservation form would ask for pick-up information before return information).

Given two ontologies, we suggest a set of algorithms to match terminologies in two Web resources. The proposed matching process is enhanced by basic Information Retrieval techniques for string matching. We introduce two new algorithms to compute similarity, based on composition and precedence.

The novelty of our approach lies in the introduction of a sophisticated matching technique that takes advantage of ontological constructs and application semantics. In particular, the use of precedence to reflect business rules has not been applied elsewhere, to the best of our knowledge. We have tested the process for a variety of Web sites in domains such as car rentals and airline reservations, and evaluated the performance of each algorithm individually as well as in combination with others. We highlight the benefits and limits of using the precedence construct as a guideline for future research into application semantics.

We shall use Web sites of car rental companies as a running case study. In our experiments, we extract ontologies from HTML documents. We recognize the fact that XML may serve as a better candidate for ontology exploration. In fact, while one can extract terms and structure from XML documents, one has to **mine** for ontologies in HTML documents. However, current trends in deploying XML as part of the organization data management scheme suggest that while XML may be used for B2B communication, and to some extent as a storage mechanism, interactive sessions still use HTML. Therefore, it is possible that XML data on the server side is "translated" into HTML before being shipped out to the client. It is also worth noting that once an ontology is extracted (from either XML or HTML documents), the process of ontology matching remains unchanged.

1.1 Research background and related work

The study builds upon two existing bodies of research, namely heterogeneous databases and ontology design. Each is elaborated below.

1.1.1 Heterogeneous databases

The evolution of organizational computing, from "islands of automation" into enterprise-level systems, has created the need to homogenize heterogeneous databases. More than ever before, companies are seeking integrated data that go well beyond a single organizational unit. In addition, a high percentage of organizational data is supplied by external resources (*e.g.*, the Web and extranets). Data integration is thus becoming increasingly important for decision support in enterprises [9]. The growing importance of data integration also implies that databases with heterogeneous schemata face an ever-greater risk that their data integration process will not effectively manage semantic differences. This may result, at least to some degree, in the mismatching of schema elements. Hence, methods for schema matching should take into account a certain level of uncertainty. Current research into heterogeneous databases is, however, largely geared towards deterministic semantic resolution [4, 42, 29, 39, 21], which may not effectively scale in computational environments with dynamically changing schemata that require a rapid response. In addition, schema descriptions differ significantly among different domains. It is often said that the next great challenge in the semantic matching arena is the creation of a generalized set of automatic matching algorithms. Accordingly, the goal of this research is to propose the use of application semantics for automatic matching.

Over the past two decades, researchers in both academia and industry have advanced many ideas for reducing semantic mismatch problems, with the goal of lessening the need for manual intervention in the matching process. A useful classification of the various solutions proposed can be found in [44]. Of the categories presented there, we focus on those that deal with the algorithmic aspect of the problem.

The proposed solutions can be grouped into four main approaches. The first approach recommends adoption of Information Retrieval techniques. Such techniques apply approximate, distance-

based (*e.g.*, edit distance [32] as proposed in [35]) matching techniques, thus overcoming the inadequacy of exact, "keyword-based," matching. This approach is based on the presumption that attribute names can be mapped using similar techniques. Attribute names are rarely, however, given in explicit forms that yield good matchings. Furthermore, they need to be complemented by either a lengthier textual description or an explicit thesaurus, which mandates greater human intervention in the process. Protègè utilizes this method in the PROMPT (formerly SMART) algorithm, a semi-automatic matching algorithm that guides experts through ontology matching and alignment [19, 20].

A second approach involves the adoption of machine learning algorithms that match attributes based on the similarity between their associated values. Most efforts in that direction (*e.g.*, GLUE [14] and Autoplex [6]) adopt some form of a Bayesian classifier [31, 15]. In these cases, mappings are based on classifications with the greatest posterior probability, given data samples. Another method, used in the area of natural language processing, involves grammatical inferences [25, 48, 40]: ie, the inference of a grammar G from a set of examples of a language $L(G)$. Machine learning was recognized as playing an important role in reasoning about mappings in [33].

Third, several researchers have suggested the use of graph theory techniques to identify similarities among schemata, where attributes are represented in the form of either a tree or a graph [51, 12, 41]. To give but one example, The TreeMatch algorithm [34] utilizes XML DTD's tree structure in evaluating the similarity of leaf nodes by estimating the similarity of their ancestors.

In a fourth approach, matching techniques from the first three groups are combined. Here, a weighted sum of the output of algorithms in these three categories is used to determine the similarity of any two schema elements. Cupid [34] and OntoBuilder are two models that support this hybrid approach. OntoBuilder, however, is the only framework, to the best of our knowledge, in which application semantics is used as a tool in matching heterogeneous schemata.

Few other systems (MOMIS [5], DIKE [43], and Clio [37], to name a few) aim at resolving semantic heterogeneity in heterogeneous databases. However, these models assume human support in the matching process.

1.1.2 Ontology design

The second body of literature we draw upon focuses on ontology design. An ontology is "a specification of a conceptualization" [26], where conceptualization is an abstract view of the world represented as a set of objects. The term has been used in different research areas, including philosophy (where it was coined), artificial intelligence, information sciences, knowledge representation, object modeling, and most recently, eCommerce applications. For our purposes, an ontology can be described as a set of terms (vocabulary) associated with certain semantics and relationships [46]. Typically, ontologies are represented using a Description Logic [16], where subsumption typifies the semantic relationship between terms; or Frame Logic [30], where a deductive inference system provides access to semi-structured data.

The realm of information science has produced an extensive body of literature and practice in

ontology construction, using tools like thesauri, and in terminology rationalization and matching of different ontologies [2, 49, 53, 56]. Other undertakings, such as the DOGMA project [28, 54], provide an engineering approach to ontology management. Finally, researchers in the field of knowledge representation have studied ontology interoperability, resulting in systems such as Chimaera [36], Protègè [20], and an interactive algorithm for ontology merging (Prompt [20]).

The body of research aimed at matching schemata by using ontologies has focused on interactive methods requiring human intervention, massive at times. In this work, we propose a fully automatic process. Our approach is based on analyzing model-dependent and application-level semantics to identify useful ontological constructs, followed by the design of algorithms to utilize these constructs in automatic schema matching.

The rest of the paper is organized as follows. Section 2 introduces our methodology for generating appropriate ontological constructs. We exemplify the methodology by introducing ontological constructs that are suitable for our case study of the Web search. The ontology matching is described in Section 3, where we detail the hybrid of algorithms we use to match ontologies. Experiments with ontology matching are provided in Section 4. The paper is concluded in Section 5.

An initial report of our methodology was introduced in [38]. The present paper adds a number of improvements, including (1) extended preprocessing techniques, (2) new algorithms for composition and precedence matching, and (3) an extended experimental setting.

2 Ontological constructs

The methodology for the process of schema matching is based on ontological analysis of application classes and the generation of appropriate ontological constructs that may assist in the matching process. We base the ontological analysis on the work of Bunge [10, 11]. We adopt a conceptual modeling approach rather than a knowledge representation approach (in the AI sense). While the latter requires a complete reflection of the modeled reality for an unspecified intelligent task to be performed by a computerized system in the future [8], the former requires a minimal set of structures to perform a given task (a Web search in this case).

We categorize ontological constructs into three classes. The first, and the most common to date, involves the use of syntactic measures to estimate semantic similarity. The second class utilizes model semantics such as the structural information that is provided by a given data model. The third class, unique to this research, involves the use of application semantics, those elements of business reasoning that affect the way concepts are presented to users, their layout and the like. The capabilities of each class complement those of the others.

To exemplify the methodology, we focus on ontological constructs in the general task of the Web search. We recognize the limited capabilities of HTML (and for that matter, also XML) in representing rich ontological constructs, and therefore we have eliminated many important constructs (*e.g.*, the class structure), simply because they cannot be realistically extracted from the content of

Web pages. Therefore, the ontological analysis of this class of applications yielded a subset of the ontological constructs provided by Bunge and added a new construct, which we term *precedence*, for posing temporal constraints.

We shall now provide a brief description of these ontological constructs. The first two constructs fall into the category of syntactical constructs. The third can be classified as a model-dependent semantic construct. Finally, the fourth is an application-dependent semantic construct.

Terms: We extract a set of terms[1] from a Web page, each of which is associated with one or more form entries. A term is a combination of the labeling of the entry and the form entry name. The former is visible to the user and provides a description of the entry content. The latter is utilized for matching parameters in the data transfer process and therefore resembles the naming conventions for database schemata, including the use of abbreviations and acronyms. For example, some of the labels we have extracted from the Avis reservation page are `Airport Location Code`, `Pick-Up Date`, `Pick-Up Time`, `Return Date`, `Return Time`, and `Country`. Entry names include `PICKUP_LOCATION_CODE`, `PICKUP_MONTH`, `PICKUP_HOUR`, `RETURN_MONTH`, `RETURN_HOUR` and `COUNTRY_CODE`. It is worth noting that in this example, several entry names are associated with the same label, *e.g.*, the label `Pick-up Date` is associated with `PICKUP_MONTH` and `PICKUP_DAY` entries.

Values: Based on Bunge [10], an attribute is a mapping of terms and value-sets into specific statements. Therefore, we can consider a combination of a term and its associated data entry (value) to be an attribute. In certain cases, the value-set that is associated with a term is constrained using drop lists, check boxes, and radio buttons. For example, the entry labeled `Pick-Up Date` is associated with two value-sets: {*Day,1,2,■ ,31*} and {*January,February,. . . ,December*}. Clearly, the former is associated with the date of the month and the latter with the month. Whenever constrained value-sets are present, we can enhance our knowledge of the domain, since such constraints become valuable when comparing two terms that do not exactly match through their labels. For example, the label corresponding to `Return Date` in Alamo's Web site is `Dropoff Date`. Although the labels only partially match (see Section 3 for more details), and the words `Return` and `Dropoff` do not appear to be synonymic in general-purpose thesauri (dropoff is not even considered a word in English, according to the Oxford English Dictionary [1]), our matching algorithm matches these terms using their value-sets, since the term `Dropoff Date` has a value-set of {*(Select),1,2,■ ,31*}.

It is our belief that designers would prefer constraining field domains as much as possible, to minimize the effort of writing exception modules. Therefore, it is less likely (although known to happen occasionally) that a field with a dropdown list in one form will be designed as a text field in another form. In the case of a small-sized domain, alternative designs may exist (*e.g.*, *AM/PM* may be represented as either a dropdown list or radio buttons). Since the extraction algorithm represents domains in a unified abstract manner, the end result is independent of the specific form of presentation.

