

Source-aware Entity Matching: A Compositional Approach

Warren Shen¹, Pedro DeRose¹, Long Vu², AnHai Doan¹, Raghu Ramakrishnan³

¹University of Wisconsin, Madison ²University of Illinois, Urbana ³Yahoo! Research

Abstract

Entity matching (a.k.a. record linkage) plays a crucial role in integrating multiple data sources, and numerous matching solutions have been developed. However, the solutions have largely exploited only information available in the mentions and employed a single matching technique. We show how to exploit information about data sources to significantly improve matching accuracy. In particular, we observe that different sources often vary substantially in their level of semantic ambiguity, thus requiring different matching techniques. In addition, it is often beneficial to group and match mentions in related sources first, before considering other sources. These observations lead to a large space of matching strategies, analogous to the space of query evaluation plans considered by a relational optimizer. We propose viewing entity matching as a composition of basic steps into a “match execution plan”. We analyze formal properties of the plan space, and show how to find a good match plan. To do so, we employ ideas from social network analysis to infer the ambiguity and relatedness of data sources. We conducted extensive experiments on several real-world data sets on the Web and in the domain of personal information management (PIM). The results show that our solution significantly outperforms current best matching methods.

1. Introduction

Entity matching decides if two given mentions in the data, such as “David Smith” and “D. Smith”, refer to the same real-world entity. This problem arises in many applications that integrate data from multiple sources. Consequently, numerous solutions have been developed (e.g. [19, 13, 24, 10, 5], see [23] for a recent tutorial).

While much progress has been made, the current solutions have largely exploited only information *within* the mentions, and employed a *single matching solution*, henceforth called a *matcher*. These restrictions often lead to the dilemma that no matter which matcher we select, we fail to match correctly a significant number of mentions, as illustrated below.

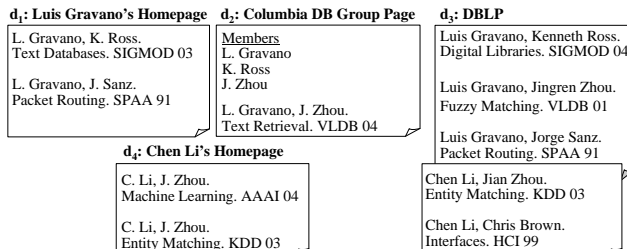


Figure 1. Matching mentions in four Web data sources.

Example 1.1 Figure 1 shows five simplified Web pages that belong to four data sources: Luis Gravano’s homepage (source d_1), the Columbia database group website (d_2), DBLP (d_3), and Chen Li’s homepage (d_4). Suppose that we have extracted various mentions from these pages (e.g., “L. Gravano”, “K. Ross”, “Text Databases”, etc.), and have inferred co-author relationships such as “K. Ross is a co-author of L. Gravano”. Now consider matching the two mentions “Chen Li” in Chen Li’s DBLP homepage (on the right of the figure):

Chen Li, Jian Zhou. Entity Matching. KDD 03
Chen Li, Chris Brown. Interfaces. HCI 99

Suppose these refer to two different researchers called “Chen Li” (in database and HCI, respectively). To distinguish them, we may want to employ a “conservative” matcher s_1 that declares two mentions matched only if they share the same name (based on some string similarity measure) and at least one co-author. Then matcher s_1 would make the correct decision of not matching the above two “Chen Li” mentions. However, it would fail to match the two “Luis Gravano” mentions in Luis Gravano’s homepage:

L. Gravano, K. Ross. Text Databases. SIGMOD 03
L. Gravano, J. Sanz. Packet Routing. SPAA 91

Though these mentions share no co-authors, they clearly refer to the same person since they occur on Luis Gravano’s homepage.

If we employ a more “relaxed” matcher, such as a matcher s_0 that declares two mentions matched if they share the same name, we would have the reverse situation. We would correctly match the above two “L. Gravano” mentions, but would incorrectly match the two “Chen Li” mentions in Chen Li’s DBLP homepage. □

Thus, in the above example it appears that no matter how we select a matcher (s_1 or s_0), we make incorrect matching decisions for certain subsets of mentions. This dilemma arises in many matching scenarios. The fundamental reason is the varying degree of “semantic ambiguity”. Frequently, some subsets of data (e.g., homepages) are less ambiguous

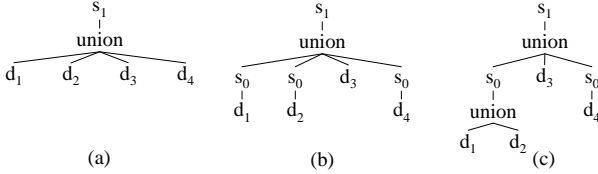


Figure 2. (a) The matching strategy used by current solutions, and (b)-(c) strategies that employ multiple matchers.

with respect to matching, while some other subsets (e.g., DBLP) are significantly more ambiguous. In such scenarios, a “conservative” matcher such as s_1 will avoid many false matches for ambiguous cases, but prevent us from matching less ambiguous ones, whereas a more “relaxed” matcher such as s_0 will reverse the situation.

The Soccer Solution: In this paper we describe Soccer (Source Conscious Compiler for Entity Resolution), a solution to the above problem. Our first idea is as follows. In many practical settings we know the underlying data sources where the mentions come from. As discussed above, such sources often vary significantly in their level of “semantic ambiguity”. Thus, we estimate the ambiguity of the data sources, then apply *multiple matchers*: a more conservative one to more ambiguous sources, and a more relaxed one to others, as illustrated below.

Example 1.2 Continuing with Example 1.1, Figure 2.a shows the matching strategy of current solutions [23]: take the union of all mentions in sources $d_1 - d_4$, then apply a single matcher, e.g., s_1 . Now suppose we know that s_1 (matches mentions if they share similar names and at least one co-author) is too “conservative” for sources d_1, d_2 , and d_4 (the homepages and group page), because they are not highly ambiguous. Then we can apply the more relaxed matcher s_0 (e.g., matches mentions if they share similar names) to these sources, take the union of all mentions in $d_1 - d_4$ (but keeping the predicted matches), then finally apply matcher s_1 to the union. The resulting “match plan” is shown in Figure 2.b. □

By replacing matcher s_1 with a more “relaxed” matcher s_0 on less ambiguous data sources, we benefit in two important ways. First, mentions in those sources can now be immediately matched. Second, these matched mentions provide extra information for subsequent matchers. To see this, consider the two mentions “L. Gravano” in d_1 :

L. Gravano, K. Ross. Text Databases. SIGMOD 03
 L. Gravano, J. Sanz. Packet Routing. SPAA 91

Once we have matched them using s_0 , we can “enrich” each mention by adding to its set of co-authors all co-authors of the other mention. Thus, the co-author sets of both “L. Gravano” mentions will be {“K. Ross”, “J. Sanz”}. These mentions will now share co-authors with the first and third mentions of “Luis Gravano” in DBLP, and hence will match those.

