Merging Hierarchies Using Object Placement

Kai Zhao
NEC Laboratories China
1 Zhongguancun East Road
Beijing, China
zhaokai
@research.nec.com.cn

Robert Ikeda
Computer Science
Department
Stanford University
Stanford, CA 94305-9040
rmikeda@cs.stanford.edu

Hector Garcia-Molina
Computer Science
Department
Stanford University
Stanford, CA 94305-9040
hector@cs.stanford.edu

ABSTRACT

The main challenge in integrating two hierarchies is determining the correspondence between the nodes and edges of each hierarchy. Traditionally, the correspondence is determined by comparing the text associated with each edge. In this paper we instead use the placement of objects present in both hierarchies to infer how the hierarchies relate. We present an algorithm that, given a hierarchy with known *facets* (label-value pairs that define what objects are placed under an edge), determines feasible facets for a second hierarchy, based on shared objects. We present a second algorithm that uses the discovered facets to copy objects that only appear in the second hierarchy into the first one, to arrive at a more comprehensive hierarchy. We present experimental results that illustrate the usefulness of our approach, and how noise in our data can be handled.

1. INTRODUCTION

Objects are often organized in a hierarchy to help in managing or browsing them. For example, products in a store can be divided by type (electronics, clothes, books, ...) and then by brand (Sony, Epson, Dockers, ...). Web pages at a site can also be placed in a hierarchy. For instance, a French tourist site may have categories cities, history, hotels, tours; within the cities category we have pages divided by city, and then by events, maps, restaurants.

In this paper we study the problem of hierarchy integration, in particular, how to combine two related hierarchies into one, more comprehensive one. The need to integrate arises in many situations where the objects come from different systems. In our product hierarchy example above, we may want to provide a comparison shopping service that offers products from two stores; in our tourism example, we may want to build a meta-web-site that combines the resources of two or more French tourism sites.

To simplify the problem, we study how to merge one hierarchy into a second known base hierarchy, by copying references to objects into the base hierarchy, and perhaps by adding some categories into the base hierarchy. To illustrate this process, and the approach we take, consider the product hierarchies shown in Figure 1. Hierarchy H_1 is the base hierarchy. Each edge in the hierarchy is annotated with a *facet* that describes the objects placed along that edge. For example, the facet "brand: Sony" indicates that products made by Sony will be found as descendants of this edge. We call "brand" the *label* of the facet, and "Sony" the *value*.

Products are placed at the nodes of the hierarchy, according to their characteristics. For example, the object called a in Figure 1 is a Sony camera that is on sale, i.e., it has facets "brand: Sony", "category: camera" and "sale: yes" (we abbreviate sales to "s" and yes to "y" in the figure). Think of a as the URL that describes this camera at the Sony web site. Notice that objects may be placed at non-leaf nodes, for instance, object d. All we know from d's placement is that it is a camera, but we do not know its brand nor whether it is on sale.

Hierarchy H_2 in Figure 1 is the one we wish to integrate into H_1. Some of the objects in H_2 also appear in H_1, but not all. We assume that by applying an entity resolution solution [3], we have matched objects across the hierarchies and stored the matches in a table. Our goal is to move the references to the new objects like x and y from H_2 into the appropriate places in H_1. We have labeled the edges in H_2 with question marks, to indicate that these facets may not match up exactly to the facets in H_1.

Of course, if the H_2 facets can be matched up exactly with the H_1 ones, then our matching problem is solved. But here we wish to study the problem where the facets do not match exactly. For instance, the facet "brand: Panasonic" may be represented by "brand: Matsushita" in the other hierarchy. (Matsushita markets their products under the name Panasonic in some countries.) There could also be typos or different wordings in the facets, for instance one tourism hierarchy may refer to "hotels" while the other uses the term "lodging".

When the facets do not match exactly, there are two techniques available. The first, and more traditional one, is to compute the textual similarity between facets in H_1 and H_2 to try to discover the correspondence. One may also "propagate" similarities up or down the hierarchy. For example, say edge e_1 in H_1 has two child edges, $e_{1,1}$ and $e_{1,2}$, while edge e_2 in H_2 has children $e_{2,1}$ and $e_{2,2}$. Say that the facet of $e_{1,1}$ is textually very similar to that of $e_{2,1}$, while the facet of $e_{1,2}$ is similar to that of $e_{2,2}$. Then we may "boost" the similarity of e_1 to e_2 beyond what is suggested by their textual labels [13].

The second alternative is to study the objects in the hierarchies to find correspondences. (Reference [10] also exploits objects, but in a different way. See Section 8.) For example, if we look at the placement of objects b and c in both hierarchies, we may be able to infer that facet ?2 is really "brand: Epson". We have to assume some properties about the hierarchies in order to reach this conclusion, and the properties will be discussed in detail later on. But for now, we can argue that the facets that were used in H_1 for b should also appear in H_2, hence the facet on edge ?2 should

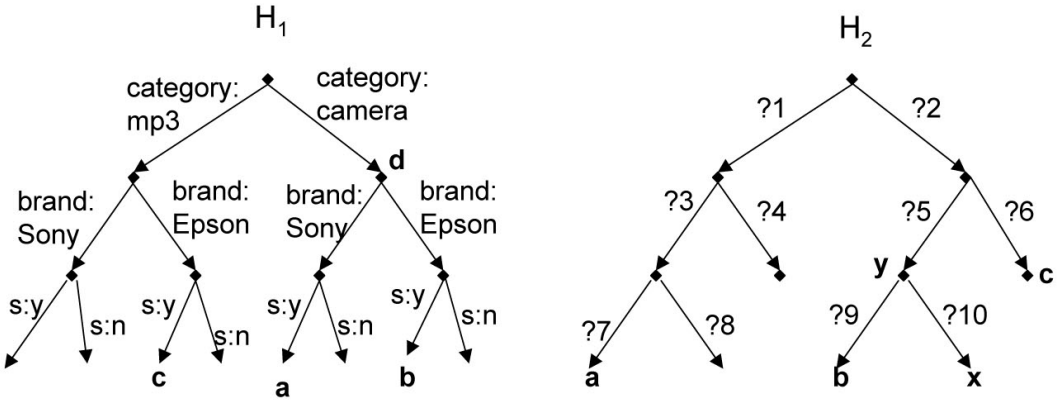

Figure 1: Motivating example of hierarchies to be merged.

either be "category: camera" or "brand: Epson" or "sale: yes". However, since c is an mp3 player, and shares ?2, then ?2 cannot represent a decision based on category. Similarly, since a is also an object on sale, but is on a branch opposite ?2, then ?2 cannot address whether the item is on sale or not. (Here we are assuming that sibling branches on H_2 use the same label.) Thus, we infer that edge ?2 must have the facet "brand: Epson".

Through similar reasoning, we conclude that edge ?5 must be "category: camera" and edge ?6 "category: mp3". These decisions leave "sale: yes" as the only available choice for ?9. Edge ?10 must use the "sale" label, and if we assume there are only two values for the sale label, then ?10 must be "sale: no".

At this point, we know the facets of some of the H_2 objects that are not in H_1. For example, because of x's placement in H_2, we know it must have facets "brand: Epson", "category: camera", and "sale: no". Thus we can move object x into H_1, at the rightmost leaf node. Similarly, we know that object y in H_2 is an Epson camera, so we can move it into H_1 at the parent of the node where b is located. (We do not know if y is on sale or not.)

In other cases we may have insufficient information to move objects just based on object placement. However, the object placement can narrow down the choices we need to make in comparing labels. For example, consider edge ?3 in H_2. Because we know ?1 is "brand: Sony" and because object a is reachable through edge ?3, there are only two choices for this edge: "category: camera" or "sale: yes". Thus, we only have to decide if the facet of ?3 is closer to "category: camera" or to "sale: yes". If we had not done the object analysis, we would need to compare the facet of ?3 to all possible labels and values. Thus, analyzing object placement may lead to fewer choices, and in the end, more accurate mappings.

Once we select the facet for ?3 (or for ?7), we can use the choice to eliminate other choices up or down H_2. For instance, if ?3 is determined to be "category: camera", then ?7 must be "sale: yes". We could also then infer the facets on ?4 and ?8 if we assume that there are only two available values for these labels. And once we identify more facets in H_2, we may be able to move additional objects to H_1.

As mentioned earlier, our strategy is based on certain properties, and if the properties do not hold, we may make "errors." For example, one of our properties (Section 3) states that objects in a hierarchy are properly classified. Suppose instead that some objects are misclassified, in particular, say object a in Figure 1 is not really a Sony camera but is an Epson MP3 player. Then clearly the reasoning we followed above will lead to incorrect conclusions. One of our goals will be to study how many "errors" a solution can have if the number of violations of the properties is "limited."

In summary, in this paper we study how we can compare object placement in two hierarchies, one base hierarchy and another new one, to infer how objects are categorized in the new hierarchy, and where they can be moved into the base hierarchy. We start by defining our model and properties (Sections 2, 3, 4), and then present our algorithms for matching hierarchies and copying objects (Sections 5, 6). In Section 7 we present our experimental results, and in Section 8 we discuss related work.

2. MODEL

In this section we introduce notation that will help us define our problem more precisely.

