

# Matching Hierarchies Using Shared Objects<sup>\*</sup>

Robert Ikeda<sup>1</sup>, Kai Zhao<sup>2</sup>, and Hector Garcia-Molina<sup>1</sup>

<sup>1</sup> Stanford University, Computer Science Department, Stanford CA 94305, USA

<sup>2</sup> NEC Laboratories China, 1 Zhongguancun East Road, Beijing, China

**Abstract.** One of the main challenges in integrating two hierarchies is determining the correspondence between the edges of each hierarchy. Traditionally, this process, which we call *hierarchy matching*, is done 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 two algorithms that, given a hierarchy with known *facets* (label-value pairs that define what objects are placed under an edge), determine feasible facets for a second hierarchy, based on shared objects. One algorithm is rule-based and the other is statistics-based. In the experimental section, we compare the results of the two algorithms, and see how their performances vary based on the amount of noise in the hierarchies.

**Key words:** data integration

## 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 matching, in particular, how to determine corresponding edges between two related hierarchies. The need to match hierarchies 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 match one hierarchy to a second known 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

---

<sup>\*</sup> A 2-page poster will be presented at ICDE08. The poster introduces the problem we address but presents no results.

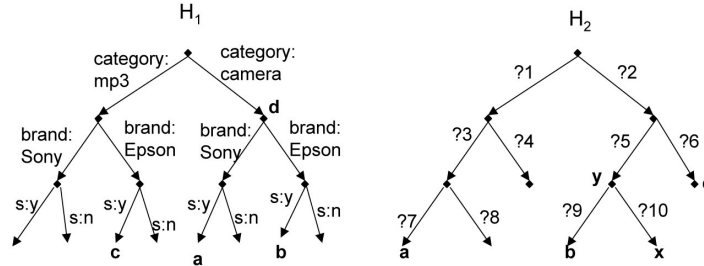


Fig. 1. Motivating example of hierarchies to be merged.

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 match to  $H_1$ . Some of the objects in  $H_2$  also appear in  $H_1$ , but not all. We assume that by applying an entity resolution solution, we have matched objects across the hierarchies and stored the matches in a table. Our goal is to determine which facets from  $H_1$  correspond to the edges in  $H_2$ . 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. The second alternative is to study the objects in the hierarchies to find correspondences. 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 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”.

In other cases we may have insufficient information to match edges 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.

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. We start by defining our model and problem (Sections 2, 3, 4), and then present our both our rule-based algorithm (Section 5) and our statistics-based algorithm (Section 6) for matching hierarchies. 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.

- We are given two hierarchies, a base hierarchy by  $H_1$  and a new hierarchy by  $H_2$ . 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.
- Each hierarchy  $H_i$ , contains nodes  $N_i$  and directed edges  $E_i$ . When the identity of the hierarchy is understood or irrelevant, we will simply omit the hierarchy subscript.

- Each edge  $e \in E_i$  is annotated with a single facet,  $f(e)$ .
- 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  $g$ .
- In our model the facet annotations of  $H_1$  are known and the annotations of  $H_2$  are unknown. We denote the set of facets found in  $H_1$  by  $f(H_1)$ .
- 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  $H$ .
- 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,  $C(g:v) = \{g':v' \text{ such that } g' = g \text{ and } v' \neq v\}$ .
- 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.
- 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.
- We assume that object identities have been resolved, that 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.
- 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  $f(e_1) = \text{“category: mp3”}$  and  $f(e_2) = \text{“category: camera”}$ . 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

Hierarchies sometimes have certain properties that 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  $H_2$ . Only one of the matching algorithms we present will explicitly exploit the properties, and will only do so when the properties are expected to hold in  $H_2$ . If some or all of the properties do not hold, then the missing properties will not be exploited. The properties that might hold for the facet annotations of a hierarchy are as follows.

- **P1.** Same label at a node. Let  $n$  be any node in a hierarchy, where  $f(n-x) = g:v$ . Then, for all edges  $n-y$ ,  $f(n-y) = g:*$ . 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  $f(n-x) = f(n-y)$ . Property P1 says sibling edges must have the same label, and this property 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.

### 4 Problem Definition

Our goal is to guess the facets that were used in  $H_2$  to organize the objects that were placed there. An assignment can be used to represent a set of guessed facets:

**Assignments.** An assignment for  $H_2$  is a function  $\delta(e)$  that assigns a *single* facet from  $H_1$  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  $\theta$  that represents the possibilities for the facets of  $H_2$ .

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

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

Our next task is to define “good” solutions (and good covered assignments), so we can design algorithms that identify such good solutions.

#### 4.1 Fraction of Correct Edges

If we have a “gold standard” that tells us the correct facets for  $H_2$ , then we can define good solutions to be those that give us facets from the gold standard.

There are several ways to quantify how many correct edges a solution has; in our experiments (Section 7) we use the following metric:

$$C(\theta) = \left[ \sum_{e \in E_2} 1\{\theta(e) = \{G(e)\}\} \right] / \left[ \sum_{e \in E_2} 1 \right]$$

where  $G$  is the gold standard assignment and  $1\{x\}$  is equal to 1 if  $x$  is true. In other words,  $C(\theta)$  is the fraction of edges in  $H_2$  where there is a single choice and that facet happens to match the facet defined in the gold standard. This metric is conservative since it gives no credit for edges where the correct facet is among the choices.

