# iMAP: Discovering Complex Semantic Matches between Database Schemas

**Robin Dhamankar, Yoonkyong Lee, AnHai Doan**
Department of Computer Science
University of Illinois, Urbana-Champaign, IL, USA
{dhamanka,ylee11,anhai}@cs.uiuc.edu

**Alon Halevy, Pedro Domingos**
Department of Computer Science and Engineering
University of Washington, Seattle, WA, USA
{alon,pedrod}@cs.washington.edu

## ABSTRACT

Creating semantic matches between disparate data sources is fundamental to numerous data sharing efforts. Manually creating matches is extremely tedious and error-prone. Hence many recent works have focused on automating the matching process. To date, however, virtually all of these works deal only with one-to-one (1-1) matches, such as address = location. They do not consider the important class of more complex matches, such as address = concat(city,state) and room-price = room-rate * (1 + tax-rate).

We describe the iMAP system which semi-automatically discovers both 1-1 and complex matches. iMAP reformulates schema matching as a *search* in an often very large or infinite match space. To search effectively, it employs a set of searchers, each discovering specific types of complex matches. To further improve matching accuracy, iMAP exploits a variety of domain knowledge, including past complex matches, domain integrity constraints, and overlap data. Finally, iMAP introduces a novel feature that generates explanation of predicted matches, to provide insights into the matching process and suggest actions to converge on correct matches quickly. We apply iMAP to several real-world domains to match relational tables, and show that it discovers both 1-1 and complex matches with high accuracy.

## 1. INTRODUCTION

Semantic mappings specify the relationships between data stored in disparate sources. They lie at the heart of any data sharing architecture, be it a data integration system, data warehouse, peer-data management system or web-service based architecture. Data sharing systems are crucial for supporting a wide range of applications, such as enterprise data integration, scientific collaborations, data management on the WWW, and cooperation between government agencies. Currently, semantic mappings are created by hand (typically supported by advanced graphical user interfaces), and in practice are extremely tedious and error-prone [26].

The problem of semi-automatically creating mappings has received significant attention recently in both the database and AI communities (see [24] for a recent survey, and [13, 12] for several works since). The bulk of the work focused on the first phase of mapping, called *schema matching*. A *match* between two schemas specifies semantic correspondences between elements of both schemas [24]. These correspondences are later elaborated (e.g., using a system such as Clio [29]) to generate the mapping. For example, the mapping can be in the form of a SQL query that translates data from one source to another. It should be noted that both the process of generating matches and mappings are expected to involve human input.

To date, the work on schema matching has focused on discovering 1-1 matches between schema elements (e.g., relation attributes, XML tags). For example, a 1-1 correspondence would specify that element location in one schema matches area in the other, or that agent-name matches name.

While 1-1 matches are common, relationships between real-world schemas involve many *complex matches*. A *complex match* specifies that a combination of attributes in one schema corresponds to a combination in the other. For example, it may specify that list-price = price * discount-rate and address = concat(city,state). In fact, in the schemas we consider in Section 6, complex matches compose up to half of the matches. Hence, the development of techniques to semi-automatically construct complex matches is crucial to any practical mapping effort.

Creating complex matches is fundamentally harder than 1-1 matches for the following reason. While the number of candidate 1-1 matches between a pair of schemas is bounded (by the product of the sizes of the two schemas), the number of candidate complex matches is not. There are an unbounded number of functions for combining attributes in a schema, and each one of these could be a candidate match. Hence, in addition to the inherent difficulties in generating a match to start with, the problem is exacerbated by having to examine an unbounded number of match candidates.

This paper describes the iMAP system which semi-automatically discovers both 1-1 and complex matches between database schemas. Currently, iMAP considers matches between relational schemas, but the ideas we offer can be generalized to other data representations. Developing iMAP required several innovations.

**Generating Matches:** To address the problem of examining an unbounded number of match candidates, we view the generation of complex matches as a *search* in the space of possible matches. To search the space effectively, we employ a set of search modules, called *searchers*, each of which considers a meaningful subset of the space, corresponding to specific types of attribute combinations.

As examples of searchers, a *text searcher* may consider

*SIGMOD 2004* June 13-18, 2004, Paris, France.
Copyright 2004 ACM 1-58113-859-8/04/06 . . . $5.00.

only matches that are concatenations of text attributes, while a *numeric searcher* considers combining attributes with arithmetic expressions. A *schema mismatch* searcher examines complex matches that involve the data instances of one schema and the schema elements of the other (such complex matches have been observed to be very common in practice [20]). Finally, a *date searcher* focuses on complex matches that involve date attributes, such as date = concat(month,"/",year).

We use *beam search* [25] to control the search through the space of candidate matches. To *evaluate* the quality of each match candidate, we employ a set of techniques, including machine learning, statistics, and heuristic methods. Since the number of match candidates is often infinite, a key challenge in adapting search to the matching context is that we do not know when the best match has been found and thus the search should be terminated. We develop a simple termination criterion based on the diminishing-returns principle and show that it is often very effective in practice.

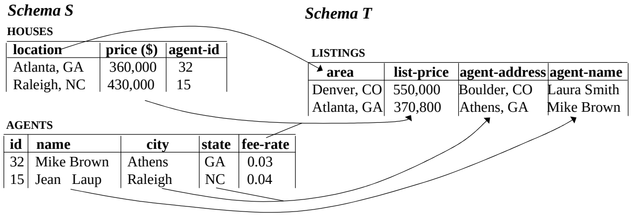
Given the matches produced by the search modules, iMAP evaluates their quality further, using criteria that are impractical to be employed at the searcher level. For example, a numeric searcher uses mathematical transformations commonly employed in the equation discovery area [27] to quickly generate a ranked list of match candidates. iMAP then re-ranks the candidates, using also the *name similarity* between the attributes involved in a match. In the final step, iMAP selects the best matches from the re-ranked candidates, taking into account domain knowledge and integrity constraints.

**Exploiting Domain Knowledge:** Several recent works [9, 8, 16] have noted the benefits of exploiting domain knowledge for schema matching. Intuitively, domain knowledge (e.g., keys) are useful for pruning some candidate matches. We show that in the context of complex matches the potential benefits of using domain knowledge are even greater: not only can domain knowledge be used to evaluate the accuracy of a proposed match, but it can also be used to prune which match candidates are even considered in the search phase.

In addition to exploiting domain knowledge in the form of integrity constraints and knowledge gleaned by learning from previous matches, iMAP exploits two new kinds of domain knowledge. First, if the databases being matched share some tuples, iMAP can utilize this *overlap data* to discover complex matches. Second, iMAP also exploits external data in the domain. For example, it can mine real estate listings to learn that the number of real estate agents in a specific area is bounded by 50. Now given the match agent-name = concat(first-name,last-name), where first-name and last-name belong to the home owner, iMAP can examine the data instances associated with the match to realize that concat(first-name,last-name) yields hundreds of distinct names, and hence is unlikely to match agent-name.

Finally, one of the important aspects of iMAP is that it tries to use domain knowledge as early as possible, in order to prune the consideration of matching candidates.

**Explaining Match Predictions:** As schema matching systems employ more sophisticated techniques, the reasons for the predictions they make become rather involved. The complexity of the decisions is even more pronounced in the context of complex matches, where the prediction for a complex match may depend on other predictions made for simpler or 1-1 matches.



*Schema S*

HOUSES

| location | price ($) | agent-id |
|---|---|---|
| Atlanta, GA | 360,000 | 32 |
| Raleigh, NC | 430,000 | 15 |

AGENTS

| id | name | city | state | fee-rate |
|---|---|---|---|---|
| 32 | Mike Brown | Athens | GA | 0.03 |
| 15 | Jean Laup | Raleigh | NC | 0.04 |

*Schema T*

LISTINGS

| area | list-price | agent-address | agent-name |
|---|---|---|---|
| Denver, CO | 550,000 | Boulder, CO | Laura Smith |
| Atlanta, GA | 370,800 | Athens, GA | Mike Brown |

Figure 1: The schemas of two relational databases on house listing, and the semantic mappings between them.

In iMAP we introduce a new feature that helps a human designer interacting with the system. We show how the system can offer an *explanation* of a predicted match, and we consider several fundamental types of explanations, such as knowing why a particular match is or is not created, and why a certain match is ranked higher than another.