Thus, the more we can “enrich” mentions (in a way that minimizes possible mistakes, such as adding incorrect co-authors), the more mention pairs we can potentially match.

This leads to the second idea underlying Soccer – it is often beneficial to group and match mentions in *related* sources first, before considering other sources:

Example 1.3 Suppose we know that sources d_1 and d_2 are related in that they refer to a set of related people. Then we can apply the match plan in Figure 2.c. This plan is similar to the one in Figure 2.b, except here we apply s_0 to the union of d_1 and d_2 , rather than to each of them individually. With this plan, all four mentions “L. Gravano” in d_1 and d_2 will match (because s_0 is applied). Consequently, the co-author set of “L. Gravano” is now {“K. Ross”, “J. Sanz”, “J. Zhou”}. This enables the matching of the mentions “L. Gravano” with the second mention “Luis Gravano” in DBLP, a matching that the previous plan cannot do. □

The plan in Figure 2.c is what Soccer would return in this matching scenario.

Motivations: The original motivation for Soccer came from our current work on building DBLife, a vertical portal for the database research community [12]. A key challenge in DBLife is to match mentions of various entity types (e.g., researchers, publications, conference) across a broad range of database-related data sources (researcher homepages, group pages, DBworld, conference homepages, etc.). We observed that these sources vary significantly in their semantic ambiguity. Another motivation for our work came from the personal information management (PIM) area (e.g., [13]), where a key problem is to match mentions from disparate sources, such as email folders, files, directories, etc. PIM data sources also often exhibit significantly varying degrees of semantic ambiguity. For example, an email folder that stores messages of several mailing lists can be highly ambiguous, but far less so is a folder that stores messages from co-authors of a specific ICDE paper. Section 5 shows that Soccer outperforms current matching methods on a data set sampled from DBLife and a PIM data set.

Contributions: Our first contribution in developing Soccer is to define the match problem. Specifically, we treat each matching algorithm as a “blackbox” operator called *matcher*, then cast mention matching as the problem of finding an optimal match plan in a large plan space, where each plan specifies which matcher to apply to each data source, and which sources should be grouped. Figures 2.a-c show examples of such plans.

We then consider a restricted version of the above general problem. Specifically, current matching solutions can be viewed as the domain expert executing a *default plan* q_{def} that employs a single matcher, denoted as s_1 , that he or she believes will provide the highest accuracy [23]. We assume the expert can also provide another matcher, denoted as s_0 , that is more “relaxed” than s_1 . Matcher s_0 can often be quickly constructed from s_1 , by “relaxing” the similarity measures or thresholds employed by s_1 (see Section 2).

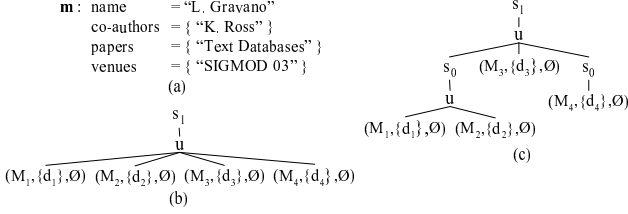


Figure 3. Examples of (a) mention and attributes, and (b)-(c) match plans.

Our goal then is to find a plan q_+ that employs both s_1 and s_0 , and significantly outperforms the default plan q_{def} . This problem setting is practical, and yet simple enough to provide a tractable first study of compositional approaches to mention matching. Considering a more general setting is a subject of future work.

We turn to the difficult challenge of estimating the semantic ambiguity and relatedness of the data sources. We observe that the entities mentioned in the data sources (e.g., people, publications, etc.) and their relationships (e.g., co-authors, advising, etc.) usually form a “social network” graph [31]. If a part of this graph is dense, then it is likely to refer to a strongly *cohesive* “social group”, where entities are highly related to one another. In a sense, the more cohesive a group is, the less ambiguity exists, suggesting that a more relaxed matcher is appropriate. Drawing from this observation, we build a “social network” graph using (among others) the matches predicted by the more conservative matcher s_1 , then exploit the graph to estimate source ambiguity and relatedness.

Finally, we show that, while it may be intractable to find the best plan q_* , we can efficiently find a good plan q_+ using the above source ambiguity and relatedness estimates. We formally analyze the properties of the matching problem, and provide general conditions under which plan q_+ (which employs both matchers s_1 and s_0) provably achieves equivalent or higher accuracy than the default plan q_{def} of employing just matcher s_1 .

2. Problem Definition

In this section we first define a general match problem, which casts mention matching as finding an optimal match plan in a large plan space. We then describe a restriction of this problem considered further in this paper.

We assume that the set of *mentions* M to be matched comes from a set of data sources D . Example sources include homepages, DBLP, DBworld, email folders, and Bibtext files. Mentions fall into multiple types: people, papers, conferences, etc. We will describe how Soccer matches mentions of a single type, then discuss how to extend Soccer to match mentions of multiple types in Section 4.

We represent each mention m with a set of attributes (e.g., name, co-authors, and pub-venues). Attributes can be atomic (e.g., name), or set-valued (e.g., co-authors).

Many algorithms have been proposed to extract mention attributes from the raw data (e.g. [1, 7, 33]). Hence, similar to recent work on mention matching [5, 13, 24, 10], we assume that the attributes have been extracted, and focus on the problem of matching the mentions. Figure 3.a shows an example mention m extracted from Luis Gravano’s homepage in Figure 1.

We denote a *match prediction* as $m_i = m_j$, stating that mentions m_i and m_j refer to the same real-world entity. When employed as an operator within a match plan, a *matcher* often takes as input not just mentions (and associated data sources), but also predictions from other matchers, and leverages the input to make more predictions. Hence, we formally define matchers as follows:

Definition 1 (Matcher) A *matcher* s takes as input a triple (N, A, P) and produces as output a triple (N, A, P') , where N is the set of mentions from a set of data sources A , P and P' are sets of match predictions, and $P \subseteq P'$.

Virtually any current matching algorithm can be viewed as a matcher in our problem context. To illustrate matchers, we briefly describe one way that matcher s_1 (mentioned in the introduction) may work:

Example 2.1 (Matcher s_1) First, matcher s_1 defines a match function f that, when taken as input two people mentions m_i and m_j , predicts “matched” only if they share similar names and at least one co-author; i.e., only if

$$[\text{sim}(m_i.\text{name}, m_j.\text{name}) \geq t] \wedge [m_i.\text{co-authors} \cap m_j.\text{co-authors} \geq 1]$$

is true, where $\text{sim}(m_i.\text{name}, m_j.\text{name})$ is a function that measures the similarity between two input strings [6], and t is a threshold set by the domain expert.