[1]The choice of words to describe ontological structures in Bunge's work had to be general enough to cover any application. We feel that the use of *thing*, which may be reasonable in a general framework, can be misleading in this context. Therefore, we have decided to replace it with the more concrete description of *term*.

Composition: We differentiate atomic terms from composite terms. A composite term is composed of other terms (either atomic or composite). In the Avis reservation Web page, all of the terms mentioned above are grouped under `Rental Pick-Up & Return Information`. It is worth noting that some of these terms are, in themselves, composite terms. For example, `Pick-Up Time` is a group of three entries, one for the hour, another for the minutes, and the third for either *AM* or *PM*. Another composite term in the same Web page is titled `Airline Information` (with the terms `*Airline Name` and `*Flight #,` where the asterisk represents an optional field). The proposed matching algorithm makes use of composition to overcome granularity differences. It is noteworthy that there is a rich body of literature on mereology (*e.g.*, [52, 55]). However, the minimal support of ontological structures in HTML render the subtleties of it immaterial in this framework.

Precedence: The last construct we consider is the precedence relationship among terms. In any interactive process, the order in which data are provided may be important. In particular, data given at an earlier stage may restrict the availability of options for a later entry. For example, the Avis Web site determines which car groups are available for a given session, using the information given regarding the pick-up location and time. Therefore, once those entries are filled in, the information is sent back to the server and the next form is brought up. Such precedence relationships can usually be identified by the activation of a script, such as (but not limited to) the one associated with a SUBMIT button. It is worth noting that the precedence construct rarely appears as part of basic ontology constructs. This can be attributed to the view of ontologies as static entities whose existence is independent of temporal constraints. We anticipate that contemporary applications, such as the one presented in this paper, will need to embed temporal reasoning in ontology construction.

The main difference between the first and second categories on the one hand, and the third category on the other, is that the equivalence of the construct in the data model is given explicitly in the former but is only implicit in the latter. In our example, terms are explicitly available as labels and entry names, and values are explicitly available as value-sets. Composition is explicitly available in XML definitions through its hierarchical structure.[2] The precedence construct, on the other hand, is only implicitly given, through the process of form submission. We argue that by utilizing implicit constructs, in addition to explicit ones, matching quality can be improved.

3 Ontology matching

In the matching process, two ontologies are merged, refining and generalizing an existing ontology. We denote by *Web resource dictionary* the set of terms extracted from a Web resource (typically composed of several Web pages within a single Web site). Let $\{v_1, v_2, ..., v_n\}$ and $\{u_1, u_2, ..., u_m\}$ be two Web resource dictionaries. The general matching process is conducted in two steps. First, pair-wise matching yields a similarity measure μ_{v_i,u_j} for all pairs $v_i \in \{v_1, v_2, ..., v_n\}$ and $u_j \in \{u_1, u_2, ..., u_m\}$. Next, a subset of the $n \times m$ pair-wise matching (dubbed a *mapping*) is selected

[2]Forms are given in HTML, which does not have a composition construct per se. Yet, our methodology transforms the HTML code into an XML definition, to be utilized in the process of schema matching.

as the "best" mapping between the two ontologies. Such a mapping may utilize some variation of a weighted bipartite graph matching [24], if the required mapping is of a $1:1$ nature. Finding a weighted bipartite matching is a well-researched problem, and efficient implementations of it (having time complexity $O\left((n+m)^3\right)$) can be found in the literature (see, for example, [24]). For a matching process that yields $1:n$ mappings, a simpler algorithm ($O(nm)$) may be applied, in which a term v_i in a candidate dictionary is mapped into a term u_j in a target dictionary if $\mu_{v_i,u_j} = \max_{1\leq k\leq m} \mu_{v_i,u_k}$. Such an algorithm enables duplicate entries in the candidate dictionary, yet does not allow the partition of a single value in the target dictionary to several values in the candidate dictionary. In Section 3.2 we show a simple method for enabling more complex forms of $1:n$, $n:1$, and $n:m$ mappings by normalizing known domains into atomic components.

This process of ontology matching is formalized and discussed in depth in [22]. In particular, we have shown in [22] that the specific methodology described herein is well suited to identifying the exact matching (as perceived by a human observer) as the matching with the highest sum (or average) of similarity measures of the selected term pairs.

This section focuses on the pair-wise matching. We describe four methods for matching, based on the four ontological constructs presented in Section 2. Section 3.1 introduces term matching. Value matching is discussed in Section 3.2. Section 3.3 is devoted to composition matching, and precedence matching is detailed in Section 3.4. We then show how to combine these methods (Section 3.5) into an overall similarity measure. We discuss the complexity of all operations separately. The overall process of ontology matching, in the case of $1:1$ matching, is dominated by the process of identifying the best mapping, giving rise to a cubic execution complexity. In the case of a $1:n$ mapping, and assuming the two ontologies are of comparable sizes, the overall complexity becomes quadratic.

3.1 Term matching

Term matching compares labels and names to identify syntactically similar terms. To achieve better performance, terms are preprocessed using several techniques originating in IR research. We have used the following preprocessing techniques in our research.

Capitalization-based separation: Mid-word capitalization (*e.g.*, `firstName` and `PUTime`) is interpreted as a concatenation of words (some of which may be acronyms). Therefore, we separate words based on mid-word capitalization. In the example given above, `firstName` is replaced with `first Name` and `PUTime` is interpreted as an acronym (`PU`) followed by `Time`.

Normalization: All uppercase characters are transformed into lowercase characters. Thus, `first Name` is replaced with `first name`.

Ignorable Character Removal: Characters such as '*', '/', '-', *etc.* are treated as "noise" and considered dispensable, and as such are removed from the terms. Hence, after this step, terms such as `*Country` and `country` are considered identical.

De-hyphenation: Labels such as `pick-up` and `pick up` are considered identical (*e.g.*, [17]). Hence, hyphens in labels are removed to improve matching and hyphenated words are merged (`pick-up` is replaced with `pickup`).

Stop Term Removal: Common terms such as 'a', 'to', 'in', and 'the' are considered stop terms (*e.g.*, [18]), and are removed.

Preprocessing is a collection of smaller algorithms (subroutines). These algorithms take a term as an input and return a transformed term. In almost all cases, the algorithms make a single pass over a term in order to transform the term as they process, while other cases require a certain constant number of passes (*e.g.*, Capitalization-based separation requires three passes). Therefore, given two ontologies of size n and m, the complexity of preprocessing is $O(k(n+m))$, where k is the maximal term size (measured in characters).

We have applied two separate methods for term matching based on string comparison as follows:

Word matching: Two terms are matched and the number of common words is identified. The similarity of two terms t_1 and t_2 using word matching (dubbed $\mu^W_{v_i,u_j}$) is defined as the ratio between the number of common words in t_1 and t_2 and the total number of different words in terms t_1 and t_2, providing a symmetric measure of the semantic similarity of these two terms. The more common words the terms share, the more similar they are considered to be. For example, consider the terms t_1=`Pickup Location` and t_2=`Pick-up location code`. The revised terms after preprocessing are $\bar{t}_1$=`pickup location` and $\bar{t}_2$=`pickup location code`. The term similarity, using word matching, is computed as

$$\mu^W_{t_1,t_2} = \frac{2\,(\texttt{pickup},\ \texttt{location})}{3\,(\texttt{pickup},\ \texttt{location},\ \texttt{code})} = 66\%$$

Two words $w_1 \in t_1$ and $w_2 \in t_2$ are considered to be common if at least one of the following three conditions holds:

Spelling: w_1 and w_2 are spelled the same. For example, $w_1 = w_2 =$`pickup`.

Soundex: Terms are compared and judged to be similar based on the way they sound, rather than the way they are spelled. Each word is encoded as a four character string, where similar words share more characters. For example, the word `pick` is represented as P200 and `pickup` is represented as P210. Two terms are identified as a match if their soundex encoding is at least 75% similar and the first letters in each encoding match. In the example given above, `pick` is matched with `pickup`. A comparison of the terms `Pickup Time` and `Pick Up Time` yields $\frac{2\,(\texttt{pick/pickup},\ \texttt{time})}{3\,(\texttt{pick/pickup},\ \texttt{up},\ \texttt{time})} = 66\%$ match effectiveness using the soundex method.

Thesaurus: Terms and labels are matched using an ever-expanding thesaurus, based on a publicly available thesaurus such as WordNet[3] and extended through user intervention. Mismatched terms can be presented to the user for manual matching. Every manual match identified by the user is accepted as a synonym, and expands and enriches the thesaurus.

[3] http://www.cogsci.princeton.edu/~wn/

Spelling, soundex, and thesaurus take two terms as input, and return a value as an indicator of the measure of similarity of the input terms. The process involves breaking down the terms into words, and checking for their membership in a number of word lists and collections. Hence, the process complexity is $O\left(\left(k'\left(n+m\right)\right)^2\right)$, where $n+m$ is the total number of terms in both ontologies and k' is the maximal number of words in a term. The quadratic behavior stems from the need to insert each word into either an intersection or a union list of words (contained in the two terms). In the worst case, the number of times we scan the lists constitutes an arithmetic series, the summation of which is $\frac{(n+m)(n+m+1)}{2}$, giving rise to the quadratic behavior.

String matching: We find the maximum common substring between two terms whose words have been concatenated by removing white spaces. The similarity of two terms using string matching (dubbed $\mu^S_{v_i,u_j}$) is computed as the length of the maximum common substring as a percentage of the length of the longest of the two terms. As an example, consider the terms `airline information` and `flight airline info`, which after concatenating and removing white spaces become `airlineinformation` and `flightairlineinfo`, respectively. The maximum common substring is `airlineinfo`, and the effectiveness of the match is $\frac{length(\texttt{airlineinfo})}{length(\texttt{airlineinfomation})} = \frac{11}{18} = 61\%$.

The string matching algorithm basically iterates over the two input terms twice in a nested fashion – the outer loop runs over the entire length of the first term, and the inner loop roughly inspects for every possible substring match with the second term for every position in the first string. Therefore, the execution complexity is $O(k^2)$, where k is the maximal length of a term.