In summary, this paper makes the following contributions:

- An architecture for semi-automatically discovering complex matches that combines search through a set of candidate matches and methods for evaluating each match in isolation, and a set of matches as a whole.
- Uses of new kinds of domain knowledge (overlap data and mining external data), and applying the knowledge as early as possible in the matching process.
- A mechanism for explaining the decisions made by the matching system.
- The iMAP system which embodies all these innovations, and a set of experiments on real-world schemas that illustrate the effectiveness of the system. Our experiments show that we can correctly match 43-92% of the complex matches in the schemas we considered.

The paper is organized as follows. Section 2 defines the schema matching problem. Sections 3-5 describe the iMAP system. Section 6 presents our experiments and discusses the current system's limitations. Section 7 reviews related work and Section 8 concludes.

## 2. PROBLEM DEFINITION

We discuss schema matching in terms of relational schemas, but the ideas we offer here carry over to other data representations (e.g., matching XML schemas and DTDs). As a running example, consider the two relational schemas $S$ and $T$ in Figure 1. Both databases store house listings and are managed by two different real-estate companies. The schema of database $T$, for example, has one table, LISTINGS, whereas database $S$ stores its data in two tables, HOUSES and AGENT.

Suppose the two real-estate companies have decided to merge. To cut costs, they eliminate database $S$ by transferring all house listings from $S$ to database $T$. Such data transfer is not possible without knowing the semantic mappings between the relational schemas of the databases. Below we show some of the mappings for the individual attributes of $T$, using SQL notation. Together, they specify how to create tuples for $T$ from data in $S$. In general, a variety of approaches have been used to specify semantic mappings (e.g., SQL, XQuery, GAV, LAV, GLAV [14]).

```
         area = SELECT location from HOUSES
agent-address = SELECT concat(city, state) FROM AGENTS
   list-price = SELECT price * (1 + fee-rate)
                FROM    HOUSES, AGENTS
                WHERE  agent-id = id
```

The process of creating mappings typically proceeds in two steps. In the first step, called *schema matching*, we find matches (a.k.a. *correspondences*) between elements of the two schemas. In the second step we elaborate the matches to create query expressions (as above) that enable automated data translation or exchange. The majority of the work in the area has considered algorithms for schema matching, with the significant exception of Clio [29], which is a nice example of a system that studies the second step of the process. We note that both steps of schema mapping may involve interaction with a designer. In fact, the goal of a schema mapping system is to provide a design environment where a human can quickly create a mapping between a pair of schemas. The human builds on the system's suggestions where appropriate, and provides the system with feedback to direct it to the appropriate mapping.

There are two kinds of schema matches. The first, and the topic of the vast majority of past works on schema matching, is *1-1 matches.* Such matches state that there is a correspondence between a pair of attributes, one in each schema. For example, attribute area in $T$ corresponds to attribute location in table HOUSES of $S$.

The second kind, *complex matches*, specify that some combination of attributes in one schema corresponds to a combination in the other. In our example, an instance of agent-address in $T$ is obtained by concatenating an instance of city and an instance of state in table AGENTS of schema $S$.

Complex matches may involve attributes from *different* tables. For example, list-price is obtained by the following combination of attributes: price $*\,(1 +$ fee-rate$)$. However, in order to obtain the appropriate pair of price and fee-rate, we need to specify that tables HOUSES and AGENTS be joined by HOUSES.agent-id $=$ AGENTS.id. In fact, discovering such join relationships was usually postponed to the second step of schema mapping [29].

In this paper we describe the iMAP system which semi-automatically discovers complex matches for relational data. Initially, our goal was to discover complex matches that involve only attributes in a single table. However, by casting the problem of finding complex matches as search, we are sometimes able to find matches that combine attributes from multiple tables (and suggesting the appropriate join path, see Section 3.1.1). The key challenge that iMAP faces is that the space of possible match candidates is unbounded, corresponding to all the possible ways of combining attributes in expressions.

# 3. THE IMAP ARCHITECTURE

In our explanation of iMAP, we assume we are trying to find matches from a source schema (in our case $S$) to a target schema ($T$). In practice, we would typically generate matches in both directions.

The iMAP architecture is shown in Figure 2. It consists of three main modules: match generator, similarity estimator, and match selector. The *match generator* takes as input two schemas $S$ and $T$. For each attribute $t$ of $T$, it generates a set of match candidates, which can include both 1-1 and complex matches. As we explain below, the generation is guided by a set of search modules. The *similarity estimator* then computes for each match candidate a score which indicates the candidate's similarity to attribute $t$. Thus, the output of this module is a matrix that stores the similarity score of $\langle$target attribute, match candidate$\rangle$ pairs. Finally, the *match selector* examines the similarity matrix and outputs the best matches for the attributes of $T$.

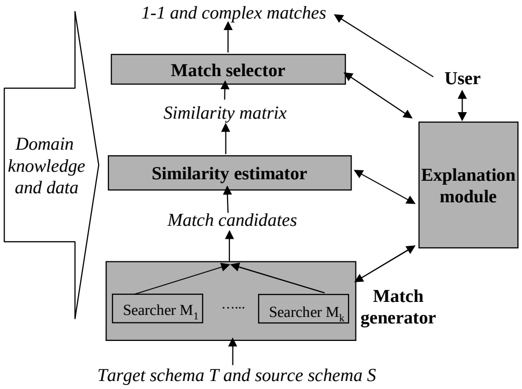


Figure 2: The iMAP architecture

During the entire matching process, the above three modules also exploit *domain knowledge and data* to maximize matching accuracy, and interact with an *explanation module* to generate explanations for matches. The rest of this section describes the three modules. We discuss the use of domain knowledge in Section 4 and the explanation facility in Section 5.

## 3.1 Candidate Match Generation

Given an attribute in the target schema, the match generator must quickly discover a relatively small set of promising candidate matches for the attribute. The key idea underlying the match generator is that it recasts this discovery process as a *search* through the space of possible match candidates.

The space of match candidates can be extremely large or even infinite, since any expression for combining attributes of the source schema can potentially be a match candidate. Some combinations make sense (e.g., concat(city,state)), and others do not (e.g., (price - agent-id) * 2). The match generator addresses this challenge by employing a set of special-purpose searchers. A searcher explores a specialized portion of the search space, based on knowledge of particular combination operators and attribute types.

The set of best match candidates is given by the *union* of the match candidates returned by the specialized searchers. The following example illustrates searchers:

Example 3.1. Consider the text searcher and the numeric searcher. Given a target attribute $t$, the text searcher examines the space of all matches that are either attributes or *concatenations* of attributes in source schema $S$, to find a small set of matches that best match attribute $t$. The text searcher accomplishes this by analyzing the textual properties of the attributes of the two schemas. In the case of target attribute agent-address (Figure 1), this searcher may return the following matches, in decreasing order of confidence: concat(city,state), location, concat(name,state).

The numeric searcher exploits the values of numeric attributes to find matches that are *arithmetic expressions* over the attributes of source schema $S$. Given target attribute list-price in Figure 1, this searcher may return the following matches: price $*$ (1 + fee-rate), price, price + agent-id. □

In general, a searcher is applicable to only certain types of attributes. For example, the text searcher examines the *concatenations* of attributes, and hence is applicable to only *textual* ones. Except for data-type information that is available in the schema, this searcher employs a set of heuristics to decide if an attribute is textual. The heuristics examine the ratio between the number of numeric and non-numeric characters, and the average number of words per data value. As another example, the numeric searcher examines arithmetic expressions of attributes, and hence is applicable to only *numeric* attributes.

There are two main benefits to using multiple searchers in iMAP. First, the searchers enable considering a small and meaningful part of the space of candidate matches. Second, the system is easily extensible with additional searchers. For example, if we later develop a specialized searcher that finds complex matches for address attributes, then we can just plug the searcher into the system. In addition, particular domains are likely to benefit from specialized searchers for the domain.

### 3.1.1 The Internals of a Searcher

Applying search to candidate generation requires addressing three issues: search strategy, evaluation of candidate matches, and termination condition.

**Search Strategy:** Even the specialized search space of a searcher, such as the space of concatenations of the text searcher, could still be very large or even unbounded. Hence, we face the challenge of efficiently searching such spaces. In iMAP we propose to address this problem using a standard search technique called *beam search* [25]. The basic idea behind beam search is that it uses a *scoring function* to evaluate each match candidate, then at each level of the search tree, it keeps only $k$ highest-scoring match candidates, where $k$ is a pre-specified number. This way, the searcher can conduct a very efficient search in any type of search space.

**Match Evaluation:** To conduct beam search, given a match candidate such as concat(city,state), we must assign to it a score which approximates the semantic distance between it and the target attribute (say agent-address). In iMAP, we use a range of techniques, including machine learning, statistics, and heuristics, to compute such candidate scores (see Section 3.1.2 on implemented searchers).