Matcher s_1 then proceeds in iterations. In each iteration it starts by using the match predictions it already has (if any) to “enrich” the mentions. For instance, given the prediction “L. Gravano” = “Luis Gravano”, s_1 enriches “L. Gravano” by adding to its set of co-authors all co-authors of “Luis Gravano”, and vice versa. Next, s_1 applies the match function f to all mention pairs (after enrichment) to predict matches (in practice various assumptions are often made so that s_1 does not have to consider all mention pairs, only a promising subset, see [26]). Matcher s_1 repeats the above two steps until a convergence criterion is reached (e.g., when the set of predicted matches has stabilized). □

Examples of matching solutions that follow this “iterative enrich and match” approach include [4, 13, 24, 29].

Soccer aims to employ multiple matchers, some being more “relaxed” than others. We say matcher s_0 is more “relaxed” than matcher s_1 if on any input the predictions made by s_0 contain those made by s_1 . Formally,

Definition 2 (Relaxed Matcher) Let the set of predictions made by a matcher s on input (N, A, P) be $\text{Pred}(s, (N, A, P))$. Then we say matcher s_0 is more relaxed than matcher s_1 if and only if $\forall (N, A, P) \text{Pred}(s_0, (N, A, P)) \supseteq \text{Pred}(s_1, (N, A, P))$.

We can create relaxed matchers in many ways. One way to do so is to start with a matcher s_1 (e.g., supplied by a

domain expert), then “relax” the match function employed by s_1 . For example, suppose s_1 states that “two mentions match only if they share similar names and at least one co-author.” Then we can “relax” this condition to “two mentions match only if they share similar names,” and obtain a new matcher s_0 . As another example, if the matcher employs a threshold for predicting matches, then lowering this threshold often creates a relaxed matcher.

Given a set of data sources D and a set of matchers S , a match plan specifies a strategy to match mentions in D , using matchers in S . Formally,

Definition 3 (Match Plan & Plan Space) Let D and S be defined as above, then a match plan q is a tree where (a) each leaf is a triple (N, A, \emptyset) where $A \subseteq D$ and N are the mentions of A , and (b) each internal node is either a matcher $s \in S$ or the union operator, and (c) the root is a matcher $s \in S$. The set of all such plans form a plan space for D and S .

In a sense, a match plan is similar to an execution tree in relational contexts. The plan specifies which matcher is applied to which subset of data sources, and how sources are grouped. Figures 3.b-c show two example plans. The first plan (Figure 3.b) groups all four data sources by taking the union of all mentions, then applies a matcher s_1 to this union. The second plan (Figure 3.c) first groups sources d_1 and d_2 , then applies matcher s_0 to their mentions. Next, matcher s_0 is applied to the mentions in d_4 . Finally, the plan takes the union of all the mentions (from $d_1 \cup d_2$, d_3 , and d_4), and applies s_1 to the union. We can now define the general match problem as follows:

Definition 4 (General Match Problem) Given (a) a set of data sources D with a set of mentions M , (b) a set of matchers S and (c) a utility function \mathcal{U} defined over the matching process (\mathcal{U} can take into account performance factors such as matching accuracy, execution time, etc.), the general match problem is to find the optimal match plan q_* . Formally, let \mathcal{Q} be the space of all possible match plans over D and S . Then, $q_* = \operatorname{argmax}_{q \in \mathcal{Q}} \mathcal{U}(q)$.

Match Problem Considered by Soccer: In this paper we consider a restricted version of the above general problem. First, like most recent work on mention matching (e.g. [13, 24, 9]), we consider maximizing only accuracy:

Definition 5 (Matching Accuracy) Given a set of data sources D with mentions M and a plan q over D , let $P(q)$ be the match predictions made after executing q , and let P_* be the correct set of predictions over M . Then, the accuracy of q is $F_1(q) = 2 \cdot \operatorname{Pr}(q) \cdot \operatorname{Re}(q) / [\operatorname{Pr}(q) + \operatorname{Re}(q)]$, where precision $\operatorname{Pr}(q) = (|P(q) \cap P_*|) / |P(q)|$, and recall $\operatorname{Re}(q) = (|P(q) \cap P_*|) / |P_*|$.

Next, we consider only two matchers. This version is still practical, and yet sufficiently simple to provide a tractable first study of our compositional approach. Specifically, for the data set D we assume that the domain expert has developed the best single-matcher solution, according to his or her knowledge, and would use this solution today if Soccer is not available. We view this solution as a default plan q_{def} with a single matcher s_1 .

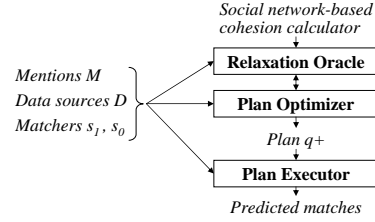


Figure 4. The Soccer architecture.

Next, we ask the expert to create a more relaxed matcher s_0 (often by just relaxing the similarity measure employed by s_1 , as discussed above). With D , the two matchers s_1 and s_0 define a space \mathcal{Q} of match plans, which also includes the default plan q_{def} . It turns out that finding the optimal plan q_* in \mathcal{Q} may be intractable (Section 4). Hence, we settle for the more modest goal of finding a good plan q_+ that significantly improves accuracy over q_{def} . In what follows we describe how Soccer finds plan q_+ , then describes certain general conditions under which q_+ is provably equivalent to or better than q_{def} .

3. The Soccer Approach

The Soccer architecture consists of three main modules (Figure 4). Given a set of mentions M over data sources D , a matcher s_1 , and a more relaxed matcher s_0 , the *relaxation oracle* uses matcher s_1 to analyze the data sources D . In doing so, it requires some minimal feedback from the user to learn a cutoff threshold. Once this is done, given any set of data sources $A \subseteq D$, the oracle can predict whether A is “relaxable”, i.e., whether the more relaxed matcher s_0 can be reliably applied to A .

Next, the *plan optimizer* repeatedly calls the relaxation oracle to evaluate the “relaxability” of various subsets of data sources, and uses this information to find a match plan q_+ that can achieve significantly higher accuracy than the default plan q_{def} . Finally, the *plan executor* executes q_+ , and returns the predicted matches.