- A facet f is a pair g:v where g is the label of f and v is the value of f. As we discuss below, facets are used to categorize objects, as well as to describe how objects are placed in hierarchies. We use the symbol "*" when a value in a facet is unspecified. For example, g:$*$ refers to any facet with label g.
- We are given two hierarchies, a base hierarchy by H_1 and a new hierarchy by H_2. Each hierarchy H_i, contains nodes N_i and directed edges E_i. We assume (see Section 3 for a complete list of assumptions) that the hierarchies are trees. Each edge $e \in E_i$ is annotated with a single facet, $f(e)$. As we state formally in our assumptions, the objects placed under edge e must have facet $f(e)$. In our model the facet annotations of H_1 are known and the annotations of H_2 are unknown. When the identity of the hierarchy is understood or irrelevant, we will simply omit the hierarchy subscript.
- We may refer to hierarchy edges by their endpoints. For example, if edge e goes from node x to node y, we refer to e as x-y.

- Two edges, x-y and z-w are siblings if $x = z$.
- An edge x-y is a descendant of edge z-w in hierarchy H if node $x = w$ or x is a descendant of w in H.
- We denote the set of objects that are referenced in the hierarchies by O. Each object $o \in O$ has a set of facets $f(o)$ associated with it. Each object $o \in O$ may be associated with at most one node in hierarchy H_i. We call the node where o is placed $l_i(o)$. If an object o is not placed in hierarchy H_i, then $l_i(o)$ is null. The hope is that there will be some non-trivial number of objects placed in both H_1 and H_2, so we can discover how H_2 is structured based on those common objects.
- Given a node x, we refer to the set of edges on the path from root r_i to x as $\mathcal{P}_i(x)$, If $l_i(o)$ is not null, then the path associated with object o in H_i is $\mathcal{P}_i(o) = \mathcal{P}_i(l_i(o))$. If $l_i(o)$ is null, then $\mathcal{P}_i(o)$ is the empty set.

To illustrate our notation, let us return to Figure 1. In this case, the labels are "category", "brand", and "sale". The objects in H_1 are a, b, c, d. The root of H_1 has two edges, call them e_1 and e_2. Their facets are $f(e_1)$ = "category: mp3" and $f(e_2)$ = "category: camera". Call n_b the node at which object b is placed in H_1. Then $\mathcal{P}(n_b)$ contains the edges with facets "category: camera", "brand: Epson" and "s: y". Since $l(b)$ is node n_b, $\mathcal{P}(b) = \mathcal{P}(n_b)$. Note that we drop the hierarchy subscript when the hierarchy is understood.

3. PROPERTIES

As argued in the introduction, if we assume certain properties about facets and hierarchies, then we can improve our ability to discover the facets of H_2. In this section we go over these properties. We group the properties into three categories for reasons that will become clear later on. Note that these properties may or may not hold in a given application. After presenting the properties, we discuss what happens if they do not hold.

We call the first four properties *fundamental*:

- Known facet space. All the facets for the O objects, and in use in both H_1 and H_2, come from a known set of facets F. That is, if $o \in O$, then $f(o) \subseteq F$. Similarly, if e is an edge in E_1 or in E_2, then $f(e) \in F$.

  This property states that we know the space of facets from which we will choose the H_2 facets. Thus, even though we do not know the precise annotations on the H_2 edges, we at least know the options. The reason why this property is useful is fairly clear: if the objects in H_2 were classified according to facets unknown to us, then it would be hard to figure out the annotations for the H_2 edges.

  Given our facet set F and a facet $g{:}v \in F$, the facet complement of $g{:}v$, denoted by $C(g{:}v)$ is defined as the set of all facets with the same label but different values. That is, $C(g{:}v) = \{g'{:}v' \text{ such that } g' = g \text{ and } v' \neq v\}$.
- Object identities have been resolved. We have run an entity resolution process over the objects in both hierarchies, so we know which objects refer to the same real world entity. We then use the same identifier to refer to all objects that were mapped to the same real world entity. For instance, in Figure 1, what we call object a can be two separate but very similar web pages that have been determined to represent exactly the same Sony camera.
- Well-formed hierarchies. H_1 and H_2 are well-formed. We say that hierarchy H_i is well-formed if for every object $o \in O$, $\cup_{j \in \mathcal{P}_i(o)} f(j) \subseteq f(o)$. In other words, for every location where an object is placed, the annotations along the path from the root to the location must correspond to facets associated with the object.
- One value per label. For any object $o \in O$, if $g{:}v_1 \in f(o)$, then there is no other $g{:}v_2 \in f(o)$, $v_2 \neq v_1$. For instance, a product cannot have both "category: cell-phone" and "category: camera" facets. It could, however, have a "category: camera-phone" facet, distinct from the other two.

Next we present the two *simplification* properties:

- Tree hierarchies. Both our base hierarchy H_1, and our new hierarchy H_2, are trees. That is, each hierarchy H_i has a single root node r_i, and all nodes (except the root) have one parent (incoming) edge.
- At most one node per object. Each object $o \in O$ is associated with at most one node in hierarchy H_i, $l_i(o)$.

Finally, the *facet* properties refer to the facet annotations of a hierarchy.

- **P1.** Same label at a node. Let n be any node in a hierarchy, where $f(n\text{-}x) = g{:}v$. Then, for all edges n-y, $f(n\text{-}y) = g{:}*$. For example, if one branch in a hierarchy has the facet "category: camera" the other sibling branches for that node must have "category" labels.
- **P2.** No duplicate facets at a node. There are no two edges n-x, n-y such that $f(n\text{-}x) = f(n\text{-}y)$. Assumption P1 says sibling edges must have the same label, and this assumption says the values must be different.
- **P3.** No duplicate labels on a path. Let x be any node in H_i. There are no two edges e_1, $e_2 \in \mathcal{P}(x)$, $e_1 \neq e_2$, such that $f(e_1) = g{:}*$ and $f(e_2) = g{:}*$. For example, it would not make sense to have one edge with facet "color: red" only to have a descendant edge have the same "color: red" facet (we already know that all object below the first edge are red). It would also not make sense to have a descendant edge with facet "color: blue" since objects along this path must be red.

Each of the properties we have presented can be exploited to yield more information during the hierarchy matching process. The more properties hold, the more we can learn about the facets of H_2 and the better a job we can do in copying objects from H_2 to H_1. In the algorithms we present in this paper we assume all properties hold in order to show how the properties can be exploited.

However, our algorithms can still operate when some of the properties do not hold. In this case we can either be conservative or aggressive.

- *Conservative Approach.* We do not want to make any mistakes, so we change our algorithm to *not* exploit any properties known not to hold. Our algorithm can trivially be modified to ignore any of the facet properties P1, P2 and P3. For instance, if P2 does not hold, we skip the step of our algorithm that enforces P2 (Step 2). (See Section 5.3 for details.) If the simplification properties do not hold, we need to extend our algorithm to handle DAGs and multi-node objects. The extensions

are straightforward and not discussed here. If the fundamental properties do not hold, then we do not attempt to match the hierarchies, as it becomes impossible to discover the facets of H_2 with absolute certainty.

- *Aggressive Approach.* If we believe that there are a limited number of exceptions to a property (i.e., "noise"), then we can assume the property for our matching process, and hope the number of errors is also limited. In Section 7 we empirically evaluate the number of errors when a small number of objects are misplaced (violating the well-formed hierarchies assumption). We also consider a second scenario where there are many misplaced objects, and propose an additional filtering step to our algorithm. (Some of the misplacements may be due to ambiguous labels or values. For instance, a particular hotel may be classified as "budget" by one hierarchy and as "mid-range" by another.) This extra step attempts to discover misplaced objects before running our matching algorithm.

4. PROBLEM DEFINITION

We break down our hierarchy merging problem into two parts: identifying the H_2 facets, and moving objects from H_2 to H_1.

We will discuss the subproblem of moving objects in Section 6. In this section, we concentrate on the subproblem of identifying the H_2 facets (the matching problem).

For the matching problem, our goal is to guess the facets that were used in H_2 to organize the objects that were placed there. Ideally, we would like to discover exactly the one facet per edge that was actually used in H_2. This notion of a complete solution is captured by assignments, as defined next.

Assignments. An assignment for H_2 is a function $\delta(e)$ that assigns a *single* facet to each edge e in H_2.

Unfortunately, in many cases we will be unable to narrow down our choices to a single facet per edge. That is, for some edges we will be left with two or more facets that could very well have been used in H_2, and from the given evidence it is impossible to narrow down the selection to a single facet. Thus, our goal will be to produce a solution θ that represents the possibilities for the facets of H_2 given the object placements and our properties.

Solution. A solution θ gives a set of facets $\theta(e)$ for each edge e in H_2.

Covers. We say that solution θ covers assignment δ if for every edge e in H_2, $\delta(e) \in \theta(e)$.

Intuitively, we want the sets in θ to be as small as possible, since the smaller the sets the more we know about the H_2 actual facets. However, the sets in θ must be large enough to cover all assignments of facets to edges that do not violate any of our properties.

For instance, in the example in the introduction, we argued that we can narrow down the choices for edge ?2 to a single facet "brand: Epson". Thus, in this case $\theta(?2) =$ {"brand: Epson"}. For edge ?3 we were able to narrow down the choices to "category: camera" and "sale: yes". Thus, $\theta(?3) =$ {"category: camera", "sale: yes"}. This solution covers at least two assignments, δ_1 and δ_2, where $\delta_1(?2) =$ "brand: Epson" and $\delta_1(?3) =$ "category: camera"; and where $\delta_2(?2) =$ "brand: Epson" and $\delta_2(?3) =$ "sale: yes". Both of the assignments covered by our solution are good because they are plausible, i.e., if we used either assignment for H_2, we would be unable to prove that H_2 is not well-formed.