## 4.2 Conflict-Free Assignments

If no gold standard is available, we at least want to ensure that our solutions cover assignments that “make sense.” For example, say a shared object  $o$  appears in  $H_1$  on a path that includes an edge with facet “color: red.” If  $H_1$  is well formed, then it must be the case that  $o$  has this facet. Thus, we would not like a solution where in  $H_2$  object  $o$  appears under an edge with a contradictory facet like “color: blue.” The following definitions formalize this notion of correctness.

- 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  $\delta$ . That is,  $S_2(o, \delta) = \cup_{j \in \mathcal{P}_2(o)} \{\delta(j)\}$ .

**Conflict-free.** We say assignment  $\delta$  is conflict-free 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  $g:v_2 \in S_2(o, \delta)$ .

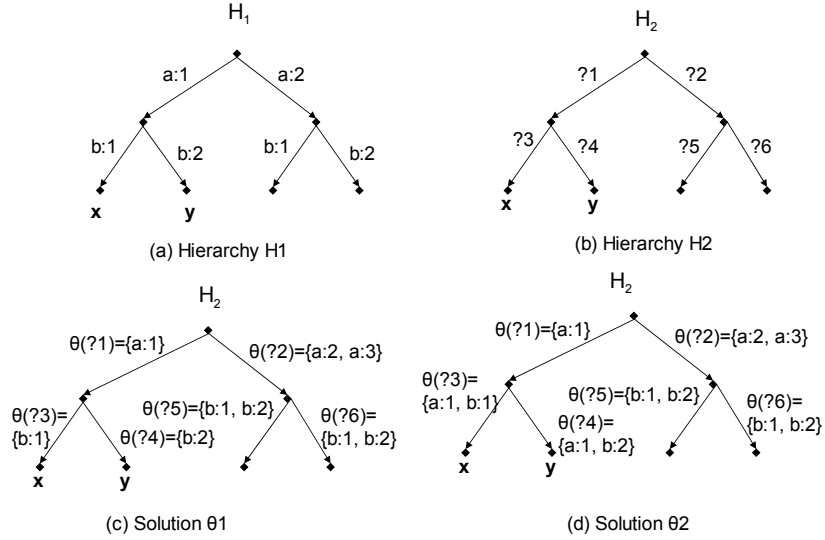
Thus, we can say that a solution is “good” if it only covers conflict-free assignments.

## 4.3 Structurally-Sound Assignments

If the properties of Section 3 are expected to hold, then we may prefer assignments that respect those properties. For example, say  $H_2$  has an edge  $e_1$  followed by a descendant edge  $e_2$ . Assume furthermore that an assignment  $\delta$  gives facet “color: red” to  $e_1$  and facet “color: blue” to  $e_2$ . If Property P3 should hold, then  $\delta$  is not a good assignment.

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

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



**Fig. 2.** Example to illustrate meaning of consistency.

#### 4.4 Consistent Solution

If we want a strong notion of correctness, we may ask for solutions that only cover conflict-free *and* structurally-sound assignments. Furthermore, we can ask that all possible conflict-free and structurally-sound assignments are covered by our solution. 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  $\theta_1$  (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,  $\theta_1$  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  $\theta_1$  because  $b:1$  is a reasonable choice for those two edges. Yet, given the machinery we have, we have no way of ruling out assignments like  $\delta_n$ . 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 consistency:

**Consistency.** A solution  $\theta$  is consistent if

- For all structurally-sound, conflict-free assignments  $\delta$ ,  $\theta$  covers  $\delta$ ;

- If  $\theta$  covers an assignment  $\delta$ , then  $\delta$  is conflict-free (but may be structurally-unsound).

In our example, it is clear that  $\theta_1$  covers all “good” assignments, but as we illustrated, it can generate some unsound (yet conflict-free) assignments like  $\delta_n$ .

At the end of Section 5 we show that our rule-based algorithm generates consistent solutions. As we will state in Section 6, the statistics-based algorithm does not guarantee consistent solutions. However, its robustness is useful when dealing with noisy data sets.

#### 4.5 Minimality

Our final potential notion of correctness tells us that a solution should cover as few unsound assignments as possible. Our algorithms do *not* satisfy this very strong correctness criteria; we present it here solely for completeness. In our opinion, an algorithm that achieves this strong notion of correctness would be too expensive. In our experience, our algorithms strike a good balance between simplicity and achieving near-minimal results in most cases.

Note that there can be multiple consistent solutions to a given matching problem. For instance, Figure 2(d) shows another consistent solution  $\theta_2$  for the same problem. The only difference between  $\theta_1$  and  $\theta_2$  is on edges ?3 and ?4:  $\theta_2(?3)$  and  $\theta_2(?4)$  offer the extra choice of  $a:1$ . An assignment  $\delta_e$  that assigns  $a:1$  both to ?1 and to ?3 or ?4 is conflict-free but unsound. This assignment  $\delta_e$  is not covered by  $\theta_1$  but is covered by  $\theta_2$ .

If we had a choice between consistent solutions  $\theta_1$  and  $\theta_2$ , we would prefer  $\theta_1$  since it generates fewer unsound assignments than  $\theta_2$ . 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 consistent 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  $\theta_1$  is minimal if it is consistent and there is no other consistent solution  $\theta_2 \neq \theta_1$  such that  $\theta_2 \leq \theta_1$ .