For example, to use learning techniques, we build a classifier for target attribute agent-address using the data in target schema $T$, then apply it to classify candidate match concat(city,state). The classifier returns a confidence value which we can assign to be the candidate match's score.

**Termination Condition:** Since the search space can be unbounded, we need a criterion for deciding when to stop the search. Our criterion is based on terminating when we start seeing *diminishing returns* from our search. Specifically, in the $i$th iteration of the beam search algorithm, we keep track of the highest score of any candidate matches (that have been seen up to that point), denoted by $Max_i$. Then if the difference in the values of $Max_i$ and $Max_{i+1}$ (i.e., two consecutive iterations) is less than a pre-specified threshold $\delta$, we stop the search and return the $k$ highest-scoring match candidates as the most promising match candidates.

The following example illustrates the text searcher in more detail.

Example 3.2. Given a target attribute such as agent-address, the text searcher begins by considering all 1-1 matches, such as agent-address = location, agent-address = price, and so on (see Figure 1).

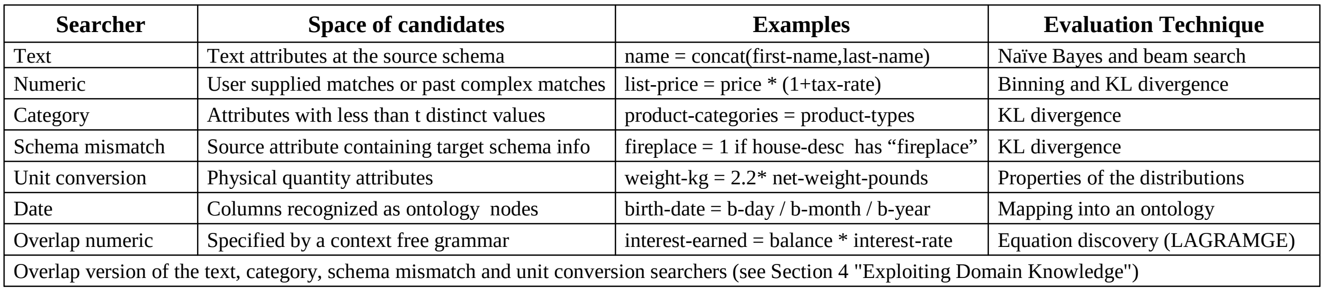
The text searcher then computes a score for each of the above matches. Consider the match agent-address = location. The searcher assembles a set of training examples for agent-address: a data instance in target schema $T$ is labeled "positive" if it belongs to agent-address and "negative" otherwise. Next, the searcher trains a Naive Bayes text classifier [10] on the training examples to learn a model for agent-address. The data instances are treated as text fragments in this training process. The searcher then applies the trained Naive Bayes text classifier to each data instance of attribute location (in source schema $S$) to obtain an estimate of the probability that that data instance belongs to agent-address. Finally, the searcher returns the average of the instance probabilities as the desired score.

After computing the scores for the 1-1 matches, the text searcher begins a beam search. It picks the $k$ highest-scoring 1-1 matches, then generates new matches by concatenating each of the $k$ matches with each attribute in $S$. For example, if agent-address = city is picked, then agent-address = *concat*(city, state) is generated as a new match. The searcher then computes the scores of the new matches as described above. In general, the score of match $t = f(s_1, \ldots, s_n)$ is computed by comparing the column corresponding to $t$ and the "composite column" corresponding to $f(s_1, \ldots, s_n)$, and the comparison is carried out using the Naive Bayes text classifier. The searcher then picks the $k$ best matches among all matches it has seen so far, and the process repeats until the diminishing-returns condition sets in, as described earlier. □

**Handling Join Paths:** Recall from Section 2 that a complex match can involve join paths. In our example, for the match list-price = price * (1 + fee-rate), we need to discover that price and fee-rate should be joined via HOUSES.agent-id = AGENTS.id.

We can find join paths for complex matches as follows. First, for each set of tables in $S$, iMAP finds all possible join paths that can relate the tables. Note that the set of reasonable join paths for any group of tables is typically small, and can be discovered using a variety of techniques, including analyzing joins in queries that have been posed over the schemas and examining the data associated with the schemas [5]. The user can also suggest additional join paths for consideration.

Once iMAP has identified a small set of join paths per each group of tables, it modifies the search process to take the join paths into consideration. Consider the text searcher. Suppose it is in the process of generating new candidate matches. The current match is $t = concat(a, b)$, where $a$ and $b$ are attributes of table $X$ in schema $S$. Now suppose the searcher is considering attribute $c$ of table $Y$ (also of $S$). Suppose iMAP has determined that tables $X$ and $Y$ can join via paths $j_1$ and $j_2$. Then the text searcher would create two candidate matches: $concat(a, b, c)$ with $a, b, c$ relating via $j_1$, and $concat(a, b, c)$ with $a, b, c$ relating via $j_2$. When materialized, each of the above two matches will likely form a different column of values, due to using different join paths.



| Searcher | Space of candidates | Examples | Evaluation Technique |
|---|---|---|---|
| Text | Text attributes at the source schema | name = concat(first-name,last-name) | Naïve Bayes and beam search |
| Numeric | User supplied matches or past complex matches | list-price = price * (1+tax-rate) | Binning and KL divergence |
| Category | Attributes with less than t distinct values | product-categories = product-types | KL divergence |
| Schema mismatch | Source attribute containing target schema info | fireplace = 1 if house-desc has “fireplace” | KL divergence |
| Unit conversion | Physical quantity attributes | weight-kg = 2.2* net-weight-pounds | Properties of the distributions |
| Date | Columns recognized as ontology nodes | birth-date = b-day / b-month / b-year | Mapping into an ontology |
| Overlap numeric | Specified by a context free grammar | interest-earned = balance * interest-rate | Equation discovery (LAGRAMGE) |
| Overlap version of the text, category, schema mismatch and unit conversion searchers (see Section 4 "Exploiting Domain Knowledge") | | | |

Table 1: Implemented searchers in iMAP.

### *3.1.2 Implemented Searchers*

Table 1 characterizes the searchers currently implemented in iMAP. The searchers cover a variety of complex match types (text, numeric, category, etc.). They employ diverse techniques to evaluate match candidates, and can exploit several forms of domain knowledge, such as domain constraints and overlap data. In what follows we describe the searchers that do not exploit overlap data. The rest will be described in Section 4. We do not discuss the text searcher any further.

**Numeric Searcher:** This searcher finds the best match for a target attribute judged to be numeric, such as lot-area. Building it raises the problem of how to compute the similarity score of a complex match, such as lot-dimension1 * lot-dimension2. We address this problem by considering the similarity between two *value distributions*: those of the values observed in column lot-area, and the values of the “composite column” created by “materializing” lot-dimension1 * lot-dimension2. We compute the similarity of value distributions using the Kullback-Leibler divergence measure [4, 18] (which has previously been used in other contexts, such as statistical natural language processing [18]).

The second problem we face is the type of matches the numeric searcher should examine. The searcher cannot consider an arbitrary space of matches, because this will likely lead it to overfit the data and find an incorrect match. Hence, we limit the numeric searcher to consider only a restricted space of common matches, such as those that add, subtract, multiply, or divide two columns. In Section 4 we discuss how the numeric searcher can exploit past complex matches or overlap data to find much more expressive matches, such as price * quantity * (1 + fee-rate).

**Category Searcher:** This searcher finds “conversion” mappings between categorical attributes, such as waterfront = $f$(near-water), where $f(“yes'') = 1$ (i.e., a data instance “yes” of near-water is converted into an instance “1” of waterfront) and $f(“no'') = 0$. Given a target attribute $t$, the searcher determines if $t$ is categorical, by counting the number of distinct values of $t$, and verifying that this number is below a threshold (currently set at 10). Next, it looks for category attributes on the source schema side, using the same technique. The searcher then discards the category source attributes whose number of distinct values are not the same as that of $t$, or whose similarity to $t$ is low (where similarity is computed using the Kullback-Leibler measure on the two value distributions).

Let the remaining category source attributes be $s_1, \ldots, s_p$. Then for each $s_i$ the searcher attempts to find a conversion function $f_i$ that transforms the values of $s_i$ to those of $t$. Currently, the function $f_i$ that the searcher produces maps the value with the highest probability in the distribution of $s_i$ to that in the distribution of $t$, then the value with the second highest probability in the distribution of $s_i$ to that in the distribution of $t$, and so on. The searcher then produces as output attributes $s_1, \ldots, s_p$ together with the conversion functions $f_1, \ldots, f_p$.