Given two ontologies with n and m terms, these methods are applied pair-wise for each of the $n \times m$ possible combinations. For each combination we compare labels and entry names separately. The overall complexity of the process can be computed as $O\left(k\left(n+m\right)\right) + O\left(\left(k'\left(n+m\right)\right)^2\right) + O(k^2) = O\left(\max\left(\left(k'\left(n+m\right)\right)^2, k^2\right)\right)$. Typically we deal with ontologies of more than 100 terms, so that $n \approx m \gg k \approx k'$, and thus the overall processing takes $O(n^2)$.

We define a threshold (t^T) to identify a reasonable match. Any match with less than t^T is discarded. This threshold can be adjusted by the user.

For each pair, we compute four figures (two for labels, $\mu^{W,L}_{v_i,u_j}$ and $\mu^{S,L}_{v_i,u_j}$, and two for names, $\mu^{W,N}_{v_i,u_j}$ and $\mu^{S,N}_{v_i,u_j}$). We combine the figures into one figure, representing the strength of the match. Therefore, the similarity measure of a term v_i with a term u_j is computed as the weighted average

$$\mu^T_{v_i,u_j} = \omega^{W,L}\mu^{W,L}_{v_i,u_j} + \omega^{S,L}\mu^{S,L}_{v_i,u_j} + \omega^{W,N}\mu^{W,N}_{v_i,u_j} + \omega^{S,N}\mu^{S,N}_{v_i,u_j} \quad (1)$$

where $\omega^{W,L}$, $\omega^{S,L}$, $\omega^{W,N}$, and $\omega^{S,N}$ are positive weights that sum to unity.

Example 1 (Term matching) *Consider the terms* `Specify Pick Up Location Code * (pickUpCode)` *and* `ID for Pick-up Location (pickupID)`. *According to our convention, a label appears first followed by a name enclosed in parentheses.*

After preprocessing, the two terms become ***specify pick up location code (pick up code)*** *and* ***id pickup location (pickup id)****. Recall that* ***pick*** *sounds like* ***pickup*** *and assume the use of a thesaurus where* ***code*** *is a synonym of* ***id****.*

- *Label matching:*
  - *Word matching:*

$$\frac{|\{location,\ pick/pickup,\ code/id\}|}{|\{specify,\ pick/pickup,\ up,\ location,\ code/id\}|} = \frac{3}{5} = 60\%$$

  - *String matching: after removing spaces one has* ***specifypickuplocationcode*** *and* ***idpickuplocation*** *as the labels for comparison. The maximum common substring is* ***pickuplocation****.*

$$\frac{length(\texttt{pickuplocation})}{length(\texttt{specifypickuplocationcode})} = \frac{14}{25} = 56\%$$

- *Name matching:*
  - *Word matching:*

$$\frac{|\{pick/pickup,\ code/id\}|}{|\{pick/pickup,\ up,\ code/id\}|} = \frac{2}{3} = 66\%$$

  - *String matching: after removing spaces one has* ***pickupcode*** *and* ***pickupid*** *as the names for comparison. The maximum common substring is* ***pickup****.*

$$\frac{length(\texttt{pickup})}{length(\texttt{pickupcode})} = \frac{6}{10} = 60\%$$

Assuming $\omega^{W,L} = \omega^{S,L} = 0.4$ *and* $\omega^{S,N} = \omega^{W,N} = 0.1$*, term-based similarity is computed, using formula 1 above, as follows:*

$$\mu^{T}_{v_i,u_j} = 0.4 * 60\% + 0.1 * 56\% + 0.4 * 66\% + 0.1 * 60\% = 62\%$$

□

3.2 Value matching

Value matching utilizes domain constraints to compute similarity measure among terms. Using value matching, one can avoid matching terms that are clearly unrelated, such as hours and minutes, though their syntactic description may be similar (*e.g.*, `Pick-Up Time (PICKUP_HOUR)` and `Pick-Up Time (PICKUP_MINUTE)`).

Fields with select, radio and check box options are processed using their value-sets. Therefore, different design methods act as no barrier in extracting the actual value sets. Value sets are pre-processed to result in generic domains. By recognizing separators in well-known data types, such

as‘ /’, ‘-’, and ‘.’ in date structures, ‘:’ in time structures, ‘()’ in telephone numbers, ‘@’ in e-mail addresses, and ‘http://’ in URLs, domains can be partitioned into basic components, creating a compound term. The name of each new subterm is constructed as a concatenation of the existing name and the recognized domain type (*e.g.*, day). For example, the term `Pickup Date (pick_date)`, which is recognized as a *date* field based on its domain entries, is further decomposed into three subterms: `Pickup Date (pick_date_day)`, `Pickup Date (pick_date_month)`, and `Pickup Date (pick_date_year)`. It is worth noting that such preprocessing also affects term matching by generating additional terms, and therefore is performed prior to term matching.

The use of domain-based normalization is pivotal in overcoming a major obstacle in schema matching, involving query rewriting using value partitions and compositions. To illustrate this point, consider a situation in which an agent is provided with a sample schema and a query over this schema (*e.g.*, a sample form, filled by a user). Next, the agent is provided with a new schema (*e.g.*, a new form) and must rewrite the original query to adapt it to the new schema (*e.g.*, must fill out the new form). Assuming a schema mapping can be found, the agent is still faced with the problem of how to partition (or concatenate) values so as to rewrite the query. Normalization becomes handy whenever such partitioning or concatenation is performed on known domains. Once normalized, atomic values can be mapped, and when joined together according to the new schema, a new value (processed using existing atomic and composite terms) is generated for the query rewriting process.

Similarity is calculated as the ratio between the number of common values in the two value sets and the total number of different values in them. For example, suppose that t_1=`Return time` and t_2=`Dropoff time` with values {*10:00am,10:30am,11:00am*} and {*10:00am,10:15am,10:30am,10:45am,11:00am*}, respectively. Preprocessing separates the domains into hour values ({*10,11*} versus {*10,11*}), minutes values ({*00,30*} versus {*00,15,30,45*}), and the value {*am*}. There is a perfect match in the hour domain, yet the minutes domains share two values (*00* and *30*) out of four (*00*, *15*, *30*, and *45*). Thus, the similarity is calculated as $\frac{2}{4} = 50\%$. The power of value matching can be further highlighted using the case of `Dropoff Date` in Alamo and `Return Date` in Avis. These two terms have associated value sets {(*Select*),*1,2,*■ ,*31*} and {(*Day*),*1,2,*■ ,*31*} respectively, and thus their content-based similarity is $\frac{31}{33} = 94\%$, which improves significantly over their term similarity ($\frac{1(\texttt{Date})}{3(\texttt{Dropoff, Date, Return})} = 33\%$).

The dominant operations in domain normalization and value matching are the subroutines for word similarity and string matching. The value matching is performed for pairs of domains, each represented as a list of terms. We compute word similarity and string matching for all possible value pairs. Given a maximum length of a value p, and a maximum domain size q, then the running time for value matching is $O\left((pq)^2\right)$.

The domain recognition component can overcome differences of representation within the same domain. For example, we can apply transformations, such as converting a 24-hours representation into one of 12 hours. Thus, a domain {*10:00, 11:00, 12:00, 13:00*} in a 24-hours representation can be transformed into three domains {*1, 10, 11, 12*}, {*00*}, and {*am, pm*} in a 12-hours representation.

Value matching is influenced not only by its domain entries as explained above, but also by the

	Boolean	Float	Integer	Date	Time	Email	URL
Boolean	0.5	0	0	0	0	0	0
Float	0	0.1	0.1	0	0	0	0
Integer	0	0.1	0.2	0	0	0	0
Date	0	0	0	0.3	0	0	0
Time	0	0	0	0	0.3	0	0
Email	0	0	0	0	0	0.4	0
URL	0	0	0	0	0	0	0.4

Table 1: Domain type similarity

term domain type. This technique is adopted from Cupid, and assumes that a similarity of a pair of domains is assigned with a positive weight $0 \leq \omega^D \leq 1$. Table 1 shows possible combinations of domain types and their associated similarity weights. For example, similarity of type *url* is weighted at 40%, even if their domain entries are not the same. Tuning the parameters depends on the weight one sets to the impact of domain in term differentiation. Therefore, if one expects only a few terms with time domain, their relative weight will be high. This tuning can be done either manually or as a training process. For the latter, the system uses the percentage of terms from a given domain in the ontologies with which the system has experience to compute ω^D.

Similarly to term matching in Section 3.1, value matching is applied pair-wise to each of the $n \times m$ possible combinations and any match below the threshold is discarded. Given a pair, v_i and u_j, we compute the similarities of their domain entries as $\mu^E_{v_i,u_j}$. The overall matching similarity is computed as the weighted average

$$\mu^V_{v_i,u_j} = \omega^D + \omega^E \mu^E_{v_i,u_j} \tag{2}$$

where ω^D is taken from Table 1 and $\omega^E = 1 - \omega^D$.

Example 2 (Value matching) *Consider the following two terms,* `Drop-off Date (drop_date)` *with a domain* {January 2002 ,..., December 2002} *and* `Return Date (return_date)` *with a domain* {Aug-2002 ,..., Dec-2002, Jan-2003, ..., Jul-2003}. *After applying the domain recognition technique both terms are categorized as Date and their domain similarity is weighted at* $\mu^D_{v_i,u_j} = 0.3$*, according to Table 1. Domain decomposition will subdivide both terms into three fields:*

- *Day: Since no day is specified in the domain entries, the first of each month is arbitrarily assumed, creating two subterms* `Drop-off Date (drop_date_day)` *and* `Return Date (return_date_day)`*, both with a single value domain,* {1}*. The domain matching results in a* 100% *similarity.*
- *Month: Two subterms are created,* `Drop-off Date (drop_date_month)` *and* `Return Date (return_date_month)`*, both with a domain* {Jan, ..., Dec} *(note the normalization of month representation). The domain matching results in a* 100% *similarity.*
- *Year: Two subterms are created,* `Drop-off Date (drop_date_year)` *with a domain* {2002} *and* `Return Date (return_date_month)` *with a domain* {2002,2003}*. When comparing both domains, we obtain a* 50% *similarity (only one value is common to both domains).*

The overall value similarity for each of the three terms, using Formula 2, is $0.3 + 0.7 * 100\% = 100\%$ *for the day and month and* $0.3 + 0.7 * 50\% = 65\%$ *for the year.* □

Given two terms, v_i and u_j, we compute a linguistic similarity $\mu^L_{v_i,u_j}$ as a weighted average of the term and value similarity measures. That is:

$$\mu^L_{v_i,u_j} = \frac{\omega^T \mu^T_{v_i,u_j} + \omega^V \mu^V_{v_i,u_j}}{\omega^T + \omega^V} \tag{3}$$

where ω^T and ω^V are positive weights such that $\omega^T + \omega^V \leq 1$. Linguistic similarity is the basis for computing composition and precedence matching, as will be discussed shortly. It is worth noting that $\omega^V = 0$ whenever there are no constraints associated with the matched terms. Thus, in a form setting, we assign $\omega^V = 0$ whenever both terms are associated with a text field. The $\mu^L_{v_i,u_j}$ are stored in an $n \times m$ matrix (M_G), to be used in composition and precedence similarity.