## 5 Rule-Based Approach

Here we present our rule-based approach for determining an appropriate solution for the facets in  $H_2$ . We are given hierarchies  $H_1$  and  $H_2$ , facet annotations  $f(H_1)$ , and a set of objects  $O$ .

The rule-based matching algorithm, Algorithm RB, takes these inputs and reasons using the properties from Section 3 to produce a solution  $\theta$ . In case that some of the properties do not hold, 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 not known to hold. Our algorithm can easily 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).

- *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 our experiments (Section 7 we study how well this approach does.

As presented here, the algorithm takes an aggressive approach, but a more conservative algorithm can be created by eliminating steps from the aggressive one.

```

Input: hierarchies  $H_1$  and  $H_2$ , facet annotations  $f(H_1)$ ,
       object set  $O$ 
Output: solution  $\theta$ 
Step 1: Use objects from  $H_1$  to eliminate possibilities.
       Let  $F :=$  the set of all facets from  $H_1$ .
       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:** Rule-based matching algorithm (RB)

Algorithm RB is presented on the next page. Let us illustrate how Algorithm RB works through an example. Suppose we have  $H_1$  and  $H_2$  as in Figure 1.

We assume in this algorithm that the facets in  $H_2$  come from  $H_1$ . If we have knowledge of other possible facets that may have been used in  $H_2$ , we can add

them to our initial set of facets,  $F$ . For this example, we will set  $F$  equal to the set of facets found in  $H_1$ :  $F = \{\text{“category: mp3”}, \text{“category: camera”}, \text{“brand: Sony”}, \text{“brand: Epson”}, \text{“s: y”}, \text{“s: n”}\}$ .

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 after each step of the execution of the example. The facets have been abbreviated to their initials for compactness.

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

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

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  $S_1(a) = \{\text{“category: camera”}, \text{“brand: Sony”}, \text{“s: y”}\}$  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

$$\begin{aligned} \theta(?1) &= \theta(?3) = \theta(?7) = F - C(c:c) - C(b:S) - C(s:y) \\ &= \{c:c, b:S, s:y\}. \end{aligned}$$

That is, the  $H_2$  edges that lead to object  $a$  must have  $\theta$  values consistent with the facets of  $a$  in  $H_1$ . Table 1 shows the  $\theta$  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  $\theta$  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  $\theta$  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  $\theta$  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  $\theta$  values of  $d_a$ 's siblings.

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

## 5.1 Consistency of Algorithm RB

The following theorem tells us that if hierarchy  $H_2$  is “clean,” (well-formed, follows properties), then Algorithm RB guarantees consistency. This property is very important since we definitely do not want a wrong answer when the correct answer is feasible. Of course, in many cases data may be noisy ( $H_2$  is not well-formed) because objects are misplaced. For instance, a particular hotel may be classified as “budget” in  $H_1$  and as “mid-range” in  $H_2$ . If  $H_2$  is noisy, no algorithm can guarantee consistency. With noisy data we can empirically

evaluate the solutions (using the metric of Section 4.1 for instance), as we will do in our experiments.

**Theorem:** Let  $H_1$  and  $H_2$  be well-formed hierarchies,  $f(H_1)$  be the facet annotations of  $H_1$ , and  $O$  be our set of objects. If the properties from Section 3 hold and all facets from  $H_2$  are in  $f(H_1)$ , then the solution  $\theta$  produced by Algorithm RB is consistent.

**Proof:**

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

Since  $\theta$  does not cover  $\delta$ , 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  $\delta$  is not conflict-free, a contradiction.

We know that all facets from  $H_2$  are in  $f(H_1)$ . Since  $\theta(e) = F = f(H_1)$  for all edges  $e$  at the beginning of the algorithm,  $\theta$  initially covers  $\delta$ . Facets are removed from  $\theta$  one by one, and at some point in the algorithm,  $\theta$  no longer covers  $\delta$ . Let  $\delta(e)$  be the first facet of  $\delta$  to be removed from  $\theta(e)$ . That is, before removing  $\delta(e) = g:v$  from  $\theta(e)$ ,  $\theta$  covers  $\delta$ . Just after removing  $\delta(e) = g:v$  from  $\theta(e)$ ,  $\theta$  does not cover  $\delta$ .

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