**Schema Mismatch Searcher:** Schema-mismatch matches relate the *data* of a schema with the *schema* of the other. In the current iMAP implementation we focus on a particular type: a binary target attribute matches the data in a source attribute. For example, if a data instance of source attribute house-description contains (does not contain) the term “fireplace”, then the corresponding instance of target attribute fireplace is “yes” (“no”). This schema-mismatch type occurs very frequently in practice (e.g., product description, course listing). The fundamental reason is that often one schema chooses to mention *a particular property* (e.g., fireplace, zoom capability, hard copy edition) of an entity in the data, but another schema chooses to create an attribute modeling that property.

Given a target attribute $t$, this searcher determines if $t$ is a binary category attribute (using the same technique as in the category searcher). Next, it searches for the presence of the name of $t$ in the data instances of source attributes. If this name appears at least $p$ times (currently set at 5) in the data of $s$, then there may exist a schema mismatch between $t$ and $s$. The searcher then transforms $s$ into a category attribute $s'$, such that each data instance of $s$ is transformed into “1” if it contains the name of $t$, and 0 otherwise. Next, the searcher creates a conversion function $f$ that transforms data values of $s'$ into those of $t$ (similar to how it is done in the category searcher).

**Unit Conversion Searcher:** This searcher finds matches such as weight = 2.2 * net-weight, a conversion between two different types of unit (pounds and kilogram in this case). It first determines physical-quantity attributes, by looking for the presence of certain tokens (e.g., “hours”, “kg”, “$”, etc.) in the name and data of the attributes. The searcher then finds the best conversion from a set of conversion functions between commonly used units.

**Date Searcher:** This searcher finds complex matches for date attributes, using a set of terms in a simple ontology that captures date entities (e.g., day, month, year, week) and relationships among them (e.g., concatenation, generalization, specialization, subset). Suppose the searcher matches target attribute birth-date to ontology concept DATE, and source attributes bday, bmonth, and byear to ontology concepts DAY, MONTH, and YEAR, respectively. Suppose from the ontology we know that DATE is composed of DAY, MONTH, and YEAR, then we can infer that birth-date = concat(bday, bmonth, byear).

## 3.2 The Similarity Estimator

For each target attribute $t$, the searchers suggest a relatively small set of promising match candidates. However, the scores assigned to each of the candidate matches is based only on a single type of information. For example, the text searcher considers only word frequencies via the Naive Bayes learner. Consequently, the accuracy reported by the searchers may not be very accurate.

The task of the similarity estimator is then to further evaluate these candidates, and assign to each of them a final score that measures the similarity between the candidate and $t$. In doing so, the similarity estimator tries to exploit additional types of information to compute a more accurate score for each match. To this end, it employs *multiple evaluator modules*, each of which exploits a specific type of information to suggest a score, and then combines the suggested scores into a final one. It is important to note that such an exhaustive evaluation would be prohibitively expensive to perform during the search phase.

Prior work suggests many evaluator modules, which exploit learning [9, 1, 11, 15], statistical [13], linguistic and heuristic [8, 3, 21] techniques. The modules can be employed at this stage of iMAP. Currently, iMAP uses two modules:

- a name-based evaluator, which computes a score for a match candidate based on the similarity of its name to the name of the target attribute. The name of a match candidate is the concatenation of the names of the attributes appearing in that candidate, together with the names of the tables that contain the attributes.
- a Naive Bayes evaluator, which is the Naive Bayes classifier described earlier in Example 3.2.

These evaluators are similar to corresponding learner modules in [9], and are described in detail in [6].

## 3.3 The Match Selector

Once the similarity estimator has revised the score of the suggested matches of all target attributes, conceivably, the best global match assignment could simply be the one where each target attribute is assigned the match with the highest score. However, this match assignment may not be acceptable in the sense that it may violate certain domain integrity constraints. For example, it may map *two* source attributes to target attribute list-price, thus violating the constraint that a house has only one price.

The task of the match selector is to search for the best global match assignment that satisfies a given set of domain constraints. The match selector is similar in spirit to the constraint handler module described in [9], but extended to handle complex matches in addition to 1-1 matches (see [6]).

A particularly interesting extension that we have developed allows the match selector to "clean up" complex matches using domain constraints. For example, in our experiments the overlap numeric searcher (described in the next section) frequently suggested matches such as lot-area = (lot-sq-feet/43560) + 1.3e-15 $*$ baths. If the selector knows that source attribute baths maps to target attribute num-baths, and that lot area and the number of baths are semantically unrelated and hence typically do not appear in the same formula, then it can drop the terms involving baths (provided that the value of the term is very small), thus transforming the above match into the correct one.

# 4. EXPLOITING DOMAIN KNOWLEDGE

As we experimented with iMAP, we soon realized that exploiting domain knowledge can greatly improve the accuracy of complex matching. Indeed, past work (e.g., [9, 8, 16]) has noted the benefits of exploiting such knowledge in the context of 1-1 matching. There, the knowledge helps in evaluating matches and pruning unlikely matches. In the context of complex matching, however, exploiting domain knowledge brings even greater benefits, because it can also help to direct the search process and prune meaningless candidates early, avoiding costly evaluation. We now describe the use of domain knowledge in iMAP.

iMAP innovates in its use of domain knowledge in two ways. The first is the *types* of knowledge that it uses. Prior work on 1-1 matching has exploited domain constraints and past matches [9, 8, 16]. Here, in addition to these types of knowledge, iMAP also exploits *overlap* data between the databases and *external* data in the domain. Second, iMAP innovates in *how* to use domain knowledge. Specifically, iMAP uses domain knowledge at all levels of the system. In fact, (in the same spirit as pushing selections in query execution plans) we try to push the relevant domain knowledge to as early a point as possible in the match generation.

We now illustrate the above points by discussing how we exploit each particular type of domain knowledge.

**Domain Constraints:** Such constraints are either present in the schemas, or are provided by the domain experts or the user. In Section 6 we show that even with just a few constraints iMAP can greatly improve matching accuracy.

Given a particular domain constraint, iMAP decides which system component should exploit it, trying to use it as early as possible. There are three cases:

- The constraint implies that two attributes of schema $S$ are unrelated, such as "name and beds are unrelated", meaning that they cannot appear in the same match formula. Any searcher can use this constraint to never *generate* any match candidate that combines name and beds.
- The constraint involves a *single* attribute of $T$, such as "the average value of num-rooms does not exceed 10". Any searcher can use this constraint to *evaluate* a match candidate for target attribute num-rooms. However, if the constraint is too expensive to be checked (e.g., when the searcher evaluates a very large number of match candidates), then it may have to be moved to the similarity estimator level, where the number of match candidates that it must be verified on will be far less than that at the searcher level.
- The constraint relates *multiple* attributes of $T$, such as "lot-area and num-baths are unrelated". This constraint can only be exploited at the match selector level, as described earlier in Section 3.3, because the previous levels consider each attribute of schema $T$ in isolation.

**Past Complex Matches:** When mapping tasks are repetitive or done in closely related domains, we may often have examples of past matches. For example, in data integration settings, we map many sources into a single mediated schema [14, 9]. In enterprise data management we often find that we are mapping the same or similar schemas (and different versions thereof) repeatedly. iMAP currently extracts the *expression template* of these matches and uses those templates to guide the search process in the numeric searcher described in Section 3.1.2. For example, given the past match price = pr * (1 + 0.06), it will extract the template VARIABLE * (1 + CONSTANT) and asks the numeric searcher to look for matches of that template.

**Overlap Data:** There are many practical mapping scenarios where the source and target databases share some data (e.g., when two sources describe the same company's data or when two databases are views created from the same underlying database). Clearly, in such "overlap" cases the shared data can provide valuable information for the mapping process (as shown in [23]). Hence, we developed searchers to exploit such data. In what follows, we describe how searchers can incorporate overlap data.

*Overlap Text Searcher:* In the "overlap" case we can use this module instead of the text searcher in Example 3.2, to obtain improved matching accuracy. The module applies the text searcher to obtain an initial set of mappings. It then uses the overlap data to re-evaluate the mappings: the new score of each mapping is the fraction of the overlap data entities for which the mapping is correct. For example, suppose we know that databases $S$ and $T$ share a house listing ("Atlanta, GA,..."). Then, when re-evaluated, mapping agent-address = location receives score 0 because it is not correct for the shared house listing, whereas mapping agent-address = *concat*(city, state) receives score 1.