The rest of this section describes the relaxation oracle, and Section 4 describes the plan optimizer. The plan executor is relatively straightforward and is not described further.

Relaxable Sets of Data Sources: To describe the relaxation oracle, we start with the notion of “relaxable” sets. Consider a set of data sources $A \subseteq D$ and two matchers s_1, s_0 . When there is no ambiguity, we will write $s_1(A)$ (or $s_0(A)$) to mean applying s_1 (or s_0) to the mentions of A .

Suppose that s_0 is more relaxed than s_1 . Now consider a match plan q (over data sources D) that contains a fragment $s_1(A)$. Let q' be the new plan obtained by replacing $s_1(A)$ in q with $s_0(A)$. We may think that q' is better than q because s_0 is more relaxed than s_1 . However, this is not necessarily so. Matcher s_0 being more relaxed only means that $s_0(A)$ makes *more predictions* than $s_1(A)$ (Definition 2). A number of the extra predictions may still be incorrect, hence overall q' may still achieve lower accuracy than q . If a subset of data sources A is such that given any plan q , s_0

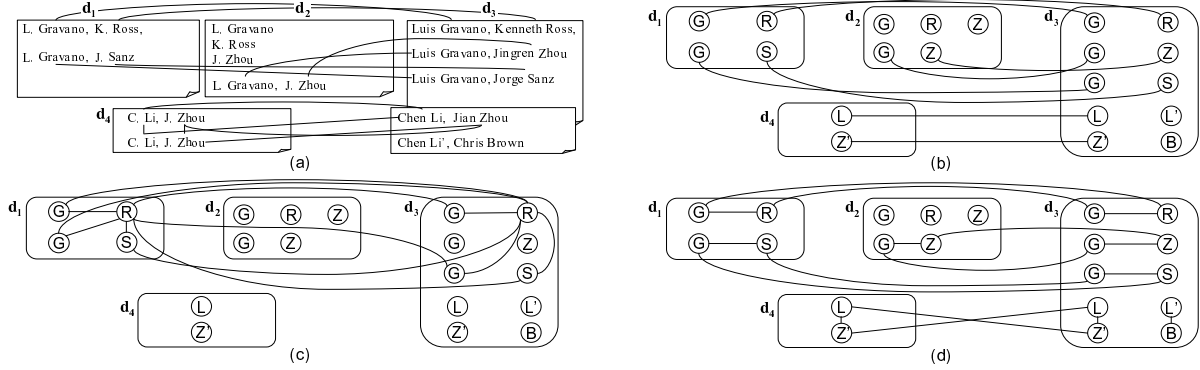


Figure 5. An example to illustrate the process of creating a “social network” graph in the relaxation oracle.

can be applied to A in place of s_1 to obtain an equivalent or better plan q' , then we call A “relaxable.” Formally,

Definition 6 (Relaxable Set) A set of sources A is relaxable with respect to s_1 and s_0 over data sources D if and only if for any match plan over D that contains fragment $s_1(A)$, replacing that fragment with $s_0(A)$ would result in an equivalent or better plan.

Given any set of data sources $A \subseteq D$, the goal of the relaxation oracle is to predict if A is relaxable. The plan optimizer then uses this information to find a good plan q_+ . In what follows we describe how the oracle predicts such relaxabilities.

Leveraging “Social Network” Cohesion: To predict relaxabilities, we leverage ideas from the area of social network analysis [31]. Specifically, we observe that the entities mentioned in the data sources and the relationships between the entities form a “social network” graph G . Now consider a set of data sources A , which corresponds to a subgraph G_A of G . If this subgraph is dense, then it is likely to refer to a strongly *cohesive* social group, where entities are highly related to each other.

In such a cohesive group, there is often less ambiguity, such that entities often need to be mentioned only informally without specifying all applicable attributes. For example, within the pages of a database group, a person can often be mentioned just by name (e.g., “Luis” or “Luis Gravano”), without title, email, and affiliation. Less cohesive settings, such as calls for papers or the Web, would require specifying more attributes (e.g., “Luis Gravano, Columbia University”), in order to disambiguate entities. Thus, in a sense, the more cohesive a group is, the more reliably we can apply a relaxed matcher.

Drawing on this observation, we compute a *cohesion score* for subgraph G_A . If this score exceeds a certain threshold t , then we say the set of data sources A is highly cohesive, i.e., less ambiguous. Hence, we predict A to be relaxable, i.e. we can safely use the more relaxed matcher s_0 on A . To realize these ideas, we must address three key issues: building the “social network” graph G , computing the cohesion of any subgraph of G , and computing the relaxability threshold t . We now address these issues in detail.

Building the “Social Network” Graph: Recall that this is a graph of entities and relationships over the data sources D . We approximate this graph as follows.

1. *Apply Matcher s_1 to Match Mentions:* In the absence of any extra knowledge, matcher s_1 is the best algorithm for mention matching according to the domain expert (i.e., the user in this case). Hence, we apply s_1 to match the mentions M over the data sources D . Figure 5.a shows the people mentions in the four data sources from Example 1.1, with edges denoting predicted matches.

2. *Create Entities to Form Graph Nodes:* Next, within each data source $d \in D$, we create a set of entities. Each entity consists of all mentions in d that match (either as declared by s_1 in Step 1, or by transitivity). We treat each so-created entity as a node in the ER graph. For example, the four mentions in d_4 in Figure 5.a become two nodes in d_4 in Figure 5.b, where the pairs of matching “C. Li” and “J. Zhou” mentions in d_4 have both been consolidated. In Figure 5.b, each node is labeled by the last initial of the real-world entity it represents (e.g. “G” for Luis Gravano). Note that Luis Gravano has multiple nodes in d_1 because matcher s_1 failed to match his mentions in d_1 .

3. *Create Relationships to Form Graph Edges:* Finally, we create three types of relationship edges between the entities.

- *Equivalence:* We create an edge representing an equivalence relationship between any two entities E_1 and E_2 where at least one mention of E_1 and one mention of E_2 match (see Step 1). Figure 5.b shows edges denoting these equivalence relationships.

- *Co-occurrence:* This type of relationship captures the intuition that if the mentions of two entities often co-occur in the data, then they are likely to be related in some way. There are many schemes to compute co-occurrences. Currently we use the following. After creating equivalence relationships we have in effect created a set of cliques, such that each clique maps to one real-world entity. For any two such cliques C_i and C_j , we compute $N(C_i, C_j)$, the number of data sources where their mentions co-occur. If $N(C_i, C_j)$ exceeds a threshold k (currently set to 2), then we add an

edge (to represent co-occurrence relationship) between every two graph nodes $n \in C_i$ and $m \in C_j$. Two mentions are judged to co-occur in a data source only if they both appear in at least one data page of the source. For example, Figure 5.c show all co-occurrence edges involving Kenneth Ross when $k = 2$. That is, edges exist between nodes for Kenneth Ross and nodes for Luis Gravano and Jorge Sanz in sources d_1 and d_3 . Note that since the entity node for “K. Ross” in source d_2 is not equivalent to any other node (Figure 5.b), its mentions co-occur with those of other cliques only in one source d_2 . Thus, that node has no co-occurrence edges connected to it.