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,  $\theta_b$  covers  $\delta$ . Since  $\delta$  is structurally-sound, P1 must hold. Then the label of  $\delta(e')$  must be  $g$ , forcing  $\delta(e') = g:v$ , but then  $\delta$  violates P2. Hence  $\delta$  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  $\theta_b$  covers  $\delta$ ,  $\delta$  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  $\delta$  is structurally-sound, P1 must hold. If  $e$  is a sibling of a descendant/ancestor  $d_1$  of the edge  $e'$ , then, since  $\delta$  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  $\theta_b$  covers  $\delta$ ,  $\delta(e') = g:*$ , but since  $e'$  and  $d_0$  are on the same path,  $\delta$  violates P3. Hence  $\delta$  is not structurally-sound, a contradiction.

Thus we have shown that if assignment  $\delta$  is structurally-sound and conflict-free, then  $\theta$  covers  $\delta$ .

We now need to show that if  $\theta$  covers an assignment  $\delta$ , then  $\delta$  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  $\theta$  covers  $\delta$ , assignment  $\delta$  only uses facets actually in  $\theta$ . Thus,  $g:v' \notin S_2(o, \delta) = \cup_{j \in \mathcal{P}_2(o)} \{\delta(j)\}$ .

Thus we have shown that if  $\theta$  covers an assignment  $\delta$ , then  $\delta$  is conflict-free.

■

## 6 Statistics-Based Approach

In contrast to Algorithm RB, our statistics-based Algorithm SB makes decisions based on the popularity of facets in  $H_2$ . The algorithm will treat the number of objects under an edge  $e$  that have a facet  $f$  as evidence that  $e$ 's facet is  $f$ . As we will see, Algorithm SB does not guarantee consistency, but it is better at coping with noisy data.

### 6.1 Statistics-Based Algorithm

Algorithm SB presented below is our statistics-based matching algorithm.

Let us illustrate how Algorithm SB works through an example. Suppose we have  $H_1$  and  $H_2$  as in Figure 1.

We assume in this algorithm that the facets in  $H_2$  come from  $H_1$ . Thus, just as in our example for Algorithm RB, we have:  $F = \{\text{"category: mp3"}, \text{"category: camera"}, \text{"brand: Sony"}, \text{"brand: Epson"}, \text{"s: y"}, \text{"s: n"}\}$ .

In Step 1 of the algorithm, for each  $e \in H_2$  and  $f \in F$ , we initialize the counters:  $c(e, f) := 0$ . We then increment the counters for each object that is under an edge with a given facet. In Table 2 we show the final counts for each edge and facet. The facets have been abbreviated to their initials for compactness.

As an example of the process, let us execute it for edge ?2. Edge ?2 has two objects underneath it:  $b$  and  $c$ . Tracing  $b$ 's path in  $H_1$  we find  $S_1(b) = \{\text{"category: camera"}, \text{"brand: Epson"}, \text{"s: y"}\}$  or equal to  $\{c:c, b:E, s:y\}$  using our abbreviations. Similarly, tracing  $c$ 's path in  $H_1$  we find  $S_1(c) = \{c:m, b:E, s:y\}$ .

After incrementing the counts for edge ?2 for each of the objects we get:  $c(?2, b:E) = 2$ ,  $c(?2, s:y) = 2$ ,  $c(?2, c:c) = 1$ ,  $c(?2, c:m) = 1$ ,  $c(?2, b:S) = 0$ ,  $c(?2, s:n) = 0$ . The maximum value for  $c(?2, f)$  is 2 and this value holds for both  $f = b:E$  and  $f = s:y$ . Thus,  $\theta(?2) = \{b:E, s:y\}$ . The solution  $\theta$  is calculated similarly for the other edges in  $H_2$ .

We see comparing Table 1 and Table 2 that the rule-based algorithm was able to learn more about the facets in  $H_2$  than the statistics-based algorithm

Input: hierarchies  $H_1$  and  $H_2$ , facet annotations  $f(H_1)$ ,  
object set  $O$   
Output: solution  $\theta$   
Step 1: Initialize and increment counters.  
Let  $F :=$  the set of all facets from  $H_1$ .  
For each  $e \in H_2$  and  $f \in F$  do  
    Initialize counter  $c(e, f) := 0$ .  
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  $c(e, f) := c(e, f) + 1$ .  
Step 2: Start from the root and choose winners.  
For each  $e = r_{2-*} \in E_2$  do  
    pick( $e$ ).  
Procedure pick( $e$ ):  
    Set  $\theta(e) := \{\}$ .  
    For  $f \in F$  do  
        If  $c(e, f)$  is maximal, add  $f$  to  $\theta(e)$ .  
    If  $\theta(e) = \{f\}$  then  
        For all descendant edges  $d$  of  $e$   
            Set  $c(d, f) := 0$ .  
    For all child edges  $d$  of  $e$   
        pick( $d$ ).

**Algorithm 2:** Statistics-based matching algorithm (SB)

Edge	c:m	c:c	b:S	b:E	s:y	s:n	Solution
?1	0	1	1	0	1	0	c:c,b:S,s:y
?2	1	1	0	2	2	0	b:E,s:y
?3	0	1	1	0	1	0	c:c,b:S,s:y
?4	0	0	0	0	0	0	c:m,c:c,b:S, b:E,s:y,s:n
?5	0	1	0	1	1	0	c:c,b:E,s:y
?6	1	0	0	1	1	0	c:m,b:E,s:y
?7	0	1	1	0	1	0	c:c,b:S,s:y
?8	0	0	0	0	0	0	c:m,c:c,b:S, b:E,s:y,s:n
?9	0	1	0	1	1	0	c:c,b:E,s:y
?10	0	0	0	0	0	0	c:m,c:c,b:S, b:E,s:y,s:n

**Table 2.** Facet counts and solution in Algorithm SB example.

was. However, the statistics-based algorithm’s robustness to noise will be demonstrated in our experiments.

## 6.2 Consistency of Algorithm SB

Algorithm SB does not guarantee consistent solutions. Below is an example of where Algorithm SB fails to generate a consistent solution.

Let  $H_1$  have three edges; let  $e_1$  and  $e_2$  be the two edges closest to  $r_1$ , and  $e_3$  be the child edge of  $e_1$ . Let  $f(e_1) = g:1$ ,  $f(e_2) = g:2$ , and  $f(e_3) = h:1$ . Let  $H_2$  have only one edge,  $?1$ . Let shared object  $a$  be under edge  $e_3$  in  $H_1$  and under edge  $?1$  in  $H_2$ . Let shared objects  $b$  and  $c$  both be under edge  $e_2$  in  $H_1$  and under edge  $?1$  in  $H_2$ .