To formalize the notion that an assignment is "plausible" we first define two sets:

- Define $S_1(o)$ to be the set of known facets of object o as seen in H_1. That is, $S_1(o) = \cup_{j \in \mathcal{P}_1(o)}\{f(j)\}$.
- Define $S_2(o, \delta)$ to be the set of facets for o implied by assignment δ. That is, $S_2(o, \delta) = \cup_{j \in \mathcal{P}_2(o)}\{\delta(j)\}$.

Conflict-free. We say δ is conflict-free (what we called "plausible" above) if for every object o there are no conflicts between $S_1(o)$ and $S_2(o, \delta)$, that is, there are no two facets $g{:}v_1$ and $g{:}v_2$, with $v_1 \neq v_2$, and $g{:}v_1 \in S_1(o)$ and $g{:}v_2 \in S_2(o, \delta)$.

In the above example, δ_1 and δ_2 are conflict-free. To show an example of an assignment that is not conflict-free, let us have assignment δ_3 with $\delta_3(?3) =$ "category: mp3". Then we will have a conflict, for in H_1, object a has facet "category: camera" but that same object a in H_2 has facet "category: mp3".

It is easy to see that if an assignment δ_c is not conflict-free, then the hierarchy defined by δ_c is not well-formed. To see this, let o_c be the object that causes the conflict, i.e., there are facets $g{:}v_1$ and $g{:}v_2$, with $v_1 \neq v_2$, and $g{:}v_1 \in S_1(o_c)$ and $g{:}v_2 \in S_2(o_c, \delta_c)$. Then assuming H_1 is well-formed, we deduce $g{:}v_1 \in f(o_c)$. Similarly, if H_2 were well-formed, then $g{:}v_2 \in f(o_c)$. However, this is a contradiction because of our one-value-per-label property. Thus, H_2 is not well-formed with annotations δ_c.

We conclude that if H_2 is well-formed, then when considering possible assignments, we should only consider those assignments that are conflict-free. Thus, we should demand that our solution θ only cover assignments δ that are conflict-free. Furthermore, we would also like the assignments to satisfy our facet properties, i.e., P1, P2 and P3.

Structurally-Sound. An assignment δ is structurally-sound if H_2 would satisfy properties P1, P2 and P3 with annotations δ. Restating the properties using δ (instead of f which applies to H_1):

- **P1.** Let n be any node in H_2, where $\delta(n\text{-}x) = g{:}v$. Then, for all edges $n\text{-}y$, $\delta(n\text{-}y) = g{:}*$.
- **P2.** There are no two edges $n\text{-}x$, $n\text{-}y$ such that $\delta(n\text{-}x) = \delta(n\text{-}y)$.
- **P3.** Let x be any node in H_2. There are no two edges $e_1, e_2 \in \mathcal{P}(x)$, $e_1 \neq e_2$, such that $\delta(e_1) = g{:}*$ and $\delta(e_2) = g{:}*$.

Ideally, we would like to define a "correct" solution θ to be one where the set of assignments covered by θ and the set of conflict-free, structurally-sound assignments are identical sets. Unfortunately, such a definition is too strong. To illustrate, consider the example in Figure 2. Here the set of facets is $F = \{a{:}1, a{:}2, a{:}3, b{:}1, b{:}2\}$. Examining objects x and y we can arrive at solution θ_1 (Figure 2(c)), where we have narrowed down the choices for the three left edges to a single facet. However, for edge ?2 we have two choices, $a{:}2$ or $a{:}3$. Similarly, for edges ?5 and ?6 we have two choices.

The problem is that such a solution covers assignments that are not structurally-sound. For example, θ_1 covers an assignment where $\delta_n(?5) = b{:}1$ and $\delta_n(?6) = b{:}1$ (which violates property P2). We cannot eliminate the facet $b{:}1$ from one of the edges ?5 or ?6 in θ_1 because $b{:}1$ is a valid choice for

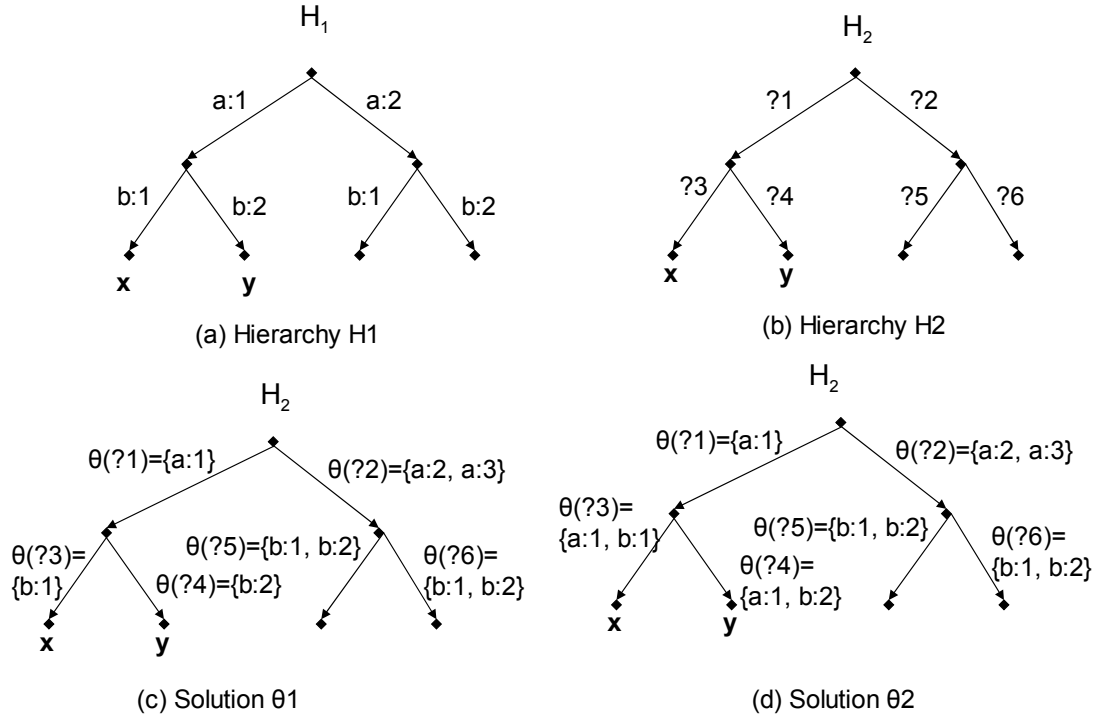

Figure 2: Example to illustrate meaning of correctness.

those two edges. Yet, given the machinery we have, we have no way of ruling out assignments like δ_n. One could define additional machinery to specify constraints among facets in a solution, but we feel such machinery is an overkill for our problem. We would rather work with a solution that covers some unsound assignments, since such unsound assignments can be identified easily once they are generated.

Thus, we arrive at the following definition for correctness:

Correctness. A solution θ is correct if

- For all structurally-sound, conflict-free assignments δ, θ covers δ;
- If θ covers an assignment δ, then δ is conflict-free (but may be structurally-unsound).

In our example, it is clear that θ_1 covers all "good" assignments, but as we illustrated, it can generate some unsound (yet conflict-free) assignments like δ_n.

There can be multiple correct solutions to a given matching problem. For instance, Figure 2(d) shows another correct solution θ_2 for the same problem. The only difference between θ_1 and θ_2 is on edges ?3 and ?4: $\theta_2(?3)$ and $\theta_2(?4)$ offer the extra choice of a:1. An assignment δ_e that assigns a:1 both to ?1 and to ?3 or ?4 is conflict-free but unsound. This assignment δ_e is not covered by θ_1 but is covered by θ_2.

If we had a choice between correct solutions θ_1 and θ_2, we would prefer θ_1 since it generates fewer unsound assignments than θ_2. The definition below captures the notion of a solution generating (covering) fewer assignments. The definition that follows, for a minimal solution, then specifies a solution that is correct and generates the fewest unsound assignments.

Order of Solutions. We say that $\theta_1 \leq \theta_2$ if for every edge e in H_2: $\theta_1(e) \subseteq \theta_2(e)$.

Minimal Solution. A solution θ_1 is minimal if it is correct and there is no other correct solution $\theta_2 \neq \theta_1$ such that $\theta_2 \leq \theta_1$.

5. MATCHING ALGORITHM

We are given hierarchies H_1 and H_2, facet annotations $f(H_1)$, a set of objects O, and a set of facets F. We want an algorithm that takes these inputs and returns a correct solution θ.

The algorithm discussed here takes an aggressive approach, as discussed in Section 3. When we present our experiments in Section 7, we will discuss what happens when the properties do not hold absolutely, i.e., when there is noise in our data.

Algorithm 1 presents our matching algorithm. Let us illustrate how Algorithm 1 works through an example. Suppose we have H_1 and H_2 as in Figure 1.

One of the properties from Section 3 states that we have a known facet space F. Thus, for this example, we will assume that

F = {"category: mp3", "category: camera", "brand: Sony", "brand: Epson", "s: y", "s: n"}.