3.3 Composition matching

Composition in Web forms is constructed using three methods, namely *multiple term association*, *name similarity*, and *domain normalization*. Multiple term association refers to the association of multiple terms with the same label, where all terms are named and grouped under that label. Name similarity groups entry labels that share identical prefixes. For example, the terms `Pick-Up Location Code`, `Pick-Up Date`, and `Pick-Up Time` in the Avis Web site are grouped under `Pick-Up`. Domain normalization was discussed in Section 3.2 and involves the splitting of a term into subterms through recognition of known domains (such as day and time). Other techniques may be employed for determining composition. For example, one may use various framing constructs of HTML, such as color alterations and headings, to identify composition. Using such techniques, one may associate terms (such as `Pick-Up Location Code`) with entry-less labels (such as `Rental Pick-Up & Return Information`).

To avoid ambiguity at an intra-dictionary level, one needs to ensure term uniqueness. We ensure term uniqueness by concatenating labels with entry names, to form *label (name)* as a term. As an example, consider the composed term `Pick-Up Time`. The terms in this composed term are named `Pick-Up Time (PICKUP_HOUR)`, `Pick-Up Time (PICKUP_MINUTE)`, and `Pick-Up Time (PICKUP_AM_PM)`.

Given a Web resource dictionary D, we define a *composition graph* of D to be a directed graph $G = (V, E)$ such that $V = D$ and $(u, v) \in E$ if u belongs to the group v. It is worth noting that G is not necessarily a tree, since the various grouping methods may associate the same node with different ancestors. The following proposition ensures a weaker property of G.

Proposition 1 *Let* $G = (V, E)$ *be a composition graph.* G *is a directed acyclic graph.*

We next detail a graph-based matching algorithm. The algorithm is based on a technique we dub *graph pivoting*, as follows. When matching two terms, we consider each of them to be a pivot

within its own ontology, thus partitioning the graph into semantically related subgraphs. The semantics of pivoting is taken from the ontological analysis. The algorithm receives as an input two composition graphs, serving as candidate and target ontologies, and a matrix of pair-wise linguistic similarity measures, as discussed in Section 3.2. The output is a matrix of pair-wise composition similarity measures $M'_G = \mu^C_{v_i,u_j}$.

Given two terms v_i and u_j, in the candidate and target ontologies, respectively, we partition each ontology into three parts, namely siblings, ancestors, and unrelated terms. Siblings are those nodes that share a parent with v_i (or u_j in its respective ontology). Ancestors are those nodes on a path to v_i (and u_j respectively). All other nodes are considered to be unrelated. The algorithm attempts to match, using linguistic matching, siblings with siblings and ancestors with ancestors (regardless of the relationships among the ancestors). The similarity of these two "subontologies" is combined as $\mu^C_{v_i,u_j}$ using weights, just as for term and value matching. Therefore, given a pair v_i and u_j, we compute their subontology similarities $\mu^B_{v_i,u_j}$ (for siblings) and $\mu^R_{v_i,u_j}$ (for ancestors). The overall matching similarity is computed as the weighted average

$$\mu^C_{v_i,u_j} = \omega^B \mu^B_{v_i,u_j} + \omega^R \mu^R_{v_i,u_j} \tag{4}$$

where ω^B and ω^R are positive weights that sum to unity. As a default, each similarity measure contributes equally to the composition matching similarity. Another alternative gives greater weight to larger ontologies, thus decreasing error rates.

Algorithm 1 (Composition matching) *Input: G', G'', and M_G*

Output: $M'_G = (\mu^C_{v_i,u_j})$

01. `Initialize` M_G `using` $m_{i,j} = \mu^L_{u_i,v_j}$
02. `for each term` $u \in V'$
03. `for each term` $v \in V''$
04. $\mu^R_{u,v}$ `= getAncestorsConfidence(u,v)`
05. $\mu^B_{u,v}$ `= getSiblingsConfidence(u,v)`
06. $\mu^C_{u,v} = \omega^B \mu^B_{v,u} + \omega^R \mu^R_{v,u}$
07. $M_{G'}[u][v] = \mu^C_{u,v}$
08. `end for`
09. `end for`
10. `return` $M_{G'}[u][v]$

The `getAncestorsConfidence` and `getSiblingsConfidence` functions produce the best mapping between two subontologies, as discussed at the beginning of Section 3. The match is performed according to the matching cardinality constraints (either $1:1$ or $1:n$).

Example 3 (Composition matching) *Figure 1.a provides the mapping with the highest similarity measure between two ontologies, using the linguistic matching $\mu^L_{v_i,u_j}$ of Formula 3. It is*

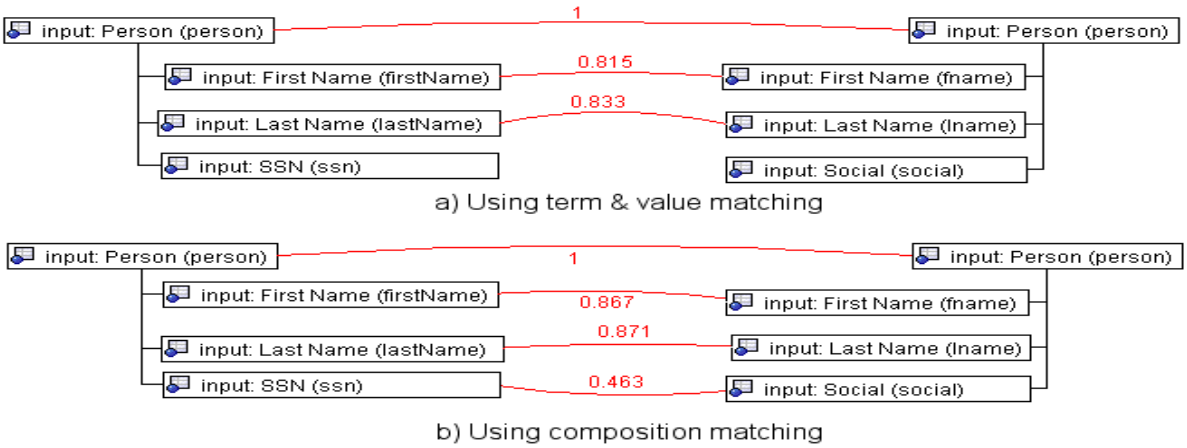

Figure 1: Composition matching example

noteworthy that the terms ***SSN (ssn)*** *and* ***Social (social)*** *are not considered a match, unless we specify them as synonyms in a thesaurus. The composition matching algorithm, however, matches both terms based on the mappings of their ancestors and siblings. To illustrate how the algorithm works, assume that* M_G *is given by the values in Figure 1.a, so for example,* $M_G[\texttt{FirstName(fname)}][\texttt{FirstName(firstName)}] = 81.5\%$.

For $u =$***SSN (ssn)***, $ancestors(u) = \{$***Person (person)***$\}$ *and* $siblings(u) = \{$***First Name (firstName)***, ***Last Name (lastName)***$\}$. *For* $v =$***Social (social)***, $ancestors(v) = \{$***Person (person)***$\}$ *and* $siblings(v) = \{$***First Name (fname)***, ***Last Name (lname)***$\}$. *Therefore,* $\mu^R_{u,v} = 100\%$ *and* $\mu^B_{u,v} = 0.5 \times 0.815 + 0.5 \times 0.833 = 82.4\%$. *Assuming equal weights for ancestors* (ω^R) *and siblings* (ω^B), *the overall matching similarity can be computed, using Formula 4, to be* $\mu^C_{u,v} = 91.2\%$. *The similarity of* ***SSN (ssn)*** *and* ***Social (social)***, *as appears in Figure 1.b, combines the linguistic similarity measure (which is close to zero) and* $\mu^C_{u,v}$ *(91.2%)* □

The algorithm considers all $n \times m$ term combinations. For each one, it computes similarities twice (once for ancestors and once for siblings). Using Proposition 1, the extraction of the subontologies is performed in $O(n+m)$. Therefore, when applying a $1:1$ matching, composition matching runs at $O\left(nm\,(n+m)^3\right) \sim O\left(n^5\right)$ for ontologies of similar size. For $1:n$ matching, the algorithm runs at $O\left(n^2m^2\right) \sim O\left(n^4\right)$ for ontologies of similar size.

3.4 Precedence matching

Let u_i and u_j be atomic terms in a Web resource dictionary. u_i *precedes* u_j if one of the following two conditions is satisfied:

1. u_i and u_j are associated with the same Web page and u_i physically precedes u_j in the page.

2. u_i and u_j are associated with two separate Web pages, U_i and U_j, respectively, and U_i is presented to the user before U_j.

Evaluating the first condition is easily achieved when the page is extracted into a DOM tree (short for Document Object Model), a W3C standard that can be used in a fairly straightforward manner to identify form elements, labels, and input elements. The properties of the precedence relation are summarized in the following proposition.

Proposition 2 *The precedence relation is irreflexive, antisymmetric, and transitive.*

The precedence relationship, as presented in this paper, serves as a crude estimation of the actual time constraints of a business process. For example, while it is clear that car rental companies would be likely to inquire about pick-up information before return information, there is no reason why either shipping address or invoice address should take precedence in a purchase order. The crudeness of this method stems from the minimal information provided by Web pages regarding the true sequence of terms. Web server cooperation would enable a deeper analysis, including the analysis of scripts, to refine term dependency. One outcome of the relationship's crude definition is the identification of many false positives by the algorithm (*i.e.*, many false good matches).

Precedence matching works similarly to composition matching using graph pivoting. Given an atomic term v_i in a Web resource dictionary with n terms $\{v_1, v_2, ..., v_n\}$, we can compute the following two sets:

- $precede(v_i) = \{v_j | v_j \text{ precedes } v_i\}$
- $succeed(v_i) = \{v_j | v_i \text{ precedes } v_j\}$

It is worth noting that, following Proposition 2, $precede(v_i) \cap succeed(v_i) = \varnothing$. Given two terms, v and u, from two Web resource dictionaries, we consider u and v to be pivots within their own ontologies. Therefore, we compute the similarity measure of matching $precede(v)$ with $precede(u)$ ($\mu^{\leftarrow}_{v,u}$), and $succeed(v)$ with $succeed(u)$ ($\mu^{\rightarrow}_{v,u}$). This computation is based on a linguistic similarity measure. Presumably, terms will tend to match better if both those that precede them and those that succeed them do so.