The solution  $\theta$  output by Algorithm SB would have  $\theta(?1) = \{g:2\}$  which does not cover  $\delta$  with  $\delta(?1) = h:1$ . even though  $\delta$  is conflict-free and structurally-sound. Thus  $\theta$  fails to cover all good assignments. Also, the covered assignments may have conflicts. (Note in the example that  $\delta(?1) = g:2$  has a conflict.) Thus, in some cases, Algorithm SB fails to generate consistent solutions.

To guarantee that Algorithm SB generates consistent solutions, we need to introduce one additional condition:

**Full facets.** We say hierarchy  $H_1$  has full facets if for every object  $o$  and for every label  $g$ , there exists a facet  $g:*$  with  $g:* \in S_1(o)$ .

**Theorem:** Let  $H_1$  and  $H_2$  be well-formed hierarchies,  $f(H_1)$  be the facet annotations of  $H_1$ , and  $O$  be our set of objects. Let us also assume that  $H_1$  has full facets. Then if the properties from Section 3 hold and all facets from  $H_2$  are in  $f(H_1)$ , then the solution  $\theta$  produced by Algorithm SB is consistent.

**Proof:**

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

Since  $\theta$  does not cover  $\delta$ , we know there exists an edge  $e$  with  $g:v = \delta(e)$  and  $\delta(e) \notin \theta(e)$ . Since  $\delta(e) \notin \theta(e)$ , looking at Step 2 of Algorithm SB, there are two possibilities:

- $c(e, g:v)$  was not maximal: There exists an object  $o$  under edge  $e$  such that  $g:v \notin S_1(o)$ .
- $c(e, g:v)$  was set to 0 because  $\theta(e') = \{g:v\}$  for some ancestor  $e'$  of  $e$ : Facet  $g:v$  was assigned to some edge higher up in the tree.

Let us first consider the first possibility. Suppose there exists an object  $o$  under edge  $e$  such that  $g:v \notin S_1(o)$ . Then since  $H_1$  has full facets, we know there exists  $g:v'$  with  $v' \neq v$  and  $g:v' \in S_1(o)$ . But then  $\delta$  is not conflict-free, a contradiction.

Let us now consider the second possibility. Suppose  $g:v$  was assigned to some edge,  $e'$ , higher up in the tree. Then  $\theta(e') = \{g:v\}$  and since  $\delta$  is structurally-sound, we know that  $\delta(e') \neq g:v$ . Then  $\delta(e') \notin \theta(e')$  for some edge  $e'$  higher up in the tree. By the previous argument, we know that either there exists an

object  $o$  under edge  $e'$  such that  $\delta(e') \notin S_1(o)$  or there exists an edge  $e''$  closer to the root than  $e'$  and with  $\delta(e'') \notin \theta(e'')$ .

Since there are finitely many edges, we know that eventually, we will have to encounter the first possibility, namely that there exists some object  $o_1$  and some edge  $e_1$  with  $o_1$  under  $e_1$  and  $\delta(e_1) \notin f(o_1)$ . But we showed above that this led to a contradiction.

We now need to show that if  $\theta$  covers an assignment  $\delta$ , then  $\delta$  is conflict-free (but may be structurally unsound). Let us prove by contradiction. Assume  $\delta$  is covered by  $\theta$  and has a conflict.

If  $\delta$  has a conflict, there must be an object  $o$  with  $g:1 \in S_1(o)$  and  $g:2 \in S_2(o, \delta)$ . If  $g:2 \in S_2(o, \delta)$ , there must be an edge  $e$  in  $H_2$  with  $\delta(e) = g:2$  and  $o$  is below  $e$ . Since  $\theta$  covers  $\delta$ ,  $\theta(e)$  must include  $g:2$ . This means there must be at least one other object  $o'$  under edge  $e$  such that  $S_1(o')$  includes  $g:2$ . (Otherwise we could not have the count for  $g:2$  be greater than or equal to the count for  $g:1$ .)

Now let us consider what facet  $H_2$  has in reality for edge  $e$ . Clearly this facet cannot be  $g:1$ , since  $o'$  would be misclassified. Similarly, the facet cannot be  $g:2$  because  $o$  would be misplaced. Thus,  $e$  must have some other facet, say  $h:1$ . Since all the object under edge  $e$  have an  $h$  facet (due to full facets condition) and they are all well placed, then the count that  $h:1$  received in Algorithm SB must be equal to the number of objects under  $e$ . Furthermore, this count must be larger than the count that  $g:1$  received (since not all objects have  $g:1$ ) and larger than the count that  $g:2$  received (same reason). Thus, Algorithm SB would not have selected  $g:2$  for  $\theta(e)$ , a contradiction. ■

## 7 Experiments

We have proven that Algorithm RB produces consistent solutions when the data is clean, and that Algorithm SB is only guaranteed to produce consistent solution when the data is clean and  $H_1$  has full facets. In this section we study how well the algorithms perform when the data is noisy. 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). In those cases we would expect Algorithm RB to do very well, and Algorithm SB slightly less well, as it may make incorrect choices when  $H_1$  is not full and there are not enough objects to get accurate statistics. However, as we will see in this section, there are also hierarchies that are not as cleanly structured, so it is important to evaluate how often the algorithms can guess the correct facets for  $H_2$ . Since data can be noisy to different degrees, we also study a parameterized scenario where we can vary the amount of noise and the number of matching objects among the hierarchies, in order to better understand how the algorithms cope with noise.