*Overlap Numeric Searcher:* In the "overlap" cases, this searcher can be used instead of the numeric searcher of Section 3.1.2. For each numeric attribute $t$ of schema $T$, this module finds the best arithmetic expression matches over numeric attributes of schema $S$. Suppose that the overlap data contains ten entities (e.g., house listings) and that the numeric attributes of $S$ are $s_1, s_2, s_3$. Then for each entity the searcher assembles a *numeric tuple* that consists of the values of $t$ and $s_1, s_2, s_3$ for that entity. Next, it applies an *equation discovery system* to the ten assembled numeric tuples in order to find the best arithmetic-expression match for attribute $t$. We use the recently developed LAGRAMGE equation discovery system [27]. This system uses a context-free grammar to define the search space of matches. As a result, this searcher can incorporate domain knowledge on numeric relationships in order to efficiently find the right numeric match. LAGRAMGE conducts a beam search in the space of arithmetic matches. It uses the numeric tuples and the sum-of-squared-errors formula (commonly used in equation discovery) to compute match scores.

*Overlap Category & Schema Mismatch Searchers:* Similar to the overlap text searcher, these searchers use their non-overlap counterparts to find an initial set of matches, then re-evaluate the matches using the overlap data.

**External Data:** Finally, another source of domain data is in sources external to the databases being matched. In principle, we can use external sources to mine properties of attributes (and their data values) that may be useful in schema matching. In fact, the mining can be completely decoupled from the matching system. In iMAP, given a target attribute (e.g., agent-name) and a feature that can be potentially useful in schema matching (e.g., number of distinct agent names), we mine external data (currently supplied by the domain experts) to learn a value distribution of the feature, then apply the learned distribution in evaluating match candidates for that target attribute.

## 5. GENERATING EXPLANATIONS

As described earlier, the goal of a schema mapping system is to provide a design environment where a human user can quickly generate a mapping between a pair of schemas. In doing so, the user will inspect the matches predicted by the system, modify them manually and provide the system feedback. As mapping systems rely on more complex algorithms, there is a need for the system to *explain* to the user the nature of the predictions being made. Explanations can greatly help users gain insights into the matching process and take actions to converge on the correct matches quickly.

In iMAP we offer a novel explanation facility. We begin with an example that illustrates a possible scenario.

Example 5.1. Suppose in matching real-estate schemas, for attribute list-price iMAP produces the ranked matches in decreasing order of confidence score:

```
list-price = price
list-price = price * (1 + monthly-fee-rate)
```

The user is uncertain which of the two is the correct match, and hence asks iMAP to explain the above ranking.

iMAP can explain as follows. Both matches were generated by the overlap numeric searcher, and that searcher ranked the match list-price = price * (1 + monthly-fee-rate) higher than list-price = price. The similarity estimator also agreed on that ranking.

However, the match selector cannot rank list-price = price * (1 + monthly-fee-rate) first because (a) it has accepted the match month-posted = monthly-fee-rate and (b) there is a domain constraint which states that the matches for month-posted and price do not share common attributes. Hence, the match selector must accept the match list-price = price, and in essence flipped the ranking between the two matches.

When asked to explain the match month-posted = monthly-fee-rate, which seems incorrect to the user, iMAP explains that the match is created because the date searcher has examined the data instances of source attribute monthly-fee-rate and concludes that it is a type of date.

At this point, the user examines monthly-fee-rate and tells iMAP that it is definitely not a type of date. iMAP responds by retracting the assumption made by the date searcher, and revising its match candidate ranking, to produce list-price = price * (1 + monthly-fee-rate) as the top match. The user accepts this match with more confidence now that an explanation for the ranking exists. □

We now explain the explanation facility in iMAP. We begin by describing the kinds of questions a user may want to ask of an explanation facility.

### 5.1 Types of User Questions

In principle, there are many question variations that a user may want to ask a matching system. In iMAP, we identified three main questions that are at the core of the rest:

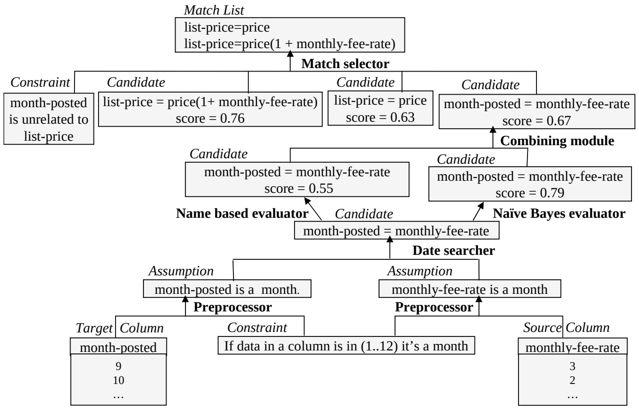


Figure 3: A sample fragment of the dependency graph as generated by iMAP.

**1. Explain existing match:** "why a certain match X is present in the output of iMAP?". An example is asking why the match month-posted = monthly-fee-rate is present in Example 5.1. In essence, the user wants to know how it was created and survived the evaluation and selection process, which components are most instrumental in getting the match to where it is in the output, and what important assumptions were made while generating it.

**2. Explain absent match:** Conversely, "why a certain match Y is *not* present in the iMAP's output".

**3. Explain match ranking:** "why match X is ranked higher than match Y in the output of iMAP".

In fact, it is interesting to note that as we were developing iMAP and experimenting with it, we were asking the same questions.

One of our important design considerations is that we can ask the above questions from *each* component of iMAP (searchers, evaluator modules in the similarity estimator, and match selector). This greatly simplifies the construction of the explanation module, since questions can be reformulated recursively to the underlying components.

## 5.2 The Explanation Module

The key data structure underlying the explanation module of iMAP is the *dependency graph*, which is constructed during the matching process. The dependency graph records the flow of matches, data, and assumptions into and out of system components. The nodes of the graph are: schema attributes, assumptions made by system components, candidate matches, and pieces of domain knowledge such as domain constraints.

Two nodes in the graph are connected by a directed edge if one of them is the successor of the other in the decision process. The label of the edge is the system component that was responsible for the decision.

Figure 3 shows a dependency graph fragment that records the creation and flow of match month-posted = monthly-fee-rate. A preprocessor finds that both month-posted and monthly-fee-rate have values between 1 and 12 and hence makes the assumptions that they represent months. The date searcher consumes these assumptions and generates month-posted = monthly-fee-rate as a match candidate.

This candidate is then scored by the name-based evaluator and the Naive Bayes evaluator. The scores are combined by a combination module to produce a single score. The match selector acts upon the several mapping candidates generated to produce the final list of mappings. Here for the target attribute list-price the selector reduces the rank of the mapping candidate price * ( 1+ monthly-fee-rate) since it discovers that monthly-fee-rate maps to month-posted.

The dependency graph is constructed as the system is run. Each of the components contributes nodes and edges during the execution of the system. At the end of the execution when the system has generated the matches the dependency graph is already in place.

**Generating Explanations:** We now briefly describe how iMAP generates explanations for the three types of predefined queries that have been described in Section 5.1. In each case, the system synthesizes an explanation in English for the user.

To answer the question "why match X is present", iMAP selects the slice of dependency graph that records the creation and processing of match X. For example, the slice for month-posted = monthly-fee-rate is the portion of the graph where the nodes participated in the process of creating that match.

To answer the question "why match X is ranked higher than match Y", the system compares the two slices of the dependency graph corresponding to X and Y. In comparing the slices, it focuses on places where the ranking is flipped and asks the relevant system component to explain why that component flips the ranking.

To answer the question "why match X is not present", iMAP first examines the dependency graph to see if match X has been generated at all. If it has, then iMAP finds out where it has been eliminated, and asks the involved system component to explain why it eliminated X.

If match X has not been generated, then iMAP asks the searchers to see if any of them is capable of generating X. Suppose a searcher $S$ indicates that it can generate X (but did not), then iMAP asks $S$ for an explanation of why it did not generate X. These explanations are processed and then presented to the user. A more elaborate description of how iMAP generates explanations can be found in [6].

**Performance:** Since each searcher produces only $k$ top matches where $k$ is the width of a beam search and hence is small, and since matching in iMAP goes through only three stages (searchers, similarity estimator, selector), it is easy to show that the dependency graph is relatively small. Hence, maintaining a dependency graph and traversing it to generate explanations incur negligible time and storage cost. We must exercise care, however, to make sure that each iMAP component can generate its explanations efficiently, in order to efficiently obtain the global explanations.