- *Domain-specific relationships*: Finally we add edges that represent any domain-specific relationships that we can infer from the data. For example, if each mention has an attribute `co-authors`, then we can infer co-author relationships between mentions, and add edges that represent this relationship between the corresponding entities.

Figure 5.d shows edges between all pairs of nodes whose entities are co-authors. For example, since Chen Li and Jian Zhou are co-authors, there is now an edge between all pairs of Chen Li and Jian Zhou nodes. The final ER graph then includes all the edges in Parts b-d of Figure 5.

Computing the Cohesion of Data Subsets: Once the graph G has been constructed, given any set A of data sources we exploit G to compute a cohesion score for A . The social network and graph theory communities have extensively discussed various notions of cohesion, and employed this notion in numerous applications (e.g., to find social cliques in large groups of people) [31]. A common intuition underlying many notions of cohesion is as follows. A group of nodes is cohesive if it has high *internal connectivity* (group members are highly related among themselves), and low *external connectivity* (group members are not very related with outsiders) (e.g., [31, 15]).

We apply this intuition to our context, and compute cohesion based on a measure proposed in [31]. Specifically, let G_A be the subgraph of G corresponding to the set of data sources A . Then, the *internal connectivity* of a set A is computed as the average pairwise distance between nodes in G_A : $iCon(A) = [\sum_{n_i, n_j \in N_A; i < j} x(n_i, n_j)] / (|N_A|(|N_A| - 1) / 2)$, where N_A is the set of nodes in G_A , and $x(n_i, n_j)$ is the distance between nodes n_i and n_j . Since G_A is unweighted, we set the distance between two nodes with at least one edge between them to 1, and to 0 otherwise. If the denominator is 0 (i.e. there is only one node), we assign it a default internal connectivity of 1. The *external connectivity* of A is computed as the average pairwise distance between nodes in G_A and outside G_A : $eCon(A) = [\sum_{n_i \in N_A; n_j \in N - N_A} x(n_i, n_j)] / (|N_A|(|N| - |N_A|))$, where N is the set of nodes in G , and $x(n_i, n_j)$ is the same distance function as used for internal connectivity.

The cohesion of A can then be computed as $C(A) = iCon(A) / eCon(A)$. If $eCon(A)$ is 0, i.e., A is not connected to any outside source, we assign it a default connectivity of 1, which intuitively neither rewards nor penalizes A 's cohesion for its lack of external connectivity.

Learning the Relaxability Threshold: Recall that if the cohesion of A , $C(A)$, exceeds a threshold t , the oracle declares that A is highly cohesive, and hence is relaxable. We now discuss how to learn t . We employ a simple active learning scheme that engages the user in binary probing as follows. First, we sort all sources in D in decreasing order of their cohesion. Suppose the resulting ranked list is d_1, \dots, d_n . Next, we set the upper bound t_u (on t) to $C(d_1)$ and lower bound t_l to $C(d_n)$. In general, suppose that the sources currently with cohesion t_u and t_l are d_i and d_j , respectively. Then, we select the source $d_{\lfloor (i+j-i)/2 \rfloor}$ (which ranks between d_i and d_j) and ask the user if it is relaxable, i.e., whether we can reliably apply matcher s_0 to it. If the user says yes, we set $t_l = C(d_{\lfloor (i+j-i)/2 \rfloor})$, otherwise we set $t_u = C(d_{\lfloor (i+j-i)/2 \rfloor})$. This continues until $j - i = 1$, at which point we set t to the average of $C(d_i)$ and $C(d_j)$.

In our experiments over three domains (Section 5), the user only had to inspect a small number of data sources (1-7), and it took the user less than 5 minutes per source to decide if a source is “relaxable”. This was because sources often come with rich human-understandable meta-data that describes the nature of mentions in the source and enables the user to quickly decide whether matcher s_0 would be appropriate. Automating this step further is a subject of our future research.

4. Finding an Optimized Plan

Ideally, we would find q_* , the globally optimal plan in the plan space. However, finding q_* may be intractable, as we briefly discuss later. Hence, we aim instead to find a plan that is likely to significantly outperform the default plan q_{def} (one that applies s_1 to data sources D).

Intuitively, we can improve q_{def} by using the relaxation oracle to find all relaxable subsets of data sources, then applying the more relaxed matcher s_0 to these sources, before applying s_1 . Specifically, let A_1, \dots, A_k be all subsets of D that the relaxation oracle predicts to be relaxable, and let

$$q_r = s_1(s_0(A_1), s_0(A_2), \dots, s_0(A_k), (D - \cup_{i=1}^k A_i)) \quad (1)$$

be a plan that applies s_0 to the relaxable sets A_1, \dots, A_k , unions the results with $(D - \cup_{i=1}^k A_i)$, then applies s_1 to the union. Then q_r is likely to be better than q_{def} (and provably so if the oracle is perfect, i.e., if all of its relaxability predictions are true). Furthermore, since q_r applies s_0 to *all* relaxable subsets of D , it should be optimal in some sense. Indeed, we show later that under certain general conditions it is optimal among all plans reachable from q_{def} using a small set of rewriting rules.

So q_r is likely a good plan. However, finding it turns out to be still prohibitively expensive. Constructing q_r reduces

to finding all relaxable subsets of D . The naive solution is to enumerate all $2^{|D|}$ subsets of D and call the relaxation oracle to predict the relaxability of each. This is clearly impractical for large D . Furthermore, the approach of generating all relaxable subsets in a compositional fashion (e.g., by combining two relaxable sets into a larger relaxable one) is also unlikely to work, because it is not difficult to prove that (a) a set A being relaxable does not imply that all of its subsets are relaxable, and (b) sets A and B being relaxable does not imply that $A \cup B$ is relaxable.

Given these observations, we turn to approximation. We implement algorithm **GFinder**, which produces a plan q_+ that approximates q_r .

The GFinder Algorithm: **GFinder** does not find all relaxable subsets of D , but it does find a substantial number of such sets, in a greedy fashion. It then constructs plan q_+ using Formula 1, but replaces the sets A_1, \dots, A_k in the formula with those relaxable sets that it has found. Thus, q_+ is just an approximation of q_r . Our experiments (Section 5) show that q_+ still significantly outperforms q_{def} .