## 7.1 GLUE Hierarchies

We start with a specific hierarchy-matching scenario used by the GLUE System [5]. There are two indexes (hierarchies) of companies, one obtained from Yahoo.com,  $H_2$ , and the other from 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.

These hierarchies deal with ambiguous categories. The correct mappings between facets (needed to evaluate results) are unclear, so to develop a gold standard we use the techniques of [5] (Jaccard similarity) to infer matches. Matches found in this fashion are called “correct” and in the gold standard  $G$  (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 in  $H_2$ . 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). As we can see, this scenario is extremely noisy, so we cannot expect any algorithm based on object-matching to do very well.

On this data set, we ran Algorithms RB and SB. Table 3 gives the fraction of decided edges (metric  $C(\theta)$  of Section 4.1) in each case. As expected, the performance of Algorithm RB is very poor: Since it is trying to reason about how  $H_2$  is organized, it gets confused by all the incorrectly placed objects. However, Algorithm SB does surprisingly well in spite of the extreme noise: it can correctly determine over 40% of the facets. Algorithm SB can do well because there are still enough correctly-placed objects so that statistics can identify the correct facets. Keep in mind that our  $C(\theta)$  metric is conservative and gives no partial credit.

Also keep in mind that even a 40%  $C(\theta)$  value can be very useful when combined with textual facet matching that we are not considering in our algorithms. That is, as discussed in the Introduction, matching based on common objects is complementary to matching based on textual similarity of facets. The facet choices discovered by our algorithms can be used to simplify the string matching problem, potentially leading to results that are better than what each technique can get on its own.

In addition to testing Algorithms RB and SB, we also tested Algorithm SB again, this time giving the algorithm the hint that facets from the top level from  $H_1$  should match up to edges in the top level of  $H_2$ , and that facets from the bottom level from  $H_1$  should match up to edges in the bottom level of  $H_2$ . Although this hint does not hold true for all edges, it significantly improved our performance, from 50 correct edges to 84. This improved performance suggests

	correct edges	% correct
Algorithm RB	1	0.8%
Algorithm SB	50	43.9%
Algorithm SB with Hint	84	73.7%

**Table 3.** GLUE experimental results.

that Algorithm SB can be improved by rules, and that a good hybrid algorithm might be most desirable.

On the same data set GLUE [5] achieved 70% matching accuracy. It is important to note that GLUE used different information to perform the matches; GLUE used the content of the objects (text), and the text labels in the hierarchies, whereas we used knowledge of shared objects. Knowledge of shared objects can yield complementary information, that will be especially valuable when the text labels are not as useful (e.g., if the labels are in different languages or represented by images and not text). Once we understand how to exploit objects (Algorithms RB, SB), our eventual goal is to combine textual and object information to obtain better performance than what each technique can achieve on its own.

## 7.2 Dmoz Experiment

We also ran an experiment in which we matched two sub-hierarchies related to museums extracted from the Open Directory (located at <http://dmoz.org>), a directory maintained by volunteers. Results are not included here due to space limitations, but are in a technical report.

## 7.3 IMDb Synthetic Experiments

Our two scenarios tell us how our algorithms perform in two specific cases, but do not provide as much insight as we would like into the reasons. To better understand how noise, the number of shared objects and other parameters affect performance, we built synthetic hierarchies where we could control parameters. The data used was real, from the Internet Movie Database, but we constructed our own hierarchies based on the data.

To synthesize hierarchies from IMDb, we first wrote a script that downloaded the descriptions of 2410 movies. The script downloaded movie descriptions based on random integer IDs to obtain a representative distribution.

Given the movie descriptions, we then assigned facets to the movies. The three labels we used were title, genre, and year. The values used for each label and the number of corresponding objects are given below. (Note on genre: If a movie was associated by IMDb with multiple genres, we picked one of these genres at random to create the corresponding facet.)