# 6. EMPIRICAL EVALUATION

We have evaluated iMAP on four real-world domains. Our goals were to evaluate the matching accuracy of iMAP, to measure the relative contributions of domain knowledge, and to examine the usefulness of match explanations.

**Domains and Data Sources:** Table 2 describes the four real-world domains. Real Estate lists houses for sale. Inventory describes product inventories of a grocery business. Cricket describes cricket players, and Financial Wizard stores financial data of clients.

Obtaining data for schema matching experiments remains a challenge (though several benchmarks are currently being

| Domains | Source schema | | Target schema | # of 1-1 matches | # of complex matches | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | # tables | # attributes | # attributes | | Total | Text | Numeric | Unit | Category | Schema Mismatch | Date |
| Real estate | 2 | 32 | 19 | 7 | 12 | 5 | 3 | 0 | 3 | 1 | 0 |
| Inventory | 2 | 44 | 38 | 27 | 11 | 4 | 4 | 0 | 3 | 0 | 0 |
| Cricket | 3 | 38 | 42 | 22 | 20 | 2 | 6 | 2 | 2 | 3 | 5 |
| Financial wizard | 4 | 41 | 22 | 8 | 14 | 3 | 5 | 0 | 5 | 0 | 1 |

Table 2: Real-world domains for our experiments.

considered or built). In our work, we began by obtaining two independently developed databases for the Cricket domain (from *cricinfo.com* and *cricketbase.com* in December 2002), and used them as the source and target databases.

For the other three domains, we obtained one real-world database for each domain. The databases came from the Internet, the sample databases of Microsoft Access 97, the students in a large undergraduate database class, and from volunteers (who spent time populating the Financial Wizard database). The numbers of tables and attributes in each database are shown under the headline "Source Schema" of Table 2. Next, for each database $S$ we asked volunteers to create a target schema $T$. We asked the volunteers to examine and create complex matches between $T$ and $S$.

The target schemas $T$ are described in Table 2. The table shows the number of attributes of $T$ (under "Target Schema"), the number of 1-1 matches (between $T$ and $S$), and then the number of complex matches, broken down into different types. Below are a few examples of such matches:

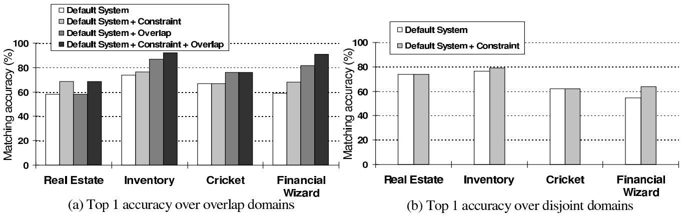
```
name = CONCAT(first_name, last_name)
test_economy_rate = 6 * t_runs_given / t_balls
ODI_overs = 0.1667 * o_balls
Marital_status = f(person_marital_stats)
         Single=f(SIN) Married=f(MAR) Divorced=f(DIV)
fireplace = 1 if house_description contains fireplace
ODI_debut = (o_debut_day-o_debut_month-o_debut_year)
```

In the final step, we populated the schemas with data, using the database obtained for each domain. As discussed in Section 4, both the "overlap" and "disjoint" scenarios where the source and target databases do and do not share data occur frequently in practice. Hence we created both scenarios for experimental purposes. We took care to ensure that the source and target databases share some data in the "overlap" scenarios, but do not share any in the "disjoint" ones.

**Data Processing:** We performed only trivial data cleaning operations such as removing "unknown" and "unk". Next, we specified domain constraints on the schemas. We specified only the most obvious constraints, such as "player-first-name does not appear in the same match as t-highest-score", and "zip-code does not appear in the same match as account-number".

**Experiments:** The above process in effect generated *eight* experimental domains, since for each application (e.g., real estate, inventory, etc.) we have two domains, with *disjoint* and *overlap* data, respectively. We run experiments with several configurations of iMAP on all eight domains, as described in the next subsection.

**Performance Measure:** iMAP outputs for each target attribute a ranked list of best matches. We define the *top-1 matching accuracy* to be the fraction of target attributes whose top-1 match candidates are correct. The *top-3 matching accuracy* is then the fraction of target attributes whose top three candidates include the correct match. The top-3 accuracy is interesting because an interactive matching system typically proposes a *ranked list* of matches to a designer, and therefore the correct match needs not be the top 1.



(a) Top 1 accuracy over overlap domains

(b) Top 1 accuracy over disjoint domains

Figure 4: Top-1 overall matching accuracy.

Several prior works [8, 7] employ the notion of precision and recall to evaluate matching algorithms. Since iMAP finds matches for *all* target attributes, its precision and recall can be shown to be the same, and to be equivalent to the notion of matching accuracy used above.

## 6.1 Overall and 1-1 Matching Accuracy

Figure 4 shows the overall top-1 matching accuracy (that is, the fraction of all target attributes whose best match candidate is correct). Part (a) of the figure shows the accuracy for the four *overlap* domains. For each domain, the four bars from left to right represent respectively the accuracy of the iMAP configuration which exploits (a) no domain knowledge (i.e., the default system), (b) domain constraints, (c) overlap data, and (d) both domain constraints and overlap data.

The results show that iMAP achieves high matching accuracy 68-92% over all four overlap domains. The default iMAP achieves accuracy 58-74%. Exploiting domain constraints or overlap data further improves accuracy by 12-23%, and exploiting both domain constraints and overlap data further improves accuracy by as much as 11%.

Part (b) of Figure 4 shows the accuracy for the four *disjoint* domains. For each domain, the two bars from left to right represent respectively the accuracy of the default iMAP configuration and the one which exploits domain constraints. (Note that there is no overlap data and hence no bars representing the accuracy over exploiting overlap data.)

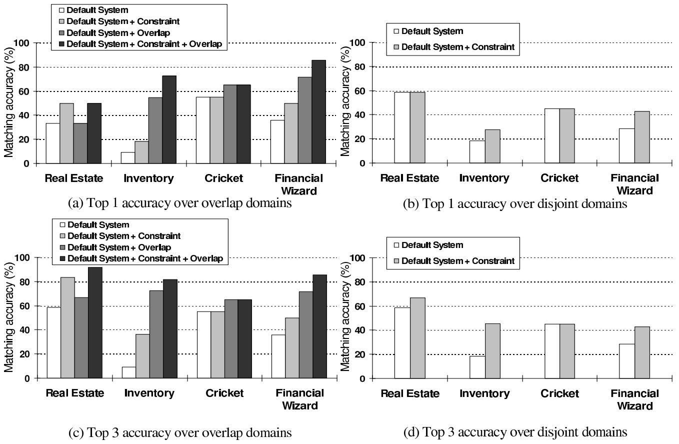
Here iMAP achieves accuracy rates 62-79%, slightly lower than those in the overlap domains. The default iMAP achieves accuracy 55-76%, and exploiting domain constraints improves accuracy by 9%.

In summary, Figure 4 shows that iMAP obtained high overall top-1 accuracy of 62-92% across domains. Its top-3 accuracy (not shown in the figure) is even higher, ranging from 64-95%. Finally, iMAP also achieves top-1 and top-3 accuracy of 77-100% over 1-1 matches (not shown in the figure). These results are competitive with those reported by existing 1-1 matching systems (e.g., [9, 8, 19, 17, 1]).

## 6.2 Complex Matching Accuracy

We now examine how well iMAP does in finding complex matches. Figure 5 shows matching accuracy in a format similar to that of Figure 4, but only for complex matches.

Part (a) of Figure 5 shows the top-1 accuracy 50-86% for the four overlap domains. The default iMAP achieves accuracy 33-55% on all domains, except for 9% on Inventory. At the end of this subsection we analyze the reasons that pre-



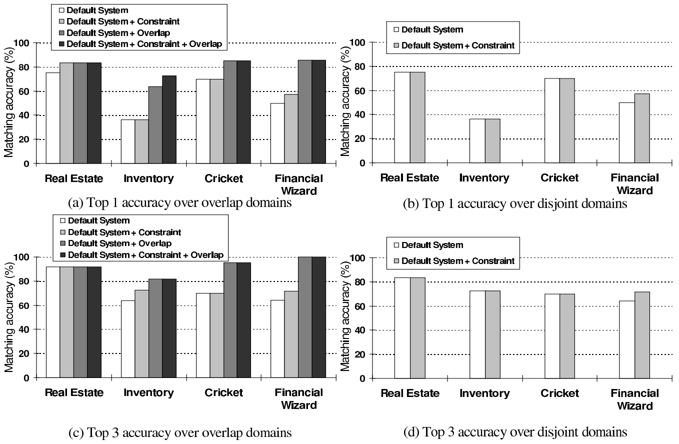
(a) Top 1 accuracy over overlap domains

(b) Top 1 accuracy over disjoint domains

(c) Top 3 accuracy over overlap domains

(d) Top 3 accuracy over disjoint domains

Figure 5: Top-1 (top row) and top-3 (bottom row) matching accuracy for complex matches.