In Step 1 of the algorithm, for each $e \in H_2$, we initialize $\theta(e) := F$. In Table 1 we show the annotations of each edge

Input: hierarchies H_1 and H_2, facet annotations $f(H_1)$, object set O, facet set F
Output: solution θ
Step 1: Use objects from H_1 to eliminate possibilities.
For each $e \in H_2$, initialize $\theta(e) := F$.
For each $o \in O$ do
Let $S_1(o) := \cup_{j \in \mathcal{P}_1(o)}\{f(j)\}$.
For each $f \in S_1(o)$ do
For each $e \in \mathcal{P}_2(o)$ do $\theta(e) := \theta(e) - C(f)$.
Step 2: Filter due to lone value for given label.
For each $n \in N_2$ do
If there is an edge n-x in E_2
such that g:$v \in \theta(n$-$x)$
and g:$v_1 \notin \theta(n$-$x)$ for any $v_1 \neq v$
then remove g:v from all other $\theta(n$-$*)$ in E_2.
Step 3: Filter due to missing label in candidate set.
For each $n \in N_2$ do
If there is an edge n-x in E_2 such that for all g:$*$
g:$* \notin \theta(n$-$x)$
then remove all g:$*$ from all $\theta(n$-$*)$ in E_2.
Step 4: Propagate when $\theta(e)$ has a single label.
For each $e \in E_2$ do
propagate(e).
Procedure propagate(e):
If there exists exactly one label g with g:$* \in \theta(e)$
then [for all descendant and ancestor edges d of e
[remove all g:$*$ from $\theta(d)$;
propagate(d);
for all sibling edges d_1 of d
[remove all g:$*$ from $\theta(d_1)$;
propagate(d_1);]]]

Algorithm 1: Matching algorithm for facets of H_2

at each point of the execution of the example. The facets have been abbreviated to their initials for compactness. The column labeled "Initialization" shows the edge annotations after the θ sets are initialized.

Then for each object o, we first trace the path from r_1 to the node in H_1 that contains o. By tracing this path in H_1, we can deduce which facets correspond to o. Then, we trace the path from r_2 to the node in H_2 that contains o. For each edge e along this path, and for each facet f deduced from the path in H_1, we subtract from $\theta(e)$ all facets that have the same label as f but different values.

As an example of this process, let us execute it for object a. Tracing its path in H_1, we find $S_1(a) = \{$"category: camera", "brand: Sony", "s: y"$\}$ or equal to $\{c$:c, b:S, s:$y\}$ using our abbreviations. We then trace the path from r_2 to the node in H_2 containing a, generating $\mathcal{P}(a) = \{?1, ?3, ?7\}$. For each edge in $\mathcal{P}(a)$, we must then subtract facets from $\theta(e)$. Since $C(c$:$c) = \{c$:$m\}$, $C(b$:$S) = \{b$:$E\}$, and $C(s$:$y) = \{s$:$n\}$, after processing object a in Step 1, we have
$\theta(?1) = \theta(?3) = \theta(?7) = F - C(c$:$c) - C(b$:$S) - C(s$:$y)$
$= \{c$:c, b:S, s:$y\}$.

That is, the H_2 edges that lead to object a must have θ values consistent with the facets of a in H_1. Table 1 shows the θ values for all edges after Step 1 completes.

Let us now apply Step 2. In this step, if there is an edge n-x in E_2 with only one facet g:v in $\theta(n$-$x)$ for a given label g, then we remove the facet g:v from all siblings $\theta(n$-$y)$, where $y \neq x$. As an example of this step, consider edge ?3. Note that b:$S \in \theta(?3)$ but there is no other facet b:$v \in \theta(?3)$. Thus, the sibling edges cannot refer to Sony products. So from $\theta(?4)$, $\theta(?3)$'s sibling, we remove b:S. In a similar manner, we also remove c:c and s:y from $\theta(?4)$.

Thus, after this step, $\theta(?4) = \{c$:m, b:E, s:$n\}$. Table 1 shows the resulting θ values of all edges at the end of Step 2.

Note that the order in which we process edges in Step 2 may yield different θ sets at the end of Step 2. For example, we could have subtracted s:y from $\theta(?1)$ as opposed to from $\theta(?2)$. However, any facet that could have been eliminated during Step 2 with a certain order is eliminated after Step 3 regardless of the order chosen during Step 2. Thus, the order in which we processed the edges in Step 2 does not matter.

Let us now move on to Step 3 in our example. In this step, we check to see if there is a $\theta(n$-$x)$ that is missing all facets g:$*$ of a given label g. If so, then we remove all facets g:$*$ from all siblings $\theta(n$-$y)$, where $y \neq x$.

As an example of this step, consider edge ?2, where $\theta(?2) = \{b$:$E\}$. Note that all facets having the label "c" (category) are missing in $\theta(?2)$. Thus, from $\theta(?1)$, $\theta(?2)$'s sibling, we remove all facets c:$*$ having the label "c", namely c:c. In a similar manner, we also remove s:y from $\theta(?1)$. Thus, after this step, $\theta(?1) = \{b$:$S\}$. Again, Table 1 gives the states of the other edges at the end of Step 3.

We are now ready to move on to Step 4, the final step, in our example. In this step, we check to see if there is an edge e such that all of the facets in $\theta(e)$ have the same label g. If so, then we remove all facets g:$*$ from all descendant and ancestor edges d of e and we proceed similarly for the siblings d_1 of edges d. As an example of this step, consider edge ?1, with $\theta(?1) = \{b$:$S\}$.

Note that all facets in $\theta(?1)$ have the label "b" (brand). Edge ?1 has no ancestors but its descendants are edges ?3, ?4, ?7, and ?8. Thus, we remove all facets with label "b" from the θ values of these edges (it makes no sense to have a "b" label since edge ?1 is already specifying the brand). After doing so, we have $\theta(?3) = \{c$:c, s:$y\}$, $\theta(?4) = \{c$:m, s:$n\}$, $\theta(?7) = \{c$:c, s:$y\}$, and $\theta(?8) = \{c$:m, s:$n\}$.

Since edge ?1 has no ancestors, we were not able to take advantage of the part in this step where we "proceed similarly for the siblings d_1 of edges d". However, had edge ?1 had an ancestor d_a, we would remove all facets with label "b" not only from $\theta(d_a)$ but also from the θ values of d_a's siblings.

The final solution θ output by our algorithm for this example is given in the last column of Table 1.

We will now prove that Algorithm 1 outputs correct solutions θ.

5.1 Correctness of Algorithm 1

Given hierarchies H_1 and H_2, facet annotations $f(H_1)$, a set of objects O, a set of facets F, and the properties from Section 3, let us show that the solution θ produced by Algorithm 1 is correct.

Theorem: Algorithm 1 is correct.

Proof:

Part 1 of Proof:

First we show that for any assignment δ that is structurally-sound and conflict-free, θ covers δ. To show this, we will prove its contrapositive: we will assume that δ is structurally-sound and conflict-free and that θ does not cover δ. We will then seek a contradiction.

Edge	Initialization	End Step 1	End Step 2	End Step 3	End Step 4
?1	c:m,c:c,b:S,b:E,s:y,s:n	c:c,b:S,s:y	c:c,b:S,s:y	b:S	b:S
?2	c:m,c:c,b:S,b:E,s:y,s:n	b:E,s:y	b:E	b:E	b:E
?3	c:m,c:c,b:S,b:E,s:y,s:n	c:c,b:S,s:y	c:c,b:S,s:y	c:c,b:S,s:y	c:c,s:y
?4	c:m,c:c,b:S,b:E,s:y,s:n	c:m,c:c,b:S,b:E,s:y,s:n	c:m,b:E,s:n	c:m,b:E,s:n	c:m,s:n
?5	c:m,c:c,b:S,b:E,s:y,s:n	c:c,b:E,s:y	c:c,b:E,s:y	c:c	c:c
?6	c:m,c:c,b:S,b:E,s:y,s:n	c:m,b:E,s:y	c:m	c:m	c:m
?7	c:m,c:c,b:S,b:E,s:y,s:n	c:c,b:S,s:y	c:c,b:S,s:y	c:c,b:S,s:y	c:c,s:y
?8	c:m,c:c,b:S,b:E,s:y,s:n	c:m,c:c,b:S,b:E,s:y,s:n	c:m,b:E,s:n	c:m,b:E,s:n	c:m,s:n
?9	c:m,c:c,b:S,b:E,s:y,s:n	c:c,b:E,s:y	c:c,b:E,s:y	c:c,b:E,s:y	s:y
?10	c:m,c:c,b:S,b:E,s:y,s:n	c:m,c:c,b:S,b:E,s:y,s:n	c:m,b:S,s:n	c:m,b:S,s:n	s:n

Table 1: State at the end of each step in Algorithm 1 example.

Since θ does not cover δ, we know there exists an edge e such that, at some point in the algorithm, $\delta(e) = g{:}v$ was removed from $\theta(e)$.

Suppose $g{:}v = \delta(e)$ was removed from $\theta(e)$ during Step 1. Then there exists $o \in O$ with $e \in \mathcal{P}_2(o)$ such that $g{:}v' \in S_1(o)$ and $g{:}v \in C(g{:}v')$. Since $e \in \mathcal{P}_2(o)$ and $g{:}v = \delta(e)$, we know that $g{:}v \in S_2(o, \delta)$. Since $g{:}v \in C(g{:}v')$, $v \neq v'$, which implies that δ is not conflict-free, a contradiction.