- Title: A-E: 927; F-O: 622; P-Z: 861.

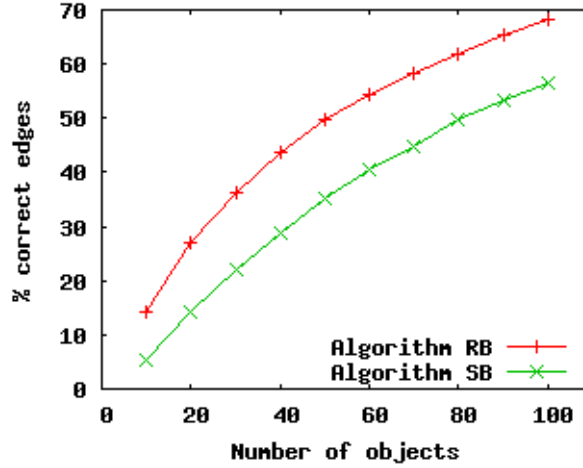


Fig. 3. Performance vs. shared objects with no noise (IMDb).

- Genre: drama: 509; comedy: 430; short: 245 documentary: 201; other: 1025.
- Year: before 1959: 419; 60s: 220; 70s: 223; 80s: 249; 90s: 440; 2000- : 859.

Given our objects with facets, let us now describe how we generated the synthetic hierarchies.

- Select at random two orderings from root to leaves of the label set ( $\{\text{title, genre, year}\}$ ).
- Create a hierarchy  $H_1$  based on one of the orderings; create  $H_2$  with the second ordering.
- Create a set of  $X$  shared objects at random based on the representative distribution given by the 2410 objects from IMDb.
- Place each of the generated shared objects into the appropriate positions in both hierarchies.

The hierarchies generated in this fashion have no noise and can be used to evaluate how the algorithms perform in well-structured scenarios. (Note that the generated  $H_1$  hierarchies have the full facets property, which is beneficial to Algorithm SB.) Figure 3 shows the performance on the RB and SB algorithms. The horizontal axis is the number of shared objects  $X$ , while the vertical axis is the percent of edges correctly decided,  $C(\theta)$ . Each point represents the average  $C(\theta)$  over 200 runs, where in each run we use different hierarchy pairs with the same  $X$  value.

As we can see in Figure 3, in an environment with no noise, Algorithm RB consistently outperforms Algorithm SB. For instance, with 100 objects shared among the hierarchies, Algorithm RB is able to correctly determine 69% of the  $H_2$  facets whereas Algorithm SB is only able to determine 58% of the facets. Also, as we would expect, the more shared objects in our hierarchies, the more we are able to learn about the facets in  $H_2$ .

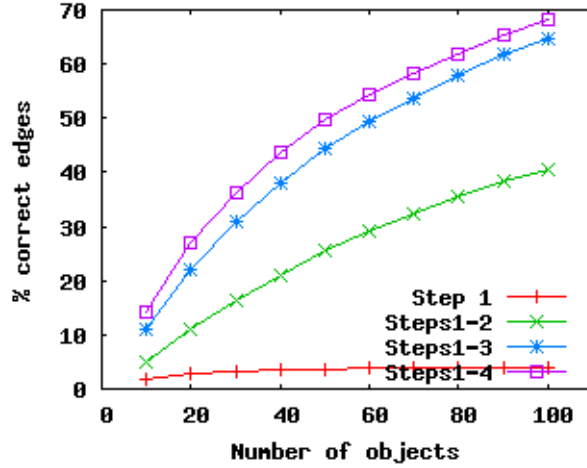


Fig. 4. Algorithm RB’s performance vs. shared objects with no noise (IMDb).

As discussed in Section 5, Algorithm RB can be run in a conservative mode where it does not assume that some or all of the hierarchy properties hold. To understand the implications of being conservative, in Figure 4 we show the performance of Algorithm RB if some of its property-enforcement steps are not executed. The curve labeled “Steps1-4” shows performance ( $C(\theta)$ ) if all four steps are performed, i.e., for the full algorithm. The results are the same as the RB results in Figure 3.

The curve labeled “Steps1-3” in Figure 4 runs RB only through Step 3, skipping Step 4. Since Step 4 enforces Property P3, this version of the algorithm is being conservative by *not* assuming that P3 holds. In this case, hierarchy  $H_2$  does satisfy P3, so the difference between the “Steps1-4” and the “Steps1-3” curves shows the cost of being conservative. But if  $H_2$  did not satisfy P3, then the difference would simply show that we can learn less about  $H_2$  when P3 does not hold.

The curve for “Steps1-2” skips Step 3 that enforces property P1 (as well as skipping Step 4), while the curve for “Step 1” skips Step 2 that enforces property P2 (as well as skipping Steps 3 and 4). The utility of these properties, at least to Algorithm RB, is again illustrated by the gaps. In summary, we see that Algorithm RB is less effective in scenarios where some or all of the properties do not hold.

#### 7.4 Noise

To study the performance of Algorithms RB and SB under different levels of noise (misplaced objects), we generated synthetic IMDB hierarchies with errors, as follows. We first generated the  $H_1$  and  $H_2$  hierarchies with  $X$  shared objects in the same manner as we did in the no-noise scenario. Then we took  $k$  objects

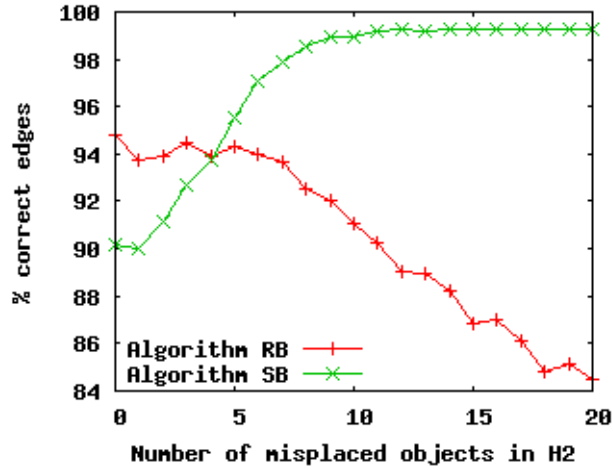


Fig. 5. Performance vs. number of misplaced objects (IMDb).

in  $H_2$ , which at this time is noise-free, and for each of the  $k$  objects, we did the following:

- Randomly chose a label and randomly chose a value that can be used with this label.
- Moved this object to the  $H_2$  position that corresponds to the facet and the new value. Note that only one facet of each object was modified, that is, only one facet per object was misclassified in  $H_2$ .

We first generated hierarchies with no noise and  $X = 500$  shared objects generated using the IMDb distribution. We then varied  $k$ , the number of misplaced objects, from 0 to 20, and for each value of  $k$ , we compared the performance of the Rule-Based Algorithm and the Stats-Based Algorithm. Figure 5 shows our results. Again, each point shows the average over 200 scenarios with the same parameters.

We see that, as we saw earlier, with no misplaced objects, Algorithm RB outperforms the Algorithm SB. However, Algorithm SB is more robust and as we have more misplaced objects, the gap between the two algorithm closes. In this set of experiments, for  $k$  greater than about 5, Algorithm SB outperforms RB.