(a) Top 1 accuracy over overlap domains

(b) Top 1 accuracy over disjoint domains

(c) Top 3 accuracy over overlap domains

(d) Top 3 accuracy over disjoint domains

Figure 6: Top-1 (top row) and top-3 matching (bottom row) accuracy for partial complex matches.

vent iMAP from correctly identifying the remaining complex matches, in general and also in the Inventory domain.

As expected, exploiting domain constraints further improves accuracy up to 17%. Exploiting overlap data improve accuracy over the default iMAP by up to 46%, and exploiting both domain constraints and overlap data improves accuracy by 10-64%.

In the four "disjoint" domains (Figure 5.b), the top-1 accuracy is lower, ranging 27-58%. The main reason for lower accuracy is that there is no overlap data to rely on. Hence accuracy for text matches slightly decreases and most numeric matches cannot be predicted. However, accuracy for categorical and schema mismatch matches remains high.

Figure 5.c-d shows that the top-3 accuracy over both overlap and disjoint domains are 43-92%, improving up to 42% over the top-1 accuracy of Figure 5.a-b. Thus, a significant number of correct complex matches are in the top three matches (per target attribute) produced by iMAP.

To examine exploiting past matches, we asked students in a database class to create database schemas in the same domain as Financial Wizard databases, then asked them to create 15 complex matches between the schemas. Next, we applied iMAP to exploit these 15 matches, as explained in Section 4. iMAP was able to find complex numeric matches and improve the top-1 matching accuracy on the disjoint Financial Wizard domain by 28%.

**Discussion:** There are several reasons that prevent the current iMAP system from identifying all complex matches. First, in many cases iMAP could not find "smaller components" of a complex match. In our example, when the correct match is agent-address = concat(apt-number, street-name,city,state), iMAP may return only the complex mapping concat(street-name,city,state). This is because the current learning and statistical techniques employed by iMAP are not sophisticated enough to detect such subtle difference of just a single number (apt-number). We believe adding format learning techniques may help in many such cases.

Second, the reverse problem also holds: in many cases iMAP added "small noise components" to a complex match. For example, in the Inventory domain, iMAP added agent-id (a single digit number) to many complex matches related to agents, thus reducing accuracy significantly. As we have shown earlier, this problem can be addressed by more aggressive match cleaning and enforcing of domain constraints. This underscores the importance of automatically learning domain constraints for complex matching.

Third, if the databases are disjoint, it was very difficult to discover meaningful numeric relationships. This is a fundamental problem that underlies *any* system that finds complex matches. Here, we have proposed a preliminary solution of exploiting past numeric matches and showed its promise. We believe more work is needed on this topic in particular, and on the issue of constructing and re-using domain knowledge in general.

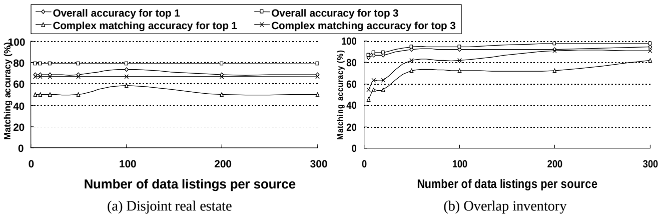
Finally, many complex matches are not in the top one, but somewhere in the top three (and more general, in the top ten) of the matches predicted by iMAP. Given the fact that finding a complex match requires gluing together so many different components, perhaps this is inevitable and inherent to any complex matching solution. This underscores the importance of generating explanations and building effective mapping design environment, so that humans can examine the top ranked matches to create mappings.

**Finding Partial Complex Matches:** So far we have considered only cases where iMAP produces the *exact* complex matches, that is, finding the *exact* attributes, expression, and relationship. We note, however, that even when iMAP finds only *partial* complex matches, these matches would still be useful, because the user can elaborate on them to find the exact matches.

Hence, we examine how well iMAP finds these partial complex matches. The first type of such partial matches finds only the right attributes. Figure 6 shows the accuracy for this type. The top-1 accuracy is 73-86% over overlap domains and 36-75% over disjoint domains. The top-3 accuracy is high, ranging from 82 to 100% over overlap domains and 70-83% over disjoint domains. These results suggest that in a significant number of cases iMAP finds the correct set of attributes in a complex match.

**Finding Value Correspondences:** Even when iMAP does not find the correct match, in many cases it would still be easy for user to examine the ranked list of candidate matches and find the correct expression. For example, in the Real Estate domain, iMAP generated the following top three matches for attribute num-rooms:

```
2 * dining-rooms + bed-rooms + bath-rooms
bath-rooms + bed-rooms + 2 * dining-rooms
bath-rooms + 2 * living-rooms + bed-rooms
```



(a) Disjoint real estate

(b) Overlap inventory

Figure 7: Performance sensitivity

For this attribute, the overlap numeric matcher converged before generating the correct complex match having all four terms. Nevertheless, a user can easily elaborate on the above top three incorrect matches to arrive at the correct expression.

Finding the correct expression is useful because such expressions, known as value correspondences, can be fed directly into a schema refinement tool such as Clio [29], to produce the final correct mapping. Hence, we are interested in knowing how well iMAP does in finding value correspondences. We asked several volunteers to examine the top three of iMAP's results, on all eight domains, and count the cases where it would be fairly obvious from the top three matches that what the correct value correspondences should be. iMAP found 75-93% of correct value correspondences over the overlap domains and 57-75% over the disjoint domains. The results, while somewhat subjective, due to the judgment of the volunteers, do suggest that iMAP can find value correspondences in a large number of cases.

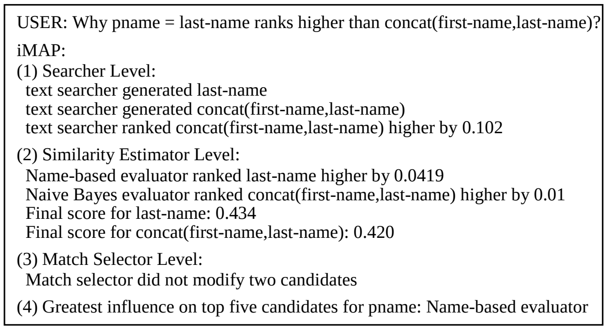
**Performance Sensitivity:** Figure 7.a-b shows the variation of the top-1 and top-3 matching accuracy as a function of the number of data tuples available from each source, for the "disjoint" Real Estate and "overlap" Inventory domains, respectively. The results show that on these domains the performance of iMAP stabilizes fairly quickly and almost levels off after 100 data tuples. Experiments with other domains show the same phenomenon. iMAP thus appears to be robust, and can work well with relatively little data. One of the reasons this observation is important is that we can reduce the running time of iMAP if we run it on fewer examples.

**Efficiency:** Our unoptimized iMAP version took 5-20 minutes on the experimental domains. (iMAP spent most of the time searching for promising match candidates.) We found that by applying a variety of techniques, including preliminary preprocessing to break the schemas into independent chunks on each of which iMAP can work in isolation, we can significantly reduce iMAP's runtime. In the long term, scaling up matching systems to very large schemas is an important and interesting topic that we are planning to pursue.

## 6.3 Explaining Match Predictions

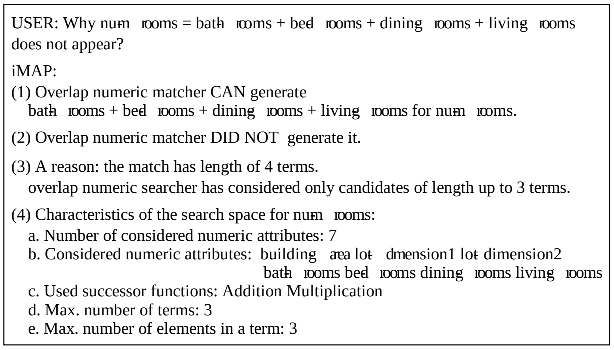
We now provide anecdotal evidence that iMAP can generate meaningful explanations that provide insight into the working of the system and that are actionable to the user.