Since $\theta(e) = F$ for all edges e at the beginning of the algorithm, θ initially covers δ. Facets are removed from θ one by one, and at some point in the algorithm, θ no longer covers δ. Let $\delta(e)$ be the first facet of δ to be removed from $\theta(e)$. That is, before removing $\delta(e) = g{:}v$ from $\theta(e)$, θ covers δ. Just after removing $\delta(e) = g{:}v$ from $\theta(e)$, θ does not cover δ.

Let θ_b be the value of θ immediately before the removal of $\delta(e)$ from $\theta(e)$. Since $\delta(e)$ is the first facet of δ to be removed from $\theta(e)$, we know that θ_b covers δ.

Now suppose $g{:}v = \delta(e)$ was removed from $\theta(e)$ during Step 2. Then e is a sibling of an edge e' in E_2 such that $g{:}v \in \theta_b(e')$ and there does not exist $v' \neq v$ with $g{:}v' \in \theta_b(e')$. As stated earlier, θ_b covers δ. Since δ is structurally-sound, P1 must hold. Then the label of $\delta(e')$ must be g, forcing $\delta(e') = g{:}v$, but then δ violates P2. Hence δ is not structurally-sound, a contradiction.

Suppose $g{:}v = \delta(e)$ was removed from $\theta(e)$ during Step 3. Then e is a sibling of an edge e' in E_2 such that for all $g{:}*$, $g{:}* \notin \theta_b(e')$. Since for all $g{:}*$, $g{:}* \notin \theta_b(e')$, and θ_b covers δ, δ violates P1 and hence is not structurally-sound, a contradiction.

Finally, suppose $g{:}v = \delta(e)$ was removed from $\theta(e)$ during Step 4. Then edge e is either a descendant/ancestor or a sibling of a descendant/ancestor of some edge e' for which there exists exactly one label g such that $g{:}* \in \theta_b(e')$. Since δ is structurally-sound, P1 must hold. If e is a sibling of a descendant/ancestor d_1 of the edge e', then, since δ satisfies P1, all sibling edges have the same label, and so $\delta(d_1) = g{:}*$. In this case and in the case that e is a descendant/ancestor of e', e' has a descendant/ancestor $d_0 \in \{e, d_1\}$ with $\delta(d_0) = g{:}*$. Since there exists exactly one label g such that $g{:}* \in \theta_b(e')$, and since θ_b covers δ, $\delta(e') = g{:}*$, but since e' and d_0 are on the same path, δ violates P3. Hence δ is not structurally-sound, a contradiction.

Thus we have shown that if assignment δ is structurally-sound and conflict-free, then θ covers δ.

Part 2 of Proof:

We now need to show that if θ covers an assignment δ, then δ is conflict-free (but may be structurally unsound).

Consider an object o. Suppose facet $g{:}v \in S_1(o)$ and there exists $g{:}v' \in F$ with $v \neq v'$. Then $g{:}v' \in C(g{:}v)$ and in Step 1 of the algorithm, $C(g{:}v)$ is removed from all $\theta(e)$ with $e \in \mathcal{P}_2(o)$. And, in the following steps, facets are not added, but just removed from $\theta(e)$. Therefore, after the last step of the algorithm, $g{:}v' \notin \theta(e)$ for $e \in \mathcal{P}_2(o)$.

Since θ covers δ, assignment δ only uses facets actually in θ. Thus, $g{:}v' \notin S_2(o, \delta) = \cup_{j \in \mathcal{P}_2(o)}\{\delta(j)\}$.

Thus we have shown that if θ covers an assignment δ, then δ is conflict-free. ■

5.2 Non-Minimality of Algorithm 1

Although we showed in the previous section that the solutions produced by Algorithm 1 are correct, the solutions are not always minimal.

To see an example of where Algorithm 1 fails to produce a minimal solution, let us examine Figure 3. Here the set of facets is $F = \{a{:}1, a{:}2, b{:}1, b{:}2, c{:}1, c{:}2\}$.

If we use Algorithm 1 on H_1 and H_2, we arrive at solution θ_1 (Figure 3(c)). While this solution is correct, it is not minimal, for the solution θ_2 (Figure 3(d)) is also correct and $\theta_2 \leq \theta_1$.

To see that θ_2 is correct, note that any structurally-sound assignment covered by θ_1 is also covered by θ_2. For example, suppose we are trying to create a structurally-sound assignment δ_n with $\delta_n(?3) = b{:}1$. Then we are forced to choose $\delta_n(?2) = a{:}2$. But this leaves no choice for $\delta_n(?1)$ that allows δ_n to be structurally-sound.

Our goal in designing Algorithm 1 was to quickly produce correct, near-minimal solutions. A slight modification of the proof given in Section 5.1 would show that our solution θ is already correct after Step 1. Steps 2-4 simply serve to make our solution approach the minimal solution.

We could add extra machinery to Algorithm 1 to make the final solution even closer to the minimal solution. This extra "constraint satisfaction" machinery could deduce, for instance, that since $a{:}2$ and $b{:}1$ will be used along the first two edges of H_2, these facets are no longer choices for ?3 and ?4. However, "constraint satisfaction" machinery is readily available, and so we omit a further discussion of these techniques.

5.3 Generalization of Algorithm 1

In Section 5, when we created Algorithm 1, we used all properties from Section 3, because the largest set of properties allowed us to deduce the most about the annotations

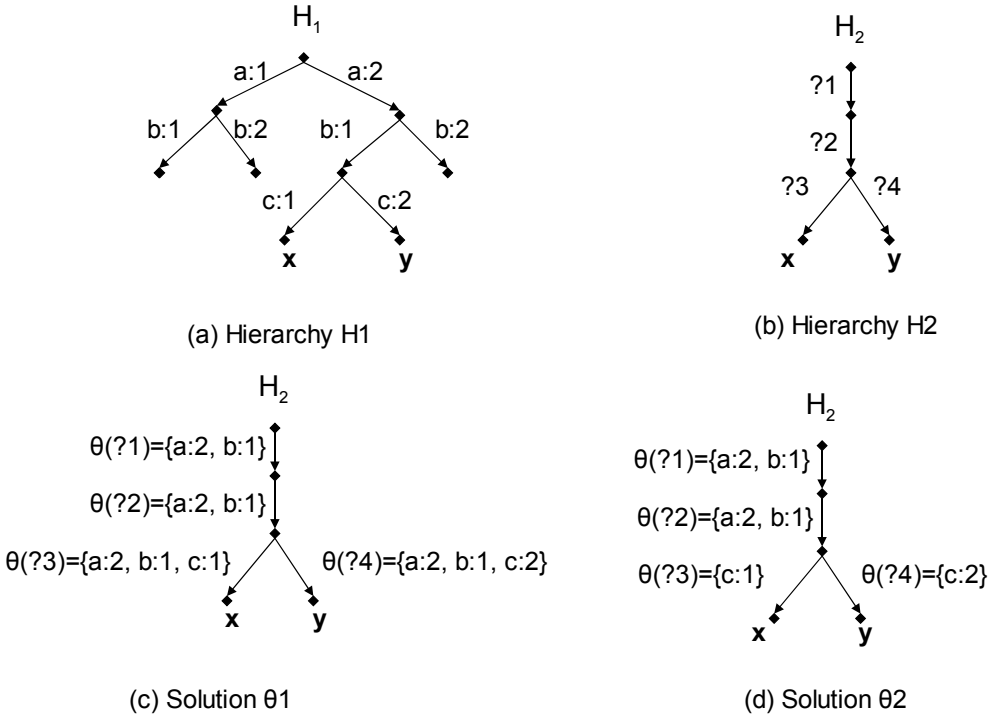

Figure 3: Example to illustrate meaning of minimality.

in H_2.

However, it is possible to create a useful algorithm even in the case when some of the properties from Section 3 do not hold. As an example, consider properties P1, P2, and P3. (See Section 3.)

Suppose only a subset T of {P1, P2, P3} holds. We present a generalized algorithm designed for this case.

Given our smaller set of properties, we would want to find a solution θ that satisfies a modified version of correctness. The only difference between the original definition of correctness and the modified definition is that instead of demanding that θ cover only those conflict-free assignments that are structually-sound, we now demand that θ cover all conflict-free assignments that satsify the properties in T. We formally define T-correctness below.

T-Correctness. Let T be a subset of {P1, P2, P3}. A solution θ is T-correct if

- For all conflict-free assignments δ that satisfy all properties in T, θ covers δ;
- If θ covers an assignment δ, then δ is conflict-free.

If a solution is correct, then it is also T-correct for any T. In other words, the definition of T-correctness is more selective than that of correctness. In the case that T ={P1, P2, P3}, T-correctness is the same as correctness.

If T is a proper subset of {P1, P2, P3}, then Algorithm 1 no longer outputs T-correct solutions.

We can generalize Algorithm 1 to produce T-correct solutions simply by removing the steps from Algorithm 1 that relied on properties not in T.

The correctness proof in Section 5.1 showed which steps in the algorithm relied on which properties. For example, since Step 2 enforces property P2, we know that if P2 does not hold, we should skip Step 2.

By revisiting the correctness proof, we can see that Step 2 requires P1 and P2, Step 3 requires P1, and Step 4 requires P1 and P3.

Thus, if P1 $\notin T$ then we should skip Steps 2-4.

If P2 $\notin T$ then we should skip Step 2.

If P3 $\notin T$ then we should skip Step 4.

If we skip the appropriate steps, then our modified algorithm produces solutions that are T-correct.