Note that the performance of Algorithm SB in Figure 5 is rather surprising: the algorithm actually performs better as the number of misplaced objects increases! The reason is that the misplaced objects populate regions of  $H_2$  that do not have shared objects, and help identify facets in those regions.

In particular, note that we generated the shared objects in the unmodified  $H_2$  using the representative distribution from IMDb. Thus, there are some combinations of facets that are more common than others, and some  $H_2$  regions are left with no objects. Thus, when we take objects and randomly replace one of their facets, we might move them to leaves in the hierarchy without any shared

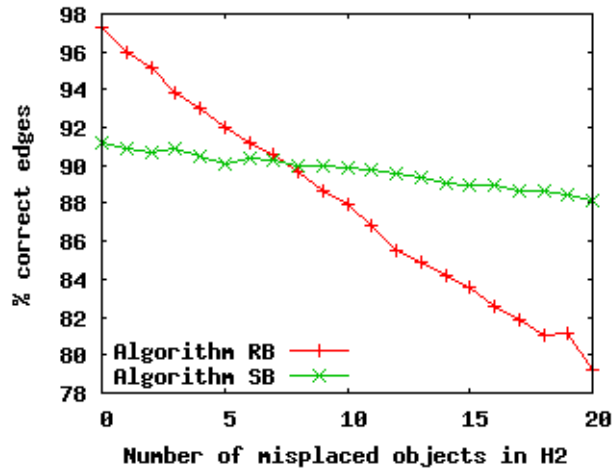


Fig. 6. Performance vs. number of misplaced objects (uniform distribution).

objects. In this manner, we can determine the facet at the edge closest to the leaf, and if there are enough correctly placed objects elsewhere, then the facet that was randomly switched might do us no harm.

To verify that this behavior would not happen in a uniform distribution, we ran a similar experiment to the one above, except this time, we used 200 objects instead of 500, and the objects were randomly taken from a uniform distribution as opposed to a distribution based on IMDb. The results are shown in Figure 6. As we can see in Figure 6, having drawn from a uniform distribution of objects, the performances of both algorithms decrease as we increase the number of misplaced objects. And again in this case we see that Algorithm RB is superior when there are few misplaced objects, but SB is better at coping with high noise.

## 8 Related Work

Our work is relevant to schema/ontology matching ([5],[8],[10],[13],[14],[16]) 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.

There are other works ([1], [15]) that consider the data instances as organized in a flat set of categories. With our data organized in hierarchies, we seek to exploit the hierarchical structure to infer matches between facets.

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)uses explicit knowledge of shared objects as an input to the algorithm; (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.

We explore how to leverage shared object identities in faceted hierarchies to find semantic matches between the facets. Agrawal's On integrating catalogs uses shared object identities to merge the catalogs, but for simplification, they flatten the hierarchies before doing any matching. Our approach is designed to find semantic matches in situations beyond those in which the leaves have a 1-1 correspondence. Hearst's Flamenco Project is one example of a user interface project that explores faceted search. We believe that in the domain of faceted hierarchies, using object identities to find facet matches is relatively unexplored.

## 9 Conclusion

We have presented both a rule-based (RB) and a statistics-based (SB) approach for matching hierarchies based on shared objects. Both matching algorithms return, for each of the edges in the new hierarchy, a set of candidate facets from the base hierarchy. Our two algorithms present an interesting set of tradeoffs: Algorithm RB is guaranteed to return consistent answers when the data is clean, and even outperforms Algorithm SB when there are a few misplaced objects. On the other hand, Algorithm SB can cope better with high noise situations, but it does not perform as well with clean or almost clean data. As a matter of fact, Algorithm SB can return inconsistent solutions even with clean data, if hierarchy  $H_1$  is not full and if there are not enough shared objects so the statistics become useful.

We believe that our object-matching approach will be most useful when used in conjunction with a more traditional textual approach that compares labels and values. For instance, if our algorithms can narrow down the choices for an edge to 2 or 3, a textual approach may have an easier time deciding among those three choices than among all choices. Analogously, if textual analysis can yield probabilities that facets match, those estimates can be folded into our object-based statistics to improve the quality of overall results. We are currently investigating such hybrid approaches to hierarchy matching.

## 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*, pages 805–810, 2005.
9. R. Ichise, H. Takeda, and S. Honiden. Rule induction for concept hierarchy alignment. In *Proceedings of the IJCAI-01 Workshop on Ontology Learning*, 2001.
10. Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *Knowledge Engineering Review*, 18(1):1–31, 2003.
11. 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.
12. 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.
13. 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.
14. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001.
15. 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.
16. 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.