We examined the matches produced by iMAP in various experimental domains, and asked for explanations in many cases of incorrect or suspicious matches. Figures 8 and 9 show the explanations in two such cases, slightly edited for space reasons.



```
USER: Why pname = last-name ranks higher than concat(first-name,last-name)?
iMAP:
(1) Searcher Level:
  text searcher generated last-name
  text searcher generated concat(first-name,last-name)
  text searcher ranked concat(first-name,last-name) higher by 0.102
(2) Similarity Estimator Level:
  Name-based evaluator ranked last-name higher by 0.0419
  Naive Bayes evaluator ranked concat(first-name,last-name) higher by 0.01
  Final score for last-name: 0.434
  Final score for concat(first-name,last-name): 0.420
(3) Match Selector Level:
  Match selector did not modify two candidates
(4) Greatest influence on top five candidates for pname: Name-based evaluator
```

Figure 8: Generated explanation for pname vs. concat(first-name,last-name).



```
USER: Why num_rooms = bath_rooms + bed_rooms + dining_rooms + living_rooms
does not appear?
iMAP:
(1) Overlap numeric matcher CAN generate
  bath_rooms + bed_rooms + dining_rooms + living_rooms for num_rooms.
(2) Overlap numeric matcher DID NOT generate it.
(3) A reason: the match has length of 4 terms.
  overlap numeric searcher has considered only candidates of length up to 3 terms.
(4) Characteristics of the search space for num_rooms:
  a. Number of considered numeric attributes: 7
  b. Considered numeric attributes: building_area lot_dimension1 lot_dimension2
                                    bath_rooms bed_rooms dining_rooms living_rooms
  c. Used successor functions: Addition Multiplication
  d. Max. number of terms: 3
  e. Max. number of elements in a term: 3
```

Figure 9: Generated explanation for numrooms.

Figure 8 shows the explanation for question "why pname = last-name ranks higher than concat(first-name,last-name)" in the Cricket domain. The figure shows that at the searcher level concat(first-name,last-name) was ranked higher than last-name. It also clearly shows that things went wrong at the similarity estimator level. Here, the Naive Bayes evaluator still ranked matches correctly, but the name-based evaluator flipped the ranking. This flipping was clearly responsible for the final ranking, since the explanation shows that the match selector did not modify the ranking that came from the similarity estimator.

The last line of the explanation also confirmed the above conclusion, since it states that the name-based evaluator has the greatest influence on the top five match candidates for pname. The influence of an evaluator is computed by (a) simulating the matching process on the dependency graph without that evaluator, and (b) compute the difference between the new ranking of candidate matches and the original ranking. The difference in ranking is currently computed as $\sum_i |n_i - o_i|$, where $n_i$ and $o_i$ are the positions of match candidate $i$ in the new ranking and old ranking, respectively.

Thus, the main reason for the incorrect ranking for pname appears to be that the name-based evaluator has too much influence. This explanation would allow the user to fine tune the system, possibly by reducing the weight of the name-based evaluator (in the score combination step)

In the second example in Figure 9, when asked why a particular match for num-rooms (in the Real Estate domain) does not appear in the output, iMAP did not find that match on the dependency graph, so it asked the searchers. The overlap numeric searcher explained that it could, but did not generate the match because the match has length four and the searcher converged before generating candidates of that length.

The above explanation suggests that the convergence criterion of the overlap numeric searcher was set too loosely. This provides grounds for the future actions of the user.

## 7. RELATED WORK

To the best of our knowledge, the only other work on complex matching is [28]. This work considers finding complex matches between two schemas by first mapping them into a domain ontology, then constructing the matches based on the relationships inherent in that ontology. Such ontology-based matching would work very well in certain contexts (e.g., see the date searcher in Section 3.1.2) and can be added to iMAP as additional searchers.

Many works have addressed the schema matching problem (e.g., [21, 2, 22, 9, 17, 19, 15, 3, 1, 16, 13], see also [24] for a survey of other works in databases and AI). Among these, iMAP is most related to the current body of work on building multi-matcher systems [9, 8, 11], but builds upon these works to find *both* 1-1 and complex matches. In particular, iMAP innovates significantly by adding the searchers to find complex matches, and showing that the multi-matcher architecture can indeed be extended to handle complex matching. Further, iMAP considers the issue of exploiting domain knowledge in more depth, and offers a novel explanation capability.

As discussed earlier, the Clio system [29] has developed a sophisticated set of user-interaction techniques to elaborate on the matches to create SQL-style mappings. Our work here is therefore complementary to Clio. Indeed, our experience with iMAP suggests that semantic mappings can best be created using a combination of iMAP-style automatic techniques and Clio-style user interaction. We are currently studying such combinations.

## 8. CONCLUSION

Semantic matches are key for enabling a wide variety of data sharing and exchange scenarios. The vast majority of the research on schema matching has focused on 1-1 matches. This paper described a solution to the problem of finding complex matches, which are prevalent in practice. The key challenge with complex matches is that the space of possible matching candidates is possibly unbounded, and evaluating each candidate is harder. iMAP uses two main techniques to search the space effectively. First, it employs a set of specialized searchers that explore meaningful parts of the space. Second, it makes aggressive use of various types of domain knowledge to guide the search and the evaluation where possible. Keeping in spirit with recent works, the architecture of iMAP is modular and extensible. New searchers and new evaluation modules can be added easily. Finally, iMAP offers a novel explanation facility that helps human users interact with the system to generate mappings quickly. Our experimental results show that iMAP achieves 43-92% accuracy on several real-world domains, thus demonstrating the promise of the approach.

## 9. REFERENCES

[1] J. Berlin and A. Motro. Database schema matching using machine learning with feature selection. In *Proc. of CAiSE-2002*.

[2] S. Castano and V. D. Antonellis. A schema analysis and reconciliation tool environment. In *Proc. of IDEAS-1999*.

[3] C. Clifton, E. Housman, and A. Rosenthal. Experience with a combined approach to attribute-matching across heterogeneous databases. In *Proc. of the IFIP Working Conference on Data Semantics (DS-7)*, 1997.

[4] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, New York, NY, 1991.

[5] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proc. of SIGMOD-2002*.

[6] R. Dhamankar. Semi-automated discovery of matches between schemas, ontologies, and data fragments of disparate data sources. *M.S. Thesis, Dept. of CS, Univ. of Illinois*. To appear.

[7] H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proceedings of the 2nd Int. Workshop on Web Databases 2002*.

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

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

[10] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, NY, 1973.

[11] D. Embley, D. Jackman, and L. Xu. Multifaceted exploitation of metadata for attribute match discovery in information integration. In *Proc. of the WIIW-01*, 2001.

[12] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *Proc. of SIGMOD-2003*.

[13] J. Kang and J. Naughton. On schema matching with opaque column names and data values. In *Proc. of SIGMOD-2003*.

[14] M. Lenzerini. Data integration; a theoretical perspective. In *Proc. of PODS-2002*.

[15] W. Li and C. Clifton. SEMINT: A tool for identifying attribute correspondence in heterogeneous databases using neural networks. *Data and Knowledge Engineering*, 33:49–84, 2000.

[16] J. Madhavan, P. Bernstein, K. Chen, A. Halevy, and P. Shenoy. Matching schemas by learning from a schema corpus. In *Proc. of the IJCAI-03 Workshop on Info. Integration*, 2003.

[17] J. Madhavan, P. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proc. of VLDB-2001*.

[18] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, US, 1999.

[19] S. Melnik, H. Molina-Garcia, and E. Rahm. Similarity flooding: a versatile graph matching algorithm. In *Proc. of ICDE-2002*.

[20] R. Miller. Using schematically heterogeneous structures. In *Proc. of SIGMOD-1998*.

[21] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proc. of VLDB-1998*.

[22] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources. In *Proc. of Fusion-1999*.

[23] M. Perkowitz and O. Etzioni. Category translation: Learning to understand information on the internet. In *Proc. of Int. Conf. on AI (IJCAI)*, 1995.

[24] E. Rahm and P. Bernstein. On matching schemas automatically. *VLDB Journal*, 10(4), 2001.

[25] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[26] L. Seligman, A. Rosenthal, P. Lehner, and A. Smith. Data integration: Where does the time go? *IEEE Data Engineering Bulletin*, 2002.

[27] L. Todorovski and S. Dzeroski. Declarative bias in equation discovery. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 1997.

[28] L. Xu and D. Embley. Using domain ontologies to discover direct and indirect matches for schema elements. In *Proc. of the Semantic Integration Workshop at ISWC-2003*.

[29] L. Yan, R. Miller, L. Haas, and R. Fagin. Data driven understanding and refinement of schema mappings. In *Proc. of SIGMOD-2001*.