Specifically, **GFinder** starts by using the relaxation oracle to find all relaxable *individual* data sources. Let these sources be d_1, \dots, d_m . Then **GFinder** initializes a set $D_+ = \{S_1, \dots, S_m\}$, where each set $S_i = \{d_i\}$ is a relaxable set of size one. Next, **GFinder** greedily grows the sets in D_+ . For each pair S_i and S_j in D_+ , **GFinder** employs the relaxation oracle to compute the cohesion of their union, then selects the pair with the highest cohesion. Suppose this pair is S_1 and S_2 . If $S_1 \cup S_2$ is declared relaxable by the oracle, then **GFinder** removes S_1 and S_2 from D_+ , replaces them with $S_1 \cup S_2$, then repeats the above “growth” step. If $S_1 \cup S_2$ is declared unrelaxable, or if all the original sources in D_+ have been merged into one set, **GFinder** stops this process to “grow” relaxable sets.

Suppose D_+ consists of relaxable sets A_1, \dots, A_p when the above process terminates. Then **GFinder** returns the plan $q_+ = s_1(s_0(A_1), s_0(A_2), \dots, s_0(A_p), (D - \cup_{i=1}^p A_i))$.

Guarantees of Optimality: **GFinder** produces a plan q_+ that captures our intuition about applying more relaxed matchers to relaxable sets of sources to obtain a plan that is better than q_{def} . We now show that under certain general conditions, we can prove this. We also prove that plan q_r (which we try to approximate with q_+) is optimal among all plans reachable from q_{def} using a set of rewriting rules. Finally, we briefly show that finding the globally optimal plan q_* may be intractable. This suggests that finding plans that outperform q_{def} might be a more fruitful research direction than trying to find the globally optimal plan q_* .

We start by defining two intuitive properties of a matching problem: *globality* and *orderliness*. Globality captures the intuition that global matching should override the predictions of any *local* matching using the *same* matcher:

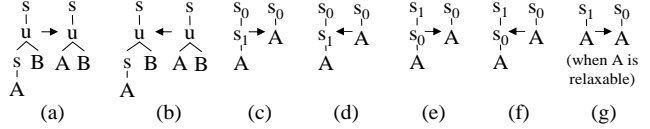


Figure 6. Seven rewrite rules for match plans.

Definition 7 (Globality) A matching problem with matchers S has the globality property if and only if for all $s \in S$, any plan fragment $s(s(A), B)$ can be rewritten as $s(A, B)$ (and vice versa) to obtain a plan with the same accuracy.

Orderliness states that a relaxed matcher should override the predictions of more conservative matchers. It reflects our intuitive understanding of the relaxability of matchers:

Definition 8 (Orderliness) A matching problem with matchers s_1 and s_0 – where s_0 is more relaxed than s_1 – has the orderliness property if and only if any plan fragment $s_0(s_1(A))$ or $s_1(s_0(A))$ can be rewritten as $s_0(A)$ (and vice versa) to obtain a plan with the same accuracy.

Globality and orderliness very commonly hold in practical matching scenarios (e.g., they hold for all matching scenarios in our experiments). Given these properties, we now can prove that plan q_+ outperforms q_{def} :

Theorem 1 (Quality of Plan q_+) If globality and orderliness hold, and if the relaxability predictions of the oracle are accurate, then plan q_+ as found by **GFinder** achieves equal or higher accuracy than plan q_{def} .

Regarding plan q_r , observe that Definitions 6-8 in effect define seven rewriting rules that we can use to rewrite plans (see Figure 6). We can now prove that:

Theorem 2 (Optimal Reachable Plan) Assume that globality and orderliness hold and that the relaxability predictions of the oracle is accurate. Then plan q_r as defined in Equation 1 is optimal among all those plans reachable from q_{def} , using the above seven rewriting rules.

It immediately follows that q_r is equivalent to or better than q_+ , since q_+ can also be reached from q_{def} using the above rewriting rules. Regarding global optimality, we can prove the following:

Theorem 3 (Shape of the Globally Optimal Plan) If globality and orderliness hold, then any plan involving matchers s_1 and s_0 can be rewritten into an equivalent plan of the form $s_1(s_0(A_1), s_0(A_2), \dots, s_0(A_k), (D - \cup_{i=1}^k A_i))$. Consequently, we only need to look for the globally optimal plan q_* among those of the above form.

An immediate question then arises: since plan q_r also has this shape, is it globally optimal? The answer is no. In particular, it is possible to construct matching scenarios where the globally optimal plan q_* applies s_0 to a *non-relaxable* subset A . This suggests that q_* is not always q_r (since the latter applies s_0 only to relaxable subsets). It further suggests that it may be intractable to find q_* , given the large space of possible candidates and little observed regularity in

	DB Researchers (189 data sources)			PIM (30 data sources)			Movies (151 data sources)	
	People	Articles	Venues	People	Articles	Venues	Actors	Movies
# Mentions	14444	2624	4040	18386	329	329	3945	3494
# Correct Pairs	192130	1125	48724	2301495	113	225	21569	2328

(a)

DB Researchers

People: <name, co-authors, venue>

Articles: <name, year, venue>

Venues: <name, year>

Movies

Actors: <name, related actors, movies>

Movies: <name, actors, year>

PIM

People: <name, email address, articles, co-authors, email contacts>

Articles: <name, year, authors, venue> Venues: <name, year, articles>

(b)

Figure 7. (a) Three real-world domains used in our experiments and (b) the schemas for each domain.

that space (e.g., any plan that applies s_0 to a non-relaxable subset can potentially be a candidate).

Handling Multiple Types of Mentions: We have described applying Soccer to match mentions of a single type. Recent work has shown that in settings with multiple types of mentions, matching mentions of all types together can improve accuracy [13, 21, 5, 28, 29]. Hence, we consider extending Soccer to collectively match mentions of multiple types. To do so, for each mention type t , we now assume two matchers s_0^t and s_1^t . Next, we find an optimized match plan q_+^t for each mention type t , by proceeding as described earlier in Sections 3-4. Finally, to match the mentions we perform only one modification to a multi-type matching algorithm. For any matching decision between mentions m_i and m_j of type t , we consult the match plan q_+^t . If q_+^t applies s_0^t to a subset that contains m_i and m_j , we apply the match function from s_0^t to m_i and m_j ; otherwise we apply the function from s_1^t . In a sense, this is similar to “executing” the plans for all the mention types simultaneously.