Given a pair, v_i and u_j, we compute their subontology similarities $\mu^{\leftarrow}_{v_i,u_j}$ and $\mu^{\rightarrow}_{v_i,u_j}$. The overall matching similarity is computed as the weighted average

$$\mu^{P}_{v_i,u_j} = \omega^{\leftarrow}\mu^{\leftarrow}_{v_i,u_j} + \omega^{\rightarrow}\mu^{\rightarrow}_{v_i,u_j} \tag{5}$$

where $\omega^{\leftarrow}$ and $\omega^{\rightarrow}$ are positive weights that sum to unity. Again, as a default, each similarity measure contributes equally to the composition matching similarity.

Example 4 (Precedence matching) *Figure 2.a illustrates the matching of terms between two ontologies for car-rental reservation application, after applying linguistic matching. The candidate*

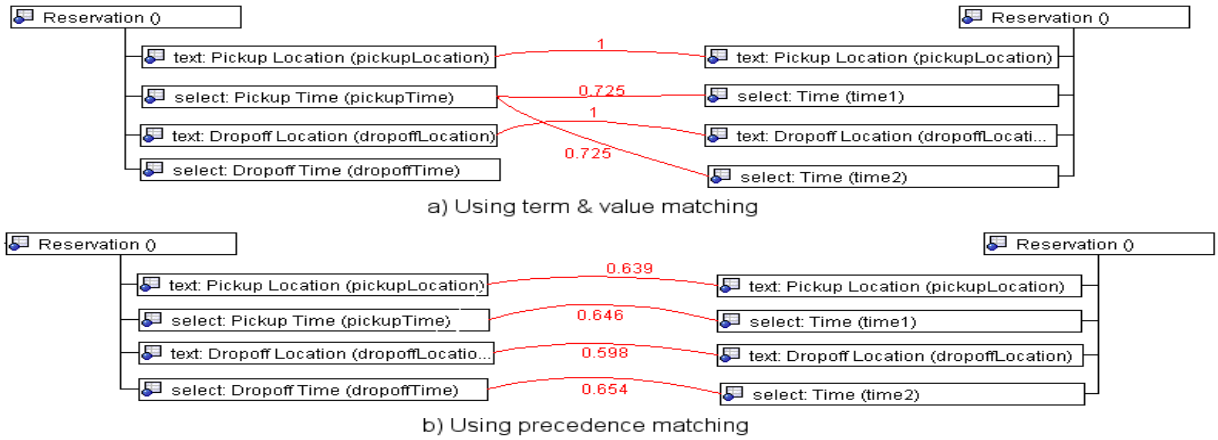

Figure 2: Precedence matching example

ontology (on the right) distinguishes implicitly between pick-up and drop-off times, based on their proximity to the `Pickup Location (pickupLocation)` *and* `Droppoff Location (dropoffLocation)`. *Using linguistic matching, we can separate* `Pickup Location` *from* `Dropoff Location`. *However, the time terms (both have domain* {12:00am ,...., 11:00pm}*) cannot be differentiated. The precedence matching algorithm matches both terms correctly, based on the presentation order of those terms within the HTML page. For ease of presentation, we have refrained from normalizing the time attributes.*

The following sets were identified for $v_1 =$`Time (time1)` *and* $v_2 =$`Time (time2)`*:*

$precede(v_1) = \{$`Pickup Location (pickupLocation)`$\}$
$precede(v_2) = \{$`Pickup Location (pickupLocation), Time (time1), Dropoff Location (dropoffLocation)`$\}$
$succeed(v_1) = \{$`Dropoff Location (dropoffLocation), Time (time2)`$\}$
$succeed(v_2) = \{\}$

The following sets were identified for $u_1 =$`Pickup Time (pickupTime)` *and* $u_2 =$`dropoffTime (dropoffTime)`*:*

$precede(u_1) = \{$`Pickup Location (pickupLocation)`$\}$
$precede(u_2) = \{$`Pickup Location (pickupLocation), PickupTime (pickupTime), Dropoff Location (dropoffLocation)`$\}$
$succeed(u_1) = \{$`Dropoff Location (dropoffLocation), Droppoff Time (dropofftime)`$\}$
$succeed(u_2) = \{\}$

The precedence similarity measures, using the time terms as pivots and assuming equal weights to precede and succeed similarity measures, are given in the following table:

	$\mu^{\leftarrow}_{v,u}$	$\mu^{\rightarrow}_{v,u}$	$\mu^{P}_{v,u}$
v_1 = *Time (time1)* and u_1 = *Pickup Time (pickupTime)*	1	$\frac{1+0.725}{2} = 0.86$	$\frac{1*0.5+0.86*0.5}{0.5+0.5} = 0.93$
v_1 = *Time (time1)* and u_2 = *Dropoff Time (dropoffTime)*	1	0	$\frac{1*0.5+0*0.5}{0.5+0.5} = 0.5$
v_2 = *Time (time2)* and u_1 = *Pickup Time (pickupTime)*	$\frac{1+0+0.5}{3} = 0.5$	0	$\frac{0.5*0.5+0*0.5}{0.5+0.5} = 0.25$
v_2 = *Time (time2)* and u_2 = *Dropoff Time (dropoffTime)*	$\frac{1+0.725+1}{3} = 0.90$	1	$\frac{1*0.5+0.90*0.5}{0.5+0.5} = 0.95$

The algorithm chooses the match with maximum similarity, thus matching `Time (time1)` *with* `Pickup Time (pickupTime)` *and* `Time (time2)` *with* `Dropoff Time (dropoffTime)`*. The combination of linguistic matching with precedence matching produces the overall matching similarity as shown in Figure 2.b.* □

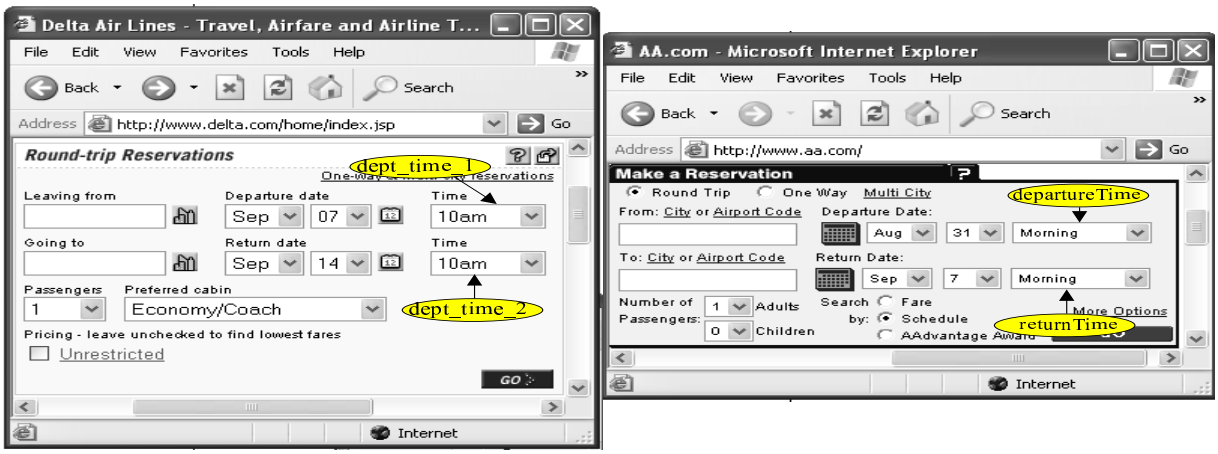

Figure 3: AA versus Delta

Example 4 is a simplification of a scenario we have identified in the Delta airline reservation system (see Figure 3). The form contains two time fields, one for departure and the other for return. Due to bad design (or designer's error), the departure time entry is named `dept_time_1` while return time is named `dept_time_2`. Both terms carry an identical label, `Time`, since the context can be easily determined (by a human observer of course) from the positioning of the time entry with respect to the date entry. For the American Airlines reservation system (Figure 3), the two time fields of the latter were not labeled at all (counting on the proximity matching capabilities of an intelligent human being), and therefore were assigned, using composition by multiple term association, with the label `Departure Date` and `Return Date`. The fields were assigned the names `departureTime` and `returnTime`. Term matching would prefer matching both `Time(dept_time_1)` and `Time(dept_time_2)` of Delta with `Return Date(returnTime)` of American Airlines (note that 'dept' and 'departure' do not match, neither as words nor as substrings). Value matching cannot differentiate the four possible combinations. Using precedence matching, the two time entries were correctly mapped.

The complexity analysis of precedence matching is similar to that of composition matching. Precedence matching considers all $n \times m$ term combinations. For each it computes similarity twice

(matching the *precede* subontologies and the *succeed* subontologies). Therefore, when applying a $1:1$ matching, composition matching runs at $O\left(n^5\right)$ for ontologies of similar size. For $1:n$ matching, the algorithm runs at $O\left(n^4\right)$ for ontologies of similar size.

3.5 Computing pair-wise similarities

Given two terms, v and u, and four similarity measures, $\mu^T_{v,u}$, $\mu^V_{v,u}$, $\mu^C_{v,u}$, and $\mu^P_{v,u}$, we can compute the overall similarity of v and u as the weighted average

$$\mu_{v_i,u_j} = \begin{cases} \frac{\tilde{\omega}^T \mu^T_{v,u} + \tilde{\omega}^V \mu^V_{v,u} + \tilde{\omega}^C \mu^C_{v,u} + \tilde{\omega}^P \mu^P_{v,u}}{\tilde{\omega}^T + \tilde{\omega}^V + \tilde{\omega}^C + \tilde{\omega}^P} & \tilde{\omega}^T + \tilde{\omega}^V + \tilde{\omega}^C + \tilde{\omega}^P > 0 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $\tilde{\omega}^T$, $\tilde{\omega}^V$, $\tilde{\omega}^C$, and $\tilde{\omega}^P$ are computed from the set $\{\omega^T, \omega^V, \omega^C, \omega^P\}$ of positive weights that sum to unity, as follows:

$$\tilde{\omega}^X = \begin{cases} \omega^X & \mu^X_{v_i,u_j} > t^X \\ 0 & \text{otherwise} \end{cases} \quad , X \in \{T, V, C, P\}$$

t^X (for $X \in \{T, V, C, P\}$) is the assigned threshold for the specific similarity measure. In addition, the following constraints are applied:

- As noted in Section 3.2, $\omega^V = 0$ whenever the terms are not constrained. This is the case whenever both terms are not atomic or whenever both terms are associated with a text field.
- Due to many false negatives and false positives in computing $\mu^C_{v,u}$ and $\mu^P_{v,u}$, we consider term and value similarity measures to be primary measures. That is, we do not move on to compute $\mu^C_{v,u}$ and $\mu^P_{v,u}$ for terms that do not pass term and value thresholds. This constraint also assists in reducing the overall complexity of the matching process, since linguistic similarity is much cheaper to compute than composition and precedence similarity.