Note that another possible way of generalizing the algorithm is by modifying steps as opposed to eliminating them. For example, if P1 does not hold and T ={P2, P3}, then we do not need to eliminate *all* of Step 2. We can at least check for the case where one edge contains only one facet g:v, and if this case holds, then we can delete any copies of g:v from that edge's siblings and still have a solution that is T-correct.

To illustrate how one would prove correctness of the modified algorithm, we consider one specific variant and show its correctness.

Algorithm 1X: Same as Algorithm 1 but with Step 4 re-

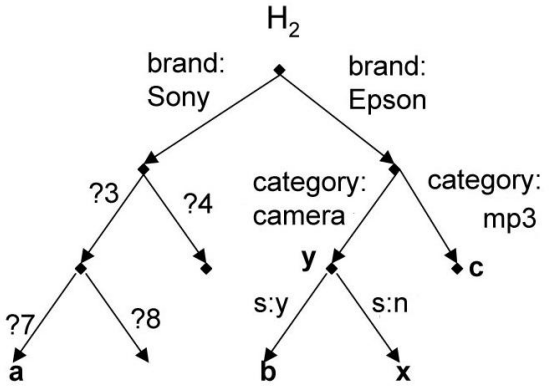

Figure 4: H_2 from motivating example shown with decided edges before text analysis.

moved.

Theorem: Algorithm 1X is T-correct for T ={P1, P2}.

Proof:

Our proof relies heavily on the proof given in Section 5.1.

We can see in the proof from Section 5.1 that any algorithm that includes Step 1 produces solutions that satisfy the second part of T-correctness's definition, namely, that any assignment covered by the solution is conflict-free.

To prove the first part of T-correctness, namely, that θ covers all conflict-free assignments δ that satisfy all properties in T, we will prove the contrapositive: we will assume that θ does not cover δ and that δ is conflict-free and satisfies all properties in T. We will then seek a contradiction.

Since θ does not cover δ, we know there exists an edge e such that, at some point in the algorithm, $\delta(e) = g{:}v$ was removed from $\theta(e)$.

Suppose $g{:}v = \delta(e)$ was removed from $\theta(e)$ during Step 1. The proof here is identical to the corresponding portion from Section 5.1.

As before, let $\delta(e)$ be the first facet of δ to be removed from $\theta(e)$.

Now suppose $g{:}v = \delta(e)$ was removed from $\theta(e)$ during Step 2. The same proof as before will show that δ violates P2. Hence δ does not satisfy T ={P1, P2}, a contradiction.

Suppose $g{:}v = \delta(e)$ was removed from $\theta(e)$ during Step 3. As before we can then see that δ violates P1 and hence does not satisfy T ={P1, P2}, a contradiction.

We have finished proving that θ covers all conflict-free assignments δ that satisfy all properties in T.

Thus we have shown that solutions output by Steps 1-3 of Algorithm 1 are T-correct for T ={P1, P2}. ■

6. COPYING OBJECTS

In Section 4, we broke down the hierarchy merging problem into two parts: identifying the H_2 facets, and moving objects from H_2 to H_1. In this section, we address the second subproblem.

Algorithm 2 presents our algorithm for moving objects from H_2 to H_1.

When moving objects from H_2 to H_1, our goal is to place our objects as far as possible from the root r_1. By doing so, we maximize the amount of information that the merged hierarchy stores for those moved objects.

Input: hierarchies H_1 and H_2, facet annotations $f(H_1)$,
  text annotations for H_2, solution θ
Output: merged hierarchy H_m
Step 1: Initialize set of decided edges: D: $D := \{\}$.
Step 2: Determine set of decided edges.
  For each edge $e \in H_2$,
    If $|\theta(e)| = 1$, then $D := D \cup \{e\}$.
Step 3: Convert undecided edges to decided edges
  via text analysis.
  Use actual H_2 text annotations.
  (Facet annotations $f(H_2)$ are unknown.)
  For each edge $e \in H_2$ such that $e \notin D$,
    If text analysis determined a single facet for edge e,
    then $D := D \cup \{e\}$.
  If we found new decided edges, then we may loop,
    revisiting Steps 2-4 of Algorithm 1
    and then revisiting Steps 2-3 of this Algorithm 2.
Step 4: Determine certain facets.
  Denote by O_a the set of objects in H_2 but not in H_1.
  For each $o \in O_a$,
    Initialize set of certain facets $X_2(o)$: $X_2(o) := \{\}$.
    For each $e \in \mathcal{P}_2(o)$,
      If $e \in D$, then $X_2(o) := X_2(o) \cup \theta(e)$.
Step 5: Attempt to place new objects into H_m.
  Initialize H_m: $H_m := H_1$.
  For each $o \in O_a$,
    Initialize current node n to the root r_m: $n := r_m$.
    Loop:
      If there exists an edge n-n_c from n to a child n_c
        such that $f(n\text{-}n_c) \in X_2(o)$,
      then $n := n_c$ and repeat Loop.
      Else, stop and place object o at n in H_m.

Algorithm 2: Moving objects from H_2 to H_1

We say that an edge e is decided if $|\theta(e)| = 1$, where θ is the solution found by Algorithm 1. Let us denote the set of decided edges in H_2 by D.

At this point, some of the undecided edges may be converted to decided edges via text analysis of the actual H_2 annotations. As discussed in the Introduction, we expect our object matching machinery to be used in conjunction with text analysis tools that help further reduce the options of H_2 facets. So for instance, in our example, at the end of Algorithm 1, $\theta(?3)$ = {"category: camera", "s: y"}. By examining the text on edge ?3 of H_2, we may determine that there are no words related to sales, discounts, prices, etc. on that edge, so we can rule out facet $s{:}y$. Thus, edge ?3 becomes a decided edge with $\theta(?3)$ = {"category: camera"}.

Once we have settled on a set of decided edges in H_2, we can then determine, for each object in H_2, which facets must be associated with the object. We will refer to these facets as certain facets. For each $o \in O$ that is in H_2 but not in H_1, let $X_2(o)$ be the set of certain facets for object o. That is, let $X_2(o) := \cup_{j \in \mathcal{P}_2(o) \cap D} \theta(j)$.

As an example, let us again consider H_1 and H_2 from Figure 1. Figure 4 displays H_2 with decided edges labeled. From Figure 4, we see that $X_2(x)$ ={"category: camera", "brand: Epson", "s: n"} and $X_2(y)$ = {"category: camera", "brand: Epson"}.

Once we have associated an object (that is in H_2 but not in H_1) with its certain facets, we can then attempt to place the object into H_1. To do so, we start at the root r_1. When

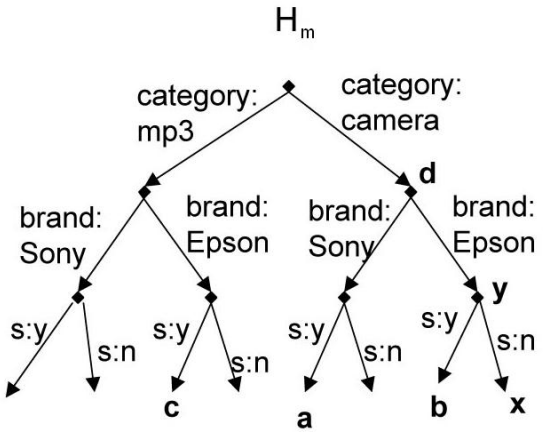

Figure 5: H_m from motivating example shown with objects x and y added from H_2.

we are at a given node, if any of the edges from the node to a child are annotated with one of the object's certain facets, then we traverse the tree to the child and repeat. Otherwise, we stop and place the object at the current node.

Continuing with our example, we use the $X_2(x)$ and the $X_2(y)$ information to place objects x and y into H_1, as far away from r_1 as possible. Since we know that object x is an Epson camera that is not on sale, and that object y is an Epson camera, we place x and y in H_1 at the appropriate nodes. (Since we do not know whether or not y is on sale, we cannot place it any deeper than we placed it in H_1).

Once we place the objects from H_2 into H_1, we have created our merged hierarchy H_m. Figure 5 shows the H_m that corresponds to H_1 and H_2 from Figure 1.

7. EXPERIMENTS

How well our scheme performs depends on two main factors:

- The number of properties that hold (and how well they hold). In some well-structured hierarchies, like those used by Amazon [2] and the Flamenco system [6], most properties hold and there are an insignificant number of exceptions (e.g., misplaced objects). As we will see in the two examples we consider in this section, there are hierarchies that are not as cleanly structured.
- The number of overlapping objects among the two hierarchies. As we will see in one of our experiments, the more overlapping objects we have, the less we learn about H_2 from each additional overlapping object.

In this section we study two scenarios, one with only a handful of what we call misplaced objects, and a second one with a more significant number of misplaced objects. Our scenarios are only intended as illustrations, as there is clearly a wide range of possible performances in practice.

7.1 Dmoz Experiment

We considered two sub-hierarchies extracted from the Open Directory (http://dmoz.org), a directory maintained by volunteers. Figure 6 illustrates the basic structures of the two sub-hierarchies, H_1 and H_2. For convenience, we labeled the nodes with integers, and for clarity, we removed from Figure 6 all objects but a few sample objects and all states but California and Ohio.