5. Empirical Evaluation

We now experimentally compare Soccer and current matching solutions, and examine the relative utility of the various Soccer components,

Data Sets: Figure 7 describes three real-world data sets for our experiments. Researchers consists of 189 data sources that cover database research-related activities (e.g., researcher homepages, conference websites, DBworld, part of DBLP, etc.). We built wrappers to extract mentions of people, articles, and venues, resulting in 14,444 people mentions (Figure 7). We also manually identified all mentions that correctly belong to each entity. This resulted in 192,130 correct pairs of matching people mentions. Note that this manual result is used only to evaluate the accuracy of matching algorithms.

We proceeded similarly to create PIM and Movies data sets. The 30 PIM data sources came from personal data of one of the authors (currently there are no public PIM data sets). They include email folders (each is treated as a source), LaTeX and Bibtex files, and Unix directories,

among others. The 151 Movies data sources include actor home pages in IMDB (*imdb.com*), sources obtained from querying Yahoo Movie Search with actor names, and sources containing movie related news articles.

Matching Algorithms: For comparison, we use Semex, a state-of-the-art matching algorithm [13]. Semex works well for mentions with missing attribute values, with set-values, and for PIM contexts. For each data set, we first tuned Semex to maximize its accuracy, by designing the best possible similarity measure and tuning its parameters on set-aside data. This essentially simulated tuning as carried out by a domain expert. For Soccer, we then treat the tuned Semex as the “conservative” matcher s_1 . Next, we replaced the similarity measure in Semex with a more “relaxed” similarity measure which match mentions based only on their names. This more “relaxed” version of Semex is then treated as matcher s_0 . Finally we applied Soccer with matchers s_0 and s_1 . Our goal is to see if Soccer outperforms the tuned Semex, and if so, by how much.

5.1. Matching Accuracy

Figure 8 shows the accuracy of Semex versus Soccer. Each numeric cell lists F_1 , precision, and recall, in that order. Rows “Semex” and “Soccer” show that Soccer achieves comparable or significantly higher accuracy than Semex in all eight cases (except on “venue” for Researchers, where its F_1 was lower by 0.3%).

In four cases, F_1 improvement ranges from moderate (3.2%) for “people” in PIM, to quite substantial (9.2-15.2%) for “people” and “articles” in Researchers and “actors” in Movies. Since Soccer could apply matcher s_0 on selected subsets of data sources, it was able to increase recall significantly, while only minimally hurting precision. For instance, for “people” of Researchers, Soccer improves recall by 16.8%, while reducing precision by only 1.2%. For “actors” of Movies, recall increases dramatically from 62.6% to 84.9%, and precision also slightly rises from 79.9% to 80.7%. The remaining four cases show no or little F_1 improvement. In some cases Semex already achieves very high F_1 (e.g., 95.2-99.6%), leaving little room for Soccer to improve. In others (e.g., “movies”), Soccer was not able to find many relaxable sources (to which it could apply s_0).

Finally, Row “# questions” of Figure 8 shows that Soccer required only minimal feedback from the user (1-7 questions), to learn the relaxation threshold.

5.2. Contributions of Soccer Components

We now examine the relative contributions of Soccer components. For each entity type (e.g, “person”, “article”, etc.), Figure 9 lists respectively the accuracy of Semex, Soccer₁, Soccer₂, and Soccer.

Soccer₁ is the “barebone” version of Soccer that applies matcher s_0 only to the *individual* data sources that are re-

	DB Researchers			PIM			Movies	
	People	Articles	Venues	People	Articles	Venues	Actors	Movies
Semex	.875 (.999/.779)	.786 (.920/.685)	.893 (.846/.945)	.929 (.997/.870)	.996 (1.0/.991)	.952 (.981/.924)	.702 (.799/.626)	.931 (.972/.893)
Soccer	.967 (.987/.947)	.938 (.923/.954)	.890 (.829/.962)	.961 (.978/.944)	.996 (1.0/.991)	.952 (.981/.924)	.827 (.807/.849)	.937 (.971/.906)
# questions	7	5	7	4	1	1	5	7

Figure 8. Matching accuracy of Soccer versus Semex. Accuracy is shown as $F_1(P/R)$.

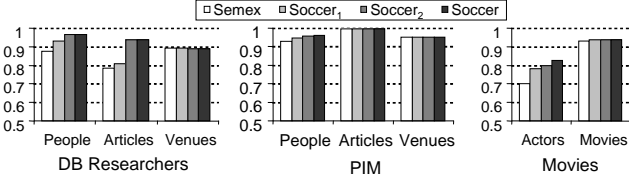


Figure 9. Matching accuracy of Semex versus three Soccer variations.

laxable (not to *groups* of data sources, as in Soccer). Figure 9 shows that, compared to Semex, applying s_0 to the individual sources already increases accuracy significantly by 0.6-8% F_1 in five cases, without hurting the remaining three cases. This often happens when some individual data sources contain a significant amount of information about various entities (e.g., an almost complete list of co-authors of a researcher). In these cases applying s_0 to such sources improves recall significantly, without hurting precision.

Recall that Soccer attempts to maximize accuracy by finding the largest possible groups that are relaxable. Is it important to find the *largest* possible groups? We explored this issue with Soccer₂. In this version we modified GFinder so that in each iteration, instead of selecting the pair with the highest cohesion, it selects a random pair of sources. In essence, Soccer₂ does not attempt to find the largest possible groups of sources that are relaxable, but only examines random groups and keeps those that are relaxable. Compared to Soccer₁, Soccer₂ significantly increases accuracy by 0.8-12.8% F_1 in four cases. This often happens when the data is “dense”, such that even two randomly selected sources, when combined, are likely to contain a significant amount of information about entities. However, for people in PIM and actors in Movies, Soccer further improves accuracy over Soccer₂ by 0.5% and 3%, respectively, suggesting that in some cases it is indeed beneficial to be even more aggressive and find the largest possible groups of data sources that are relaxable.

5.3. Scalability of Soccer

The first line of Figure 10.a shows the time it takes for Soccer to generate the optimized plan q_+ (Section 4), as we increase the number of data sources from 10 to 180 (these sources were randomly sampled from Researchers domain). The results show that this time grows quite modestly and that Soccer can generate an optimized plan q_+ in a reasonable amount of time (e.g., under three minutes for 180 sources, using our unoptimized Soccer version). This plan generation time is composed of (a) the time it takes to

run matcher s_1 (the second line of Figure 10.a), and (b) the time it takes using the predictions of s_1 to construct the social network and to run GFinder algorithm, excluding the time spent interacting with the user to learn the relaxability threshold (the third line of Figure 10.a). The results show that running matcher s_1 clearly dominates the time Soccer takes to generate plan q_+ . In contrast, constructing the social network and running GFinder takes far less time. Much research has focused on improving the run time of current matching solutions (e.g., [19, 4, 26]). Since we often use these solutions as matcher s_1 in Soccer, the above analysis suggests that advances in the above line of research should also help significantly reduce the plan generation time of Soccer.