In Section 3.3 and Section 3.4 we have computed the similarity of "subontologies" using linguistic similarity matching only. This computation can be easily derived from Formula 6 by setting $\omega^C = \omega^P = 0$.

4 Experiments

We are now ready to report our experiences with schema matching. Our experiments are aimed at measuring the effectiveness of the proposed algorithms for automatic ontology matching. We use OntoBuilder to measure effectiveness. In what follows, we first present the experimental methodology and then the experimental results.

4.1 Experimental methodology

4.1.1 OntoBuilder

We developed a tool that extracts ontologies from Web applications and maps ontologies to answer user queries against data sources in the same domain. The input to the system is an HTML page representing the Web site main page. The whole process is divided into four phases. In phase 1, the HTML page is parsed using a library for HTML/XML documents, and produces a DOM tree representing the page. The DOM tree is then used to identify all the form elements and their labels in phase 2. In phase 3, the system produces an initial version of the target and candidate ontologies. Later, in phase 4, the ontologies are matched to produce a mapping using the algorithms presented in Section 3.

The label identification algorithm implemented in OntoBuilder is very powerful, being able to identify a high percentage of elements and their associated labels. On average, the label identification algorithm achieves more than 90% effectiveness. These results do not include hidden fields, submits, resets and buttons in general, and images. Although these elements are used in the ontology extraction, they usually do not have an associated label (*e.g.*, hidden fields are not even shown to the user).

OntoBuilder supports an array of matching and filtering algorithms, as discussed above. Additional algorithms can be implemented and added to the tool as plug-ins. All algorithms are extensions of an abstract algorithm interface. The interface describes the signature (methods and functions) that matching algorithms must implement in order to be used in the tool. Algorithm parameters (such as weights) are specified using an XML configuration file which can be edited using a user-friendly interface. OntoBuilder was developed using Java, and runs under the Java 2 JDK version 1.4 or greater. OntoBuilder also provides an applet version with the same features of the standalone version and the added functionality that allows users to access and use it within a Web client.

OntoBuilder is available at `http://ie.technion.ac.il/OntoBuilder`.

4.1.2 Data

For our experiments, we selected 104 Web forms from 14 different domains, namely flight reservations, book stores, car rentals, cosmetics, dating and matchmaking, accommodation portals, hotel Web sites, job hunting, moving, news, search engines, ticket booths, vacation timesharing, and Web mail. We matched the Web forms in pairs, where pairs were taken from the same domain. In our experiments, we used $1 : n$ mapping. This means that while each concept from the candidate ontology could be paired to no more than one concept from the target ontology, several concepts from the target ontology could be mapped to the same candidate concept. $1 : n$ mappings result in asymmetric mappings. Therefore, for each pair we ran two sets of experiments, exchanging the roles of candidate and target ontologies.

All experiments were conducted using OntoBuilder (see Section **??**). We tested six different variations of OntoBuilder algorithms, namely term, value, linguistic (term+value), composition, precedence, and combined (linguistic+composition+precedence). In most cases, we ran all six algorithms on each Web form pair, varying the threshold level from 0% to 85% for each experiment. A 0% threshold obliges an algorithm to find a match for all concepts in the target ontology, no matter how poor the match is. For example, in the Hertz/Alamo case, `Country: (DROP_OFF_COUNTRY)` was incorrectly matched at a threshold of 0% with `(ADDITIONALDRIVERSCOUNT)` using term matching that managed to find a random match (`COUNT`). The number of pairs to be tested for each algorithm ranged from 90 to 104.

For each Web pair, a human assessor determined whether a mapping was possible, and if so what that mapping was.

4.1.3 Evaluation methodology

We evaluate the performance of the various algorithms using three metrics, namely recall, precision, and error. Recall is computed as the ratio between the number of matched terms and the overall number of identified terms. That is, given n attributes in the target Web resource dictionary, of which $m \leq n$ are matched, recall is computed as

$$R = \frac{m}{n}$$

Recall is affected by the chosen threshold, where a threshold of 0% forces all attributes to be matched, yielding a recall of 100%. Recall typically falls as the threshold rises.

Precision is computed as the ratio between the number of correctly matched terms and the total number of matched terms. In our experiments, we used two variants of precision. The first variant takes into account all terms of both ontologies, whether or not they can be matched, as determined by the human assessor. The second variant considers only those attributes that can be matched in the assessor's judgment. Formally, Let $V = \{v_1, v_2, ..., v_n\}$ be a target Web resource dictionary and let U be a candidate Web resource dictionary. U partitions V into two subsets V_1 and V_2, such that V_1 is the set of all matchable terms and V_2 contains all those terms that cannot be matched with any term in U. Let M be a set of cardinality m, representing the set of all attributes in V that were matched by the algorithm. *General precision* (P_G) is computed as

$$P_G = \frac{|V_1 \cap M|}{m}$$

and *precision with respect to target* (P_T) is computed as

$$P_T = \frac{|V_1 \cap M|}{|V_1|}$$

P_G measures, among other things, the ability of an algorithm to combat domain heterogeneity, while P_T ignores domain heterogeneity, focusing on terminology and structural heterogeneity only.

Typically, a rise in precision comes at the expense of recall. Therefore, the selection of a threshold must aim at balancing the number of matched terms with the number of **correctly**

matched terms. In our experiments, we also used a combined measure as suggested in [45]. For a precision value P, a recall value R, and an importance measure b, the combined measure E is computed to be

$$E = 1 - \frac{(1+b^2)PR}{b^2P + R}$$

E represents the amount of error in the model. Therefore, a low E value indicates a higher combined value of P and R. b balances the importance of P and R, where for $b = 0.5$ (the value of choice in our experiments), precision bears more weight. It is worth noting that E is a generalization of the more well known *F-value*. *F-value* is defined to be

$$F = \frac{2PR}{P + R}$$

Therefore, by setting $b = 1$, we get $E = 1 - F$. Like the measure of precision, the error measure comes in two flavors, E_G and E_T.

4.2 Experimental results

4.2.1 Recall

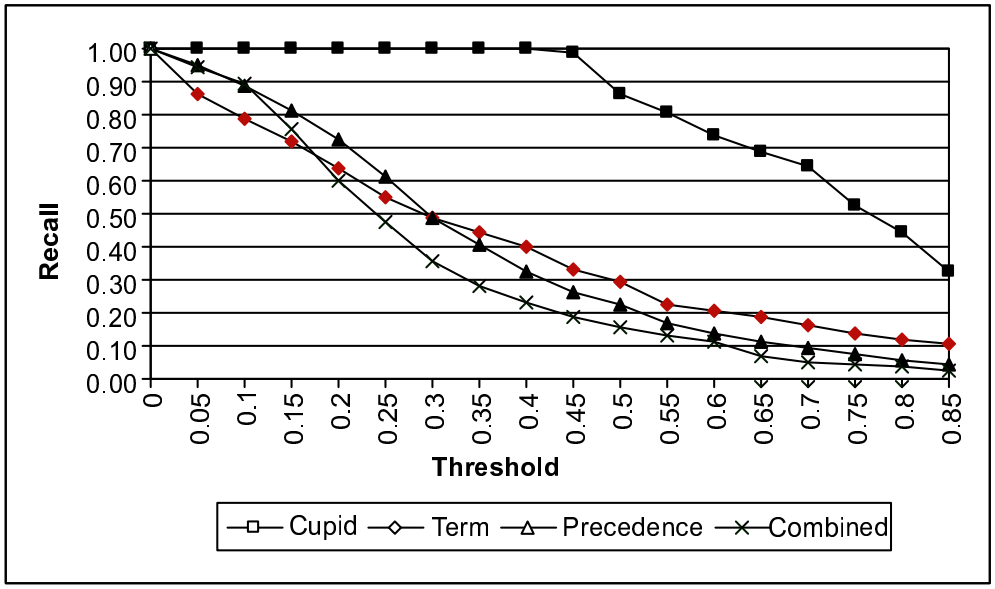

Figure 4: Recall vs. Threshold

We examine the recall of the six algorithms by using threshold levels of 0%, 5%, 10%,..., 85%. Figure 4 provides a representative sample of three of the seven algorithms. As expected, the higher the threshold, the lower the recall measure. All algorithms show a graceful decline. Clearly, different algorithms require different thresholds to obtain a reasonable level of recall. To achieve an 80% recall, the combined algorithm needs to maintain a somewhat low threshold of about 13%. This observation should be kept in mind when performing precision analysis (see below).

4.2.2 Precision

In Section 4.1.3 we differentiated between two types of precision. P_G is computed out of all terms, while P_T is computed out of matchable terms only. In this section we examine each measure separately, then compare them.