Both of the sub-hierarchies deal with art museums, art organizations, history museums, and history organizations in the United States. The objects in the two hierarchies, e.g., o_1, o_{10} in Figure 6, are URLs, e.g., "http://www.erieartmuseum.org/". The difference between H_1 and H_2 is that while H_1 organizes objects in the "art/society—museum/organization—USA state" way (branching first on whether the URL fits more closely under "art" or "society"), H_2 organizes objects in the "USA state—art/society—museum/organization" way (branching on the state before branching on art/society).

Because Dmoz is maintained by volunteers, some objects appear in only one of the hierarchies, others appear in both. For example, *Artcom Museum Tour: California* (http://www.artcom.com/museums/ca.htm) can be found at node 7 of Figure 6(2), but not at node 7 of Figure 6(1). Clearly, it would be beneficial to consolidate these two hierarchies as we propose here.

Table 2 presents some statistics for the two hierarchies, including the number of nodes, the number of objects, and the number of distinct facets. In this scenario, there were 552 overlapping objects present in both H_1 and H_2.

As a first step we did some data cleansing to make the hierarchies comply with our model and to eliminate some problematic objects. It is important to note that this whole cleansing process could be done automatically, without knowledge of how H_2 is organized. In particular, we did *not* eliminate misplaced objects from H_2 because that can only be done by understanding the facets of H_2, which is what our algorithm will do.

- The Dmoz hierarchies use implicit labels on the edges. Thus, for the edges with values such as "Arizona" or "California", the reader must infer that they refer to "US States", i.e., have the label "US States". Therefore, to create our facet set F, we transform the facets from H_1 into "label:value" format. We used synthetic labels "f1" through "f7" since our algorithm does not associate any semantic meaning to labels.
- The leaf-nodes normally contain URLs. However, some leaf-nodes point to other nodes, e.g., node 123 in H_1 pointed to node 11 in H_2. In this case, all URLs at node 11 in H_2 were copied to node 123 in H_1.
- If an object o appears at more than one node in H_1, then its conflicting facets (those facets in $S_1(o)$ that share a label with another facet in $S_1(o)$) are removed in Step 1 of Algorithm 1. This is done to avoid errors involving the one-value-per-label property. (There were 4 such objects in H_1.)
- If an object o appears at more than one node in H_2, then during Step 1 of Algorithm 1, we ignore o when considering a set of sibling edges in H_2 in which more than one of these edges is in a path from root r_2 to a node containing o. This is done to avoid errors involving the one-value-per-label property. (There were 3 such objects in H_2.)
- To simplify our programming, we remove objects at non-leaf nodes. (We removed 340 objects from H_1, and 1441 objects from H_2.) We stress that we perform this action only to simplify our experiment. A generalized version of our algorithm can handle these objects (and would

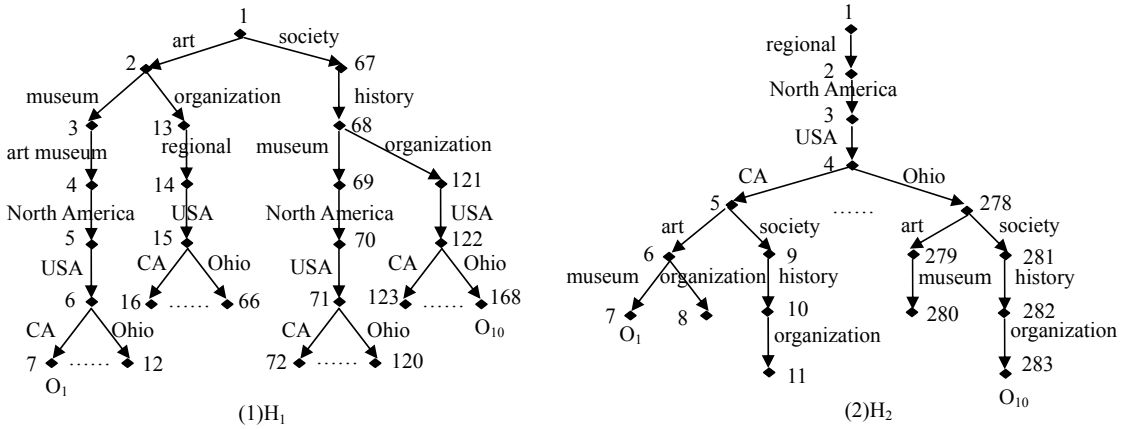

Figure 6: The two dmoz directories.

presumably perform better since it would use more objects).

Even after the data cleansing, our resulting hierarchies may still have some misplaced objects (violating the well-formed hierarchies property). For example, "http://www.texasranger.org/" is considered a history museum in H_1 but an art museum in H_2. There are a total of 9 misplaced objects in our data set. These objects were not removed and are the cause of the errors we observe in our result.

To evaluate the performance of our algorithm, we use metrics from the following three categories. Note that for our evaluation we have available the "correct answer," i.e., the facet annotations of H_2 (something that is not available to our algorithm). We say that an edge e is a *correctly decided edge* if the output of our algorithm for that edge, $\theta(e)$, contains a single facet that matches the H_2 facet.

- Metric Category 1: Number, recall, and accuracy of decided edges. As defined in Section 6, an edge e is decided if $|\theta(e)| = 1$. *Recall* is defined as the number of decided edges divided by the number of edges. Obviously, a larger recall suggests that our algorithm was able to learn more about H_2. Because of misplaced objects, errors may occur. Hence, we define *accuracy* as the number of correct decided edges divided by the number of decided edges.
- Metric Category 2: Number, compress rate, and accuracy of candidate facets on undecided edges. If an edge e is not decided, $\theta(e)$ contains all the candidates for the facet on e. The number of candidate facets is defined as $\sum_{e,where|\theta(e)|\neq 1} |\theta(e)|$. Obviously, a smaller number suggests that our algorithm performs better. To measure the algorithm's ability to eliminate candidate facets, we introduce the *compress rate* and define it as the quantity $1 - X/Y$ where X is $(\sum_{e,where|\theta(e)|\neq 1} |\theta(e)|)$ and Y is $|F|$ times the number of undecided edges. Because of errors, candidate facets may be wrong. To quantify, we define *accuracy* as the number of undecided edges whose correct facet is included in $\theta(e)$ divided by the number of undecided edges.
- Metric Category 3: Average depth and correctness of copied new objects. For each object that is copied from H_2 to H_1 we compute its depth from the H_1 root. The *average depth* is the average depth over all copied objects. Because of errors, some objects may be copied to the wrong place in H_1. *Correctness* is defined as the number of objects that are copied to correct places divided by the number of copied objects.

Table 3 shows the results of our experiment on the Dmoz hierarchies. Experiment 1 uses the hierarchies as we have described them; Experiment 2 is a variant we describe below.

For Experiment 1, there were 105 decided edges, six of which were wrong, so that the accuracy of decided edges was $\frac{105-6}{105} = 94.3\%$. The compress rate of candidate facets on undecided edges was $1 - \frac{839}{((282-105)*59)} = 92.0\%$.

Although there were some incorrectly-decided facets in H_2, fortunately, the new objects categorized by these facets in H_2 were all copied to the root of H_1. Therefore, none of this incorrect information was propagated so the correctness of the copied objects was 100%. The average depth of copied objects was 0.39: there were 10 objects copied to a depth of 2, and 26 objects copied to a depth of 3.

The reason for the relatively low average depth of copied objects lies with the "odd" structure at the top of H_2. In particular, there are three edges in a row with no siblings. This is an artifact of the process that extracted H_2 from the main Dmoz hierarchy. One would not expect many zero-sibling edges in practice (zero-sibling edges are not very useful for navigation), and yet our H_2 has three edges in a row with no siblings.

Given this "odd" top structure, our algorithm in Experiment 1 figured out that these three edges in H_2 (1-2, 2-3, 3-4) involved three of the facets among "f3: history", "f4:regional", "f5:North America", or "f6: USA", but it could not figure out which facet corresponded to which edge. Not knowing the facets of the three edges precisely prevented us from being able to eliminate those facets from the other edges below.

To evaluate performance in a less unusual setting, we conducted Experiment 2. In this case, we manually decided the facets on edges 1-2, 2-3, 3-4 in H_2. We can think of this mapping as the result of using text analysis to choose among the candidate facets yielded by Algorithm 1. As we have argued,

	Num. of nodes	Num. of edges	Num. of objects	Num. of distinct facets
H_1	168	167	1275	59
H_2	283	282	915	59

Table 2: Some statistics for the two dmoz hierarchies.

Exp #	Decided edges			Candidate facets on undecided edges			Copied objects	
	num.	accuracy	recall	num.	compress rate	accuracy	average depth	correctness
1	105	94.3%	37.2%	839	92.0%	94.9%	0.39	100%
2	109	94.5%	38.7%	529	94.8%	94.8%	0.62	100%

Table 3: Dmoz experimental results.

the results of Algorithm 1 can naturally be used in conjunction with text analysis to improve overall performance.

After the three top facets were decided, we re-ran Algorithm 1. In comparison to the results from Experiment 1, in Experiment 2 the number of candidate facets of undecided edges decreased by 37%, and the compress rate increased by 3% (see Table 3). Although the average depth of copied objects in Experiment 2 was still low, it showed significant improvement over the average depth in Experiment 1, and this was achieved by deciding only three facets. In this experiment, 36 objects were copied to a depth of 5 or greater.