In the next step, we compare the run time of the optimized plan q_+ produced by Soccer with that of the default plan (matcher s_1 in our case). The results (see Figure 10.b) show comparable run times. In three cases, Soccer actually took less time. This is mostly due to using matcher s_0 , which is not as expensive timewise as matcher s_1 .

6. Related Work

Numerous mention matching solutions have been developed, under a variety of names: record linkage, (fuzzy) tuple matching, entity matching, merge/purge, reference reconciliation, and others (e.g. [14, 2, 19, 13, 25, 30, 22, 17, 32, 8, 11], see [23] for a recent survey). These works have largely employed a single matching technique. Soccer takes a next logical step of treating each proposed technique as a “blackbox” matcher, then showing how multiple matchers can be employed to improve matching accuracy. In doing so, it exploits information about data sources, which to the best of our knowledge has not been considered by prior work.

Social network analysis [31] has been applied to a variety of data management applications, most notably to Web search [27] and keyword search in relational and XML databases [18, 3]. Recent work [20] analyze the link structure of a social network to disambiguate the mentions represented by the network nodes. In contrast, we employ social network analysis to determine the semantic properties and relatedness of data sources.

Soccer proposes a compositional, multi-component approach to mention matching, taking cues from the compositional nature of relational data management. The work [4] has applied this compositional approach - however, the goal of this work is to optimize for *run-time efficiency* of a sin-

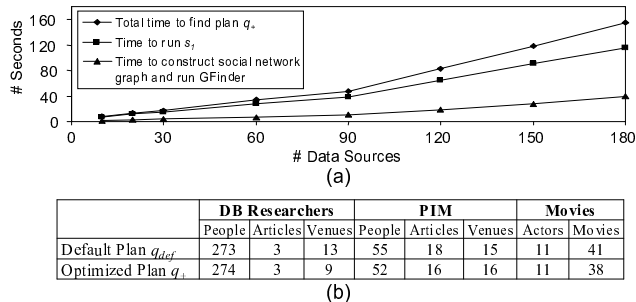


Figure 10. (a) Time for Soccer to generate the optimized plan q_+ , and (b) run time of q_+ versus that of q_{def} .

gle matcher, by minimizing the calls to a similarity function that compares mentions. In contrast, our work focuses on using multiple matchers to optimize *accuracy*. Recent systems [16, 14] have proposed compositional frameworks for data cleaning. However, the main goal of these frameworks is to allow the user to specify and tune plans *manually*. In contrast, our work focuses on optimizing mention matching plans automatically with minimal user feedback.

7. Conclusion & Future Work

Current mention matching approaches have largely exploited only information within the mentions and employed a single matching solution. We have described Soccer, a novel approach that exploits information about the data sources and employs multiple matching solutions to significantly improve matching accuracy. Soccer casts mention matching as finding a good match plan in a large plan space, where each plan specifies which matching solution is applied to which subset of data sources, and how sources are grouped. Soccer leverages ideas from social network analysis to efficiently find a good plan. Extensive experiments over three real-world data sets show that our solution significantly outperforms state-of-the-art matching methods. As future work, we plan to further develop compositional mention matching approaches. In particular, we plan to consider more expressive match plan spaces (e.g., considering more than two matchers, or subsets of data not just at the data-source level). We also plan to consider more general utility functions, such as those that combine both execution time and matching accuracy.

References

- [1] E. Agichtein and V. Ganti. Mining reference tables for automatic text segmentation. In *KDD-04*.
- [2] P. Andritsos, A. Fuxman, and R. J. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE-06*.
- [3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB-04*.
- [4] O. Benjelloun, H. Garcia-Molina, Q. Su, and J. Widom. Swoosh: A generic approach to entity resolution. Technical report, Stanford University, March 2005.
- [5] I. Bhattacharya, L. Getoor, and L. Licamele. Query-time entity resolution. In *KDD-06*.
- [6] M. Bilenko, R. J. Mooney, W. W. Cohen, P. Ravikumar, and S. E. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 2003.
- [7] V. Borkar, K. Deshmukh, and S. Sarawagi. Automatic text segmentation for extracting structured records. In *SIGMOD-01*.
- [8] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE-06*.
- [9] S. Chaudhuri, V. Ganti, and R. Motwani. Robust identification of fuzzy duplicates. In *ICDE-05*.
- [10] W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD-98*.
- [11] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Quality*. Wiley, 2003.
- [12] A. Doan, R. Ramakrishnan, F. Chen, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. *IEEE Data Engineering Bulletin*, 29(1), 2006.
- [13] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD-05*.
- [14] M. G. Elfeke, V. S. Verykios, and A. K. Elmagarmid. TAILOR: A record linkage toolbox. In *ICDE-02*.
- [15] G. W. Flake, R. E. Tarjan, and K. Tsioutsouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 2004.
- [16] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB-01*.
- [17] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB-01*.
- [18] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRank: Ranked keyword search over XML documents. In *SIGMOD-03*.
- [19] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD-95*.
- [20] R. Holzer, B. Malin, and L. Sweeney. Email alias detection using network analysis. In *SIGKDD Workshop on Link Discovery: Issues, Approaches, and Applications*, 2005.
- [21] D. V. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM-05*.
- [22] N. Koudas, A. Marathe, and D. Srivastava. Flexible string matching against large databases in practice. In *VLDB-04*.
- [23] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: Similarity measures and algorithms (tutorial). In *SIGMOD-06*.
- [24] X. Li, P. Morie, and D. Roth. Identification and tracing of ambiguous names: Discriminative and generative approaches. In *AAAI-04*.
- [25] W. L. Low, M.-L. Lee, and T. W. Ling. A knowledge-based approach for duplicate elimination in data cleaning. *Inf. Syst.*, 2001.
- [26] A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD-00*.
- [27] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.
- [28] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In *NIPS-03*.
- [29] P. Singla and P. Domingos. Object identification with attribute-mediated dependences. In *PKDD-05*.
- [30] S. Tejada, C. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *KDD-02*.
- [31] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.
- [32] M. Weis and F. Naumann. Dogmatix tracks down duplicates in XML. In *SIGMOD-05*.
- [33] Y. Zhai and B. Liu. Extracting web data using instance-based learning. In *WISE-05*.