Figure 5:

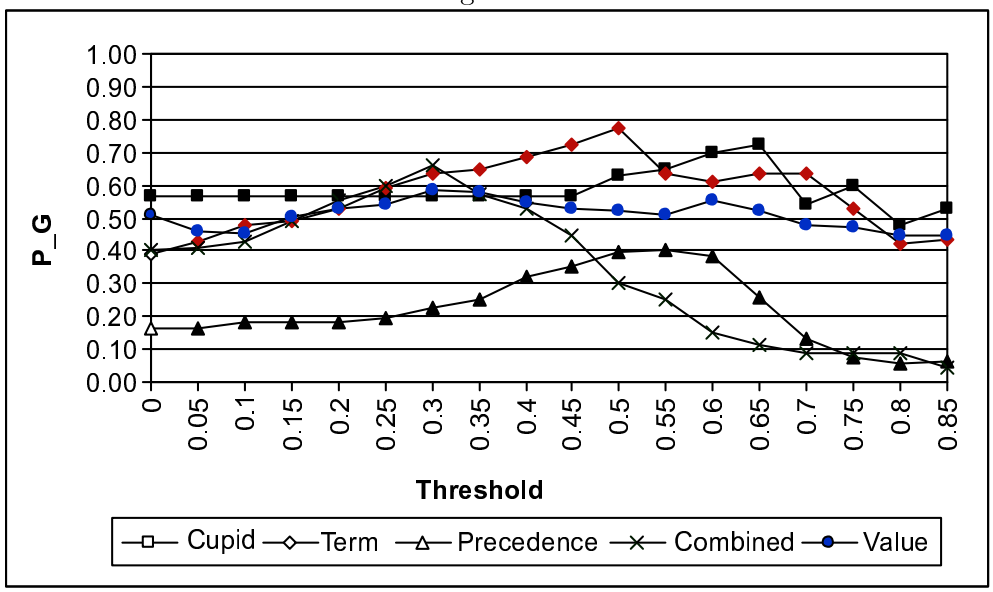

Figure 6: P_G vs. Threshold

Figure 6 provides a pictorial comparison of four algorithms with respect to P_G. Term outperforms the other algorithms for almost all thresholds. These results indicate that in the types of ontologies we have extracted from the Web, syntactical analysis carries more weight than semantic analysis (such as structural information). As further evidence for this conclusion, one may consider the performance of the Precedence algorithm, which measures significantly lower in precision than Term. Precedence produces many false positive errors, suggesting that such an algorithm is put to better use in refuting possible matches than in supporting them. It is also interesting to note the reasonable performance of the Value algorithm. What is not evident from this graph is that Value performance varies much more than that of other algorithms. Clearly, for ontologies with many different data types Value has good prediction capabilities (much better than Term), while for onotologies in which many terms share the same domain, Value will find it much harder to predict correct mappings. Finally, the performance of the Combined algorithm is affected by the performance of the four algorithms that construct it, namely Term, Value, Composition, and Precedence. Up to a threshold of 0.3 its performance is similar to that of the Term algorithm. For higher thresholds, its performance rapidly deteriorates under the impact of the Composition and Precedence algorithms.

The analysis of P_T is based on Figure 8. The overall patterns we have noticed in Figure 6 repeat here as well, only with a higher precision level. The dominance of Term is more visible here,

Figure 7:

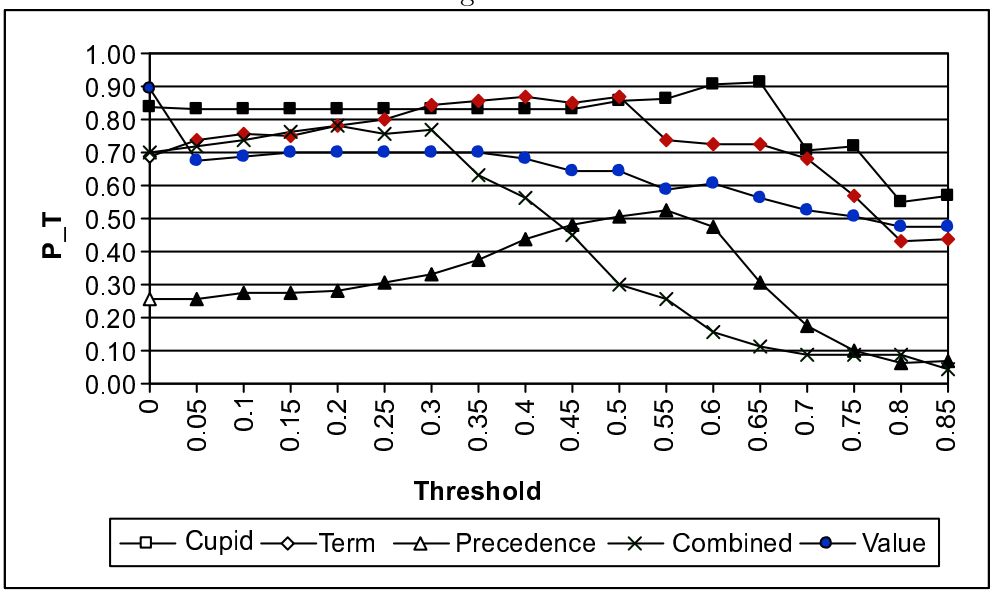

Figure 8: P_T vs. Threshold

with better performance for thresholds in $[0.05, 0.75]$. An interesting phenomenon is the ability of the Value algorithm to outperform all other algorithms for a 0 threshold (which means a recall of 100%), with an average precision of 90%.

4.2.3 Error

We next turn our attention to the Error measure. Recall that the Error measure provides a combined measure of recall and precision, given a specific ratio of importance between them. In our case, we have set $b = 0.5$, giving more importance to precision than to recall. The lower the Error measure, the better the combined impact of an algorithm. Figure 10 presents the change in error levels (using P_G) with a rising or falling threshold. For low thresholds (up to 0.3), Term performs slightly better. For higher threshold values, the combined algorithm performs better than its counterparts.

Figure 12 provides a similar result for the E_T measure. Term outperforms the other algorithms for a threshold of up to 0.4. For higher thresholds, the combined algorithm outperforms other algorithms, with the Precedence algorithm achieving similar error levels for thresholds of 0.8 and 0.85.

Figure 9:

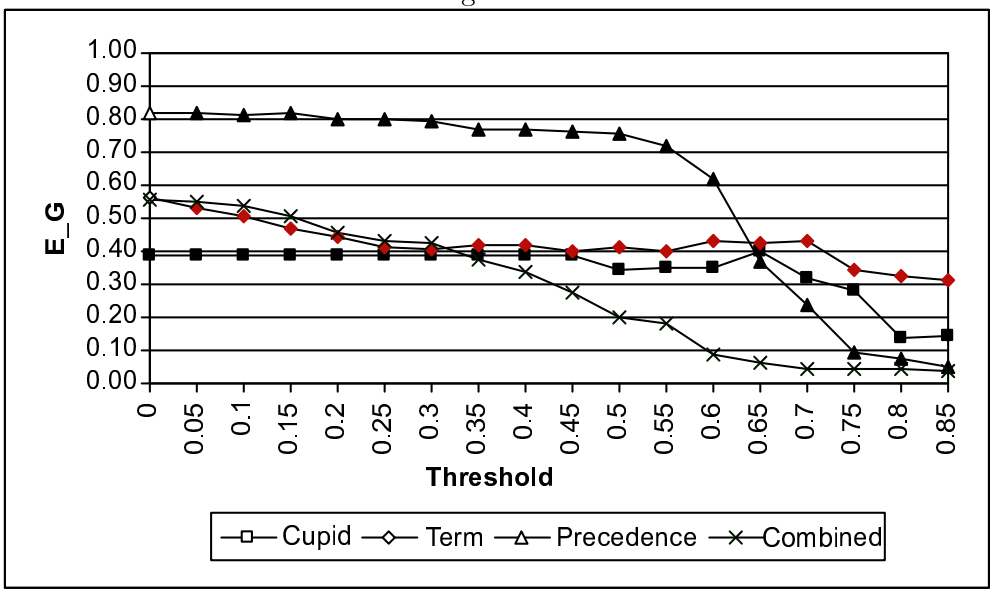

Figure 10: E_G vs. Threshold

5 Concluding remarks

In this paper, we have proposed the use of application semantics to enhance the process of ontology matching. Application semantics involves those elements of business reasoning that affect the way in which concepts are presented to users, for example via their layout. In particular, we have introduced the *precedence* ontological construct, in which temporal constraints determine the sequence of concepts presented to the user. While the paper has suggested the extraction of ontologies from HTML forms, we consider the use of ontologies to be essential for the broad area of Web search. Current search engines (in particular Google) have applied IR techniques in matching documents with user queries. We believe that the addition of structures such as composition and precedence to search engines, whenever suitable, would enhance the precision of the search process. We leave this as an open research question. In particular, we will explore the use of additional ontology structures to improve the effectiveness of the matching process.

It is our conjecture that using application semantics as a means for semantic reconciliation can be generalized beyond its application to HTML Web forms. For example, the relational model has little ability to represent application semantic means such as precedence. However, many relational databases are interfaced nowadays through the use of HTML forms, for which we have shown in this paper that precedence increases the success of semantic reconciliation. Also, analysis of typical queries for a given application reveals information regarding the typical use of concepts, which can be further utilized in the semantic reconciliation process. We plan on investigating the methods illustrated above in future research.

While precedence has proven itself in certain instances, a good algorithm is still needed to

Figure 11:

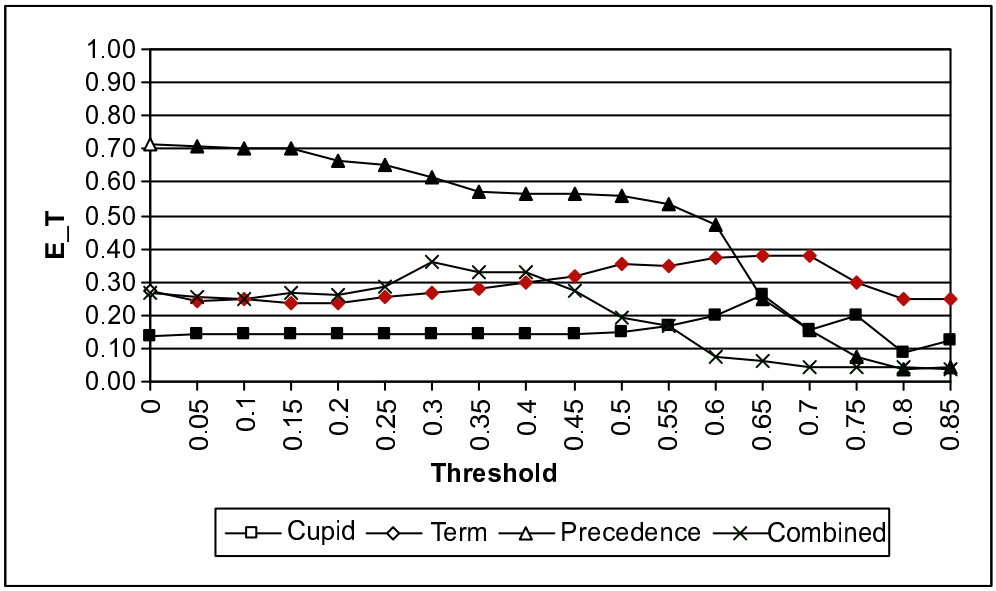

Figure 12: E_T vs. Threshold

extract this knowledge and put it to use, as our experiments show. The conceptual framework we provide, however, opens the door to more application-semantic concepts to be introduced and used in the ontology matching process. Another observation derived from our experiments is that combining matching algorithms via weighted average (or sum) methods may be counterproductive. In ... we pursue other alternatives to this approach, based on preference theory.