To understand the impact of overlapping objects, we ran a sensitivity analysis on the Dmoz hierarchies. We randomly removed overlapping objects (objects common to both hierarchies) and measured how that affected the number and accuracy of decided edges returned by the algorithm. As expected, the more overlapping objects we removed, the fewer decided edges we were able to achieve. We calculated an average over five runs. We started with 105 decided edges with 552 common objects; with 251 common objects, we had 77.6 decided edges, and with 52 common objects remaining, we had on average 40.4 decided edges. Throughout this process of removing common objects, the accuracy of decided edges remained relatively constant.

In summary, our results on the Dmoz hierarchies show that object placement can indeed yield very useful information about H_2, even if there are some misplaced objects that "partially" invalidate our properties. And because our analysis is complementary to traditional text-based mapping techniques, we believe that both approaches can be used jointly in practice to achieve better results than what each approach can get on its own.

7.2 GLUE Hierarchies

For our second scenario we consider company indices obtained from Yahoo.com, H_2, and TheStandard.com, H_1. In each company index, companies are first organized by sector and then, within each sector, the companies are organized by industry. H_1 has 333 nodes and 13634 objects, while H_2 has 115 nodes and 9504 objects (See Table 4). This dataset was also used in an experiment by the GLUE system [5].

These hierarchies deal with ambiguous categories and have many more misplaced objects than the Dmoz hierarchies. The correct mappings between facets (needed to evaluate results) are unclear, and we need to use the techniques of [5] (Jaccard similarity) to infer matches. We went over those 114 matches manually, and threw out those judged to be wrong, i.e., based on too few overlapping companies. We call the remaining 103 matches "correct" (only available for evaluation). As an example of a "correct" match, we find that "Mobile Homes RVs" in Yahoo.com corresponds to "Recreational Vehicles" in TheStandard.com.

Given these correct matches, we find a significant number of misplaced objects. For example, "Skyline Corporation," which is classified under "Mobile Homes RVs" in Yahoo.com, is under "Manufactured Buildings" in TheStandard.com. Among the 5389 overlapping objects, there are 1986 conflicting ones, 1986/5389 = 37%.

Among the 1986 conflicting objects, 1201 of them have one facet conflict (only one facet is conflicting, the other one is the same), while 785 of them have two facet conflicts (in this example, we have just two levels in our hierarchies).

To deal with this ambiguous data, we made two modifications to Algorithm 1:

- We add a filtering step (before Step 1) to try to eliminate misplaced objects. For each object, we trace its path in H_2, and for each edge along the path, we record the facets of the object. After we are done processing all of the objects, we are left with a facet vote (count) at each edge; the vote records how many objects under it possess a particular facet. During the filtering step, for each edge e we set $\theta(e)$ equal to the set of those facets f for which the percentage of objects under e that possess f exceeds a certain threshold. (We used a threshold = 25% for this experiment.)
- In Step 3 we ignore edges e with $\theta(e) = \{\}$. An empty facet set for e only can happen when the well-formed property is violated, and hence is a signal that Algorithm 1 is unable to deal with edge e. By eliminating e from further consideration, we ensure the problem does not propagate to other edges.

Using the modified algorithm on the TheStandard-Yahoo hierarchies, we were able to achieve 73 decided edges out of 103 total edges, and of those 73 decided edges, 65 of them agreed with the correct matches. The accuracy of our decided edges is thus 65/73 = 89% and the recall is 73/103 = 71%. If we wanted to increase our recall, then we could, after Step 4, for each of the edges e with $|\theta(e)| > 1$, simply choose the candidate facet with the most votes. Doing this allows us to achieve 93 decided edges, with 78 of those being correct matches. Accuracy of decided edges is now 78/93 = 84% while recall is increased to 93/103 = 90%.

These performance numbers are similar to the results obtained by the GLUE system for the same domain. In contrast to the GLUE system, which returns a best guess for

	Num. of nodes	Num. of edges	Num. of objects	Num. of distinct facets
H_1	333	332	13634	332
H_2	115	114	9504	114

Table 4: Some statistics for the TheStandard-Yahoo experiment.

each concept in the hierarchy, our method only returns a guess for an edge if the edge is decided. Thus, when our system does make a guess, we can have more confidence that the guess returned is correct.

Given that the GLUE system did not exploit shared objects and that ours did, a strict comparison of performance numbers would not be fair. However, we emphasize that our method of merging hierarchies is designed to be complementary. It can enhance the performance of other methods that do not take shared objects into account.

8. RELATED WORK

Our work is relevant to schema/ontology matching [5, 8, 11, 12, 14, 15, 17], as both methods try to find corresponding facets (i.e., concepts, in schema/ontology mapping) between different structures (schemas/ontologies/hierarchies). Our approach, in contrast to some of the other methods, uses object placement as opposed to text-matching as the primary means to compute similarity between facets. Similar to other methods such as "similarity flooding" [5, 13], our approach infers information about an edge based on knowledge gained from the edge's neighbors (siblings/ancestors/descendants), but we do so by removing candidate facets of a edge based on the facet properties.

This work is similar to entity resolution [3] in that while entity resolution associates multiple records with a common entity, our method of merging hierarchies attempts to associate multiple text strings (e.g., "automobile", "vehicle") with a common facet (e.g., "type: car").

The problem of merging hierarchies is a specific instance of the more general problem of data integration [4, 9]. With our data organized in hierarchies, we seek to exploit the hierarchical structure to infer matches between facets. There are other works [1, 16] that ignore the hierarchical structure, and instead, consider the case when the instances are organized in a flat set of categories.

GLUE [5] is a system that uses machine learning to learn how to map between ontologies. Compared with GLUE, the method proposed in this paper (1)does not use learning technology to calculate similarity, but rather uses shared objects to eliminate possibilities; (2)returns a set of possible matches, as opposed to always determining a best match; (3)maps between hierarchies as opposed to ontologies. The difference between ontologies and hierarchies is that typically in ontologies, the child node is a specialization of its parent, while in two different hierarchies, we could branch on attributes in different orders. Thus, an algorithm that deals with hierarchies must create a framework flexible enough to handle this extra degree of freedom.

HICAL [10] uses machine learning to map between hierarchies. Our system addresses the same scenario–two hierarchies with objects(instances) placed in each hierarchy–and the goal for both systems is to transfer instances from one hierarchy to the other. Our system also uses the same basic idea of exploiting the shared objects. However, the methods are different. HICAL uses a statistical method: the "κ statistic" method [7], while we emphasize the implicit knowledge contained in human-constructed hierarchies, namely the relationships among facets on sibling, ancestor and child edges, i.e., P1, P2 and P3 (see Section 3). The HICAL method has difficulty with hierarchies in which the general and more specific categories are independent; this is one case that our method was designed to handle.

9. CONCLUSION

We have presented a method for merging hierarchies that uses object placement to deduce facet matches between the hierarchies. In our method, our matching algorithm returns, for each of the edges in the new hierarchy, a set of candidate facets from the base hierarchy. We can then use text-matching to further eliminate candidate facets. Having deduced which facets correspond to which objects from the new hierarchy, we then use this information to place the new objects into the base hierarchy. Our proof of concept experiments showed some success, but we propose that the most value can be gained from our approach when used in conjunction with text analysis tools.

10. ACKNOWLEDGMENTS

The authors would like to thank AnHai Doan of the University of Wisconsin for his useful comments on an earlier draft and for providing access to his data. Sergey Melnik of Microsoft Research also provided comments.

11. REFERENCES

[1] R. Agrawal and R. Srikant. On integrating catalogs. In *Proc. of the Tenth Int'l World Wide Web Conference*, pages 603–612, 2001.

[2] Amazon.com. http://www.amazon.com.

[3] O. Benjelloun, H. Garcia-Molina, J. Jonas, Q. Su, and J.Widom. Swoosh: a generic approach to entity resolution. Technical report, Stanford University, 2005.

[4] P. Brown, P. Haas, J. Myllymaki, H. Pirahesh, B. Reinwald, and Y. Sismanis. Toward automated large-scale information integration and discovery. In *Data Management in a Connected World*, pages 161–180, 2005.

[5] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *The Eleventh International WWW Conference*, pages 662–673, 2002.

[6] Flamenco system. http://flamenco.berkeley.edu/index.html.

[7] J. L. Fleiss. *Statistical Methods for Rates and Proportions*. John Wiley and Sons, 1973.

[8] L. M. Haas, M. A. Hernandez, H. Ho, L. Popa, and M. Roth. Clio grows up: From research prototype to industrial tool. In *Proceedings of the 24th ACM*

SIGMOD International Conference on Management of Data, pages 805–810, 2005.

[9] A. Y. Halevy, A. Rajaraman, and J. Ordille. Data integration: The teenage years. In *Proceedings of VLDB*, 2006.

[10] R. Ichise, H. Takeda, and S. Honiden. Rule induction for concept hierarchy alignment. In *Proceedings of the IJCAI-01 Workshop on Ontology Learning*, 2001.

[11] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *Knowledge Engineering Review*, 18(1):1–31, 2003.

[12] D. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, pages 483–493, 2000.

[13] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th ICDE*, 2002.

[14] N. F. 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.

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

[16] S. Sarawagi, S. Chakrabarti, and S. Godbole. Cross-training: Learning probabilistic mappings between topics. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2003.

[17] G. Stumme and A. Maedche. FCA-Merge: Bottom-up merging of ontologies. In *7th Intl. Conf. on Artificial Intelligence (IJCAI '01)*, pages 225–230, 2001.