We aim at continually improving the proposed algorithms. For example, the use of a linear algorithm for finding the maximal substrings and superstrings of two given strings was suggested in the context of bioinformatics [47]. Embedding a variation of this algorithm in our system may reduce the complexity of string matching. Finally, we intend to research in depth the problem of complex query rewriting in a heterogeneous schemata setting, using data types identification and domain normalization. The method proposed in Section 3.2 serves as a promising starting point, yet a more thorough methodology is yet to be developed.

Research complementing the present paper provides sufficient conditions for matching algorithms to identify exact mappings, as conceived by an expert. This work is reported in [3, 23].

Acknowledgement

Our thanks is given to Louiqa Raschid and AnHai Doan for useful discussions. This research was partially supported by the Fund for the Promotion of Research at the Technion (191-496) and by the Fund of the Vice President for Research at the Technion (191-507). Giovanni Modica and Hasan Jamil's research was partially supported by National Science Foundation EPSCoR Grants

(EPS-0082979 and EPS-0132618), a USDA-ARS Cooperative Agreement grant (CRIS-6406-21220-005-15S) and a Southwest Mississippi Resource Conservation & Development Grant (01050412). Avigdor Gal's research was partially supported by the IBM Faculty Award (2002). We thank Ido Peled, Haggai Roitman, and the class of "Information Systems and Knowledge Engineering Seminar," Fall Semester, 2002, for their assistance in collecting and analyzing the data.

References

[1] *Concise Oxford Dictionary.* Oxford Univ. Press, 8 edition, 1991.

[2] J. Aitchison, A. Gilchrist, and D. Bawden. *Thesaurus construction and use: a practical manual.* Aslib, London, third edition, 1997.

[3] A. Anaby-Tavor, A. Gal, and A. Trombetta. Evaluating matching algorithms: the monotonicity principle. In S. Kambhampati and Craig A. Knoblock, editors, *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web*, pages 47–52, Acapulco, Mexico, August 2003.

[4] Y. Arens, C.A. Knoblock, and W. Shen. Query reformulation for dynamic information integration. In G. Wiederhold, editor, *Intelligent Integration of Information*, pages 11–42. Kluwer Academic Publishers, 1996.

[5] S. Bergamaschi, S. Castano, M. Vincini, and D. Beneventano. Semantic integration of heterogeneous information sources. *Data & Knowledge Engineering*, 36(3), 2001.

[6] J. Berlin and A. Motro. Autoplex: Automated discovery of content for virtual databases. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Cooperative Information Systems, 9th International Conference, CoopIS 2001, Trento, Italy, September 5-7, 2001, Proceedings*, volume 2172 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2001.

[7] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. *Scientific American*, May 2001.

[8] A. Borgida. Knowledge representation, semantic data modelling: What's the difference? In *Proceedings of the 9th International Conference on Entity-Relationship Approach (ER'90)*, pages 1–2, Lausanne, Switzerland, 1990.

[9] M. Brodie. The grand challenge in information technology and the illusion of validity. Keynote lecture at the International Federated Conference 'On the Move to Meaningful Internet Systems and Ubiquitous Computing', 2002.

[10] M. Bunge. *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World.* D. Reidel Publishing Co., Inc., New York, NY, 1977.

[11] M. Bunge. *Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems.* D. Reidel Publishing Co., Inc., New York, NY, 1979.

[12] S. Castano, V. De Antonellis, M.G. Fugini, and B. Pernici. Conceptual schema analysis: Techniques and applications. *ACM Transactions on Database Systems (TODS)*, 23(3):286–332, 1998.

[13] B. Convent. Unsolvable problems related to the view integration approach. In *Proceedings of the International Conference on Database Theory (ICDT)*, Rome, Italy, September 1986. In *Computer Science*, Vol. 243, G. Goos and J. Hartmanis, Eds. Springer-Verlag, New York, pp. 141-156.

[14] A. Doan, P. Domingos, and A.Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In Walid G. Aref, editor, *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, Santa Barbara, California, May 2001. ACM Press.

[15] P. Domingos and M. Pazzani. Conditions for the optimality of the simple bayesian classifier. In *Proc. ICML*, pages 105–112, 1996.

[16] F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logic. In G. Brewka, editor, *Principles on Knowledge Representation, Studies in Logic, Languages and Information*, pages 193–238. CSLI Publications, 1996.

[17] C. Fox. Lexical analysis and stoplists. In W.B. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*, pages 102–130. Prentice Hall, Englewood Cliffs, NJ 07632, 1992.

[18] W. Francis and H. Kucera, editors. *Frequency Analysis of English Usage*. Houghton Mifflin, New York, 1982.

[19] N. Fridman Noy and M.A. Musen. Smart: Automated support for ontology merging and alignment. In *Proceedings of the Twelfth Banff Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Alberta, 1999.

[20] N. Fridman Noy and M.A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 450–455, Austin, TX, 2000.

[21] A. Gal. Semantic interoperability in information services: Experiencing with CoopWARE. *SIGMOD Record*, 28(1):68–75, 1999.

[22] A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, 2004. to appear.

[23] A. Gal, A. Trombetta, A. Anaby-Tavor, and D. Montesi. A model for schema integration in heterogeneous databases. In *Proceedings of the 7th International Database Engineering and Application Symposium*, Hong Kong, China, July 2003.

[24] Z. Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys*, 18(1):23–38, March 1986.

[25] R.C. Gonzales and M.G. Thomanson. *Syntactic Pattern Recognition – An Introduction*. Addison-Wesley, Reading, Massachusetts, 1978.

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

[27] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 51–61. ACM Press, 1997.

[28] M. Jarrar and R. Meersman. Formal ontology engineering in the DOGMA approach. In *Proceedings International Federated Conference 'On the Move to Meaningful Internet Systems and Ubiquitous Computing'*, pages 1238–1254, October 2002.

[29] J. Kahng and D. McLeod. Dynamic classification ontologies for discovery in cooperative federated databases. In *Proceedings of the First IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, pages 26–35, Brussels, Belgium, June 1996.

[30] M. Kifer, G. Lausen, and J. Wu. Logical foundation of object-oriented and frame-based languages. *Journal of the ACM*, 42, 1995.

[31] P. Langley, W. Iba, and K. Thompson. An analysis of bayesian classifiers. In *Proceedings of the Tenth National Conference on AI*, pages 223–228, 1992.

[32] I.V. Levenstein. Binary codes capable of correcting deletions, insertions, and revrsals. *Cybernetics and Control Theory*, 10(8):707–710, 1966.

[33] J. Madhavan, P.A. Bernstein, P. Domingos, and A.Y. Halevy. Representing and reasoning about mappings between domain models. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, pages 80–86, 2002.

[34] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the International conference on very Large Data Bases (VLDB)*, pages 49–58, Rome, Italy, September 2001.

[35] A. Maedche and S. Staab. Measuring similarity between ontologies. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW 2002)*, pages 251–263, Siguenza, Spain, October 2002.

[36] D.L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, 2000.

[37] R.J. Miller, M.A. Hernàndez, L.M. Haas, L.-L. Yan, C.T.H. Ho, R. Fagin, and L. Popa. The Clio project: Managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.

[38] G. Modica, A. Gal, and H. Jamil. The use of machine-generated ontologies in dynamic information seeking. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Cooperative Information Systems, 9th International Conference, CoopIS 2001, Trento, Italy, September 5-7, 2001, Proceedings*, volume 2172 of *Lecture Notes in Computer Science*, pages 433–448. Springer, 2001.

[39] A. Moulton, S.E. Madnick, and M. Siegel. Context mediation on Wall Street. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS'98)*, pages 271–279, New York City, New York, August 1998. IEEE-CS Press.

[40] M. Nadler and E. Smith. *Pattern Recognition Engineering*. John Wiley & Sons Inc., 1993.

[41] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In L.M. Haas and A. Tiwary, editors, *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, pages 295–306, Seattle, Washington, June 1998. ACM Press.

[42] A.M. Ouksel and C.F. Naiman. Coordinating context building in heterogeneous information systems. *Journal of Intelligent Information Systems (JIIS)*, 3(2):151–183, April 1994.

[43] L. Palopoli, L.G. Terracina, and D. Ursino. The system DIKE: Towards the semi-automatic synthesis of cooperative information systems and data warehouses. In *PADBIS-DASFAA*, pages 108–117, 2000.

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

[45] C.J. Van Rijsbergen, editor. *Information Retrieval*. Butterworths, London, 1979.

[46] Stuart Russell and Peter Norving. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey, 1995.

[47] Walter L. Ruzzo and Martin Tompa. A linear time algorithm for finding all maximal scoring subsequences. In Thomas Lengauer, Reinhard Schneider, Peer Bork, Douglas L. Brutlag, Janice I. Glasgow, Hans-Werner Mewes, and Ralf Zimmer, editors, *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 234–241, Heidelberg, Germany, 1999. AAAI.

[48] R. Schalkoff. *Pattern Recognition: Statistical, Structural, and Neural Approaches*. John Wiley & Sons Inc., 1992.

[49] P.L. Schuyler, W.T. Hole, and M.S. Tuttle. The UMLS (Unified Medical Language System) metathesaurus: representing different views of biomedical concepts. *Bulletin of the Medical Library Association*, 81:217–222, 1993.

[50] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[51] A.P. Sheth, S.K. Gala, and S.B. Navathe. On automatic reasoning for schema integration. *Intenational Journal on Intelligent Cooperative Information Systems (IJICIS)*, 2(1):23–50, June 1993.

[52] P. Simon. *Parts: A Study in Ontology*. Clarendon Press, New York, NY, 1987.

[53] D. Soergel. *Organizing information : principles of data base and retrieval systems*. Academic Press, Orlando, FA, 1985.

[54] P. Spyns, R. Meersman, and M. Jarrar. Data modelling versus ontology engineering. *ACM SIGMOD Record*, 31(4), 2002.

[55] A. Varzi. On the boundary between mereology and topology. In R. Casati, B. Smith, and G. White, editors, *Philosophy and the Cognitive Sciences*. Hoelder-Pichler-Tempsky, Vienna, Austria, 1994.

[56] B.C. Vickery. *Faceted classification schemes*. Graduate School of Library Service, Rutgers, the State University, New Brunswick, N.J., 1966.