Critical Points for Interactive Schema Matching

Guilian Wang[1], Joseph Goguen[1], Young-Kwang Nam[2], and Kai Lin[3]

[1] Department of Computer Science and Engineering, U.C. San Diego,
{guilian, goguen}@cs.ucsd.edu
[2] Department of Computer Science, Yonsei University, Korea,
yknam@dragon.yonsei.ac.kr
[3] San Diego Supercomputer Center, U.C. San Diego, klin@sdsc.edu

Abstract. Experience suggests that fully automated schema matching is infeasible, especially for n-to-m matches involving semantic functions. It is therefore advisable for a matching algorithm not only to do as much as possible automatically, but also to accurately identify the critical points where user input is maximally useful. Our matching algorithm combines several existing approaches with a new emphasis on using the context provided by the way elements are embedded in paths. A prototype tested on biological data (gene sequence, DNA, RNA, etc.) and on bibliographic data shows significant performance improvements from utilizing user feedback and context checks. In non-interactive mode on the purchase order schemas used in the COMA project, it compares favorably, and also correctly identifies critical points for user input.

1 Introduction

Many scientific and commercial applications use multiple distributed information sources with metadata, of which schemas are the most important, and finding semantic correspondences between different schemas is a critical step in many such applications, including 1) data warehousing, to find correct data transformations from the source schemas to a single target schema; 2) virtual data integration, to provide a basis for rewriting user queries over a mediated schema to optimized sub-queries over the source schemas (often called query discovery) [6, 7]; 3) schema integration, to find similar structures (or "integration points") across multiple schemas [1]; and 4) e-business or scientific workflow, to identify semantically correct mappings of messages, often in XML format, between steps.

The **schema matching problem** is to identity semantically corresponding elements in a set of schemas, possibly using some auxiliary information [3, 5, 15]. This can be very difficult, since even schemas for the same entities from different sources may have very different structural and naming conventions, and may also use different data models. Moreover, similar, or even the same, labels may be used for schema elements having totally different meanings, or having subtly different semantics, due to differences in the unit, precision, resolution, aggregation, measurement protocol, etc.; this is extremely common in domains like environmental science, ecology, biology, and commerce.

There can be problems beyond naming and structural differences. For example, in bibliographic schemas, price units may differ, items may be missing (e.g., price, publication date, or publisher name when the source consists of that

publisher's books), author names may be full or separated into first and last, and some may have a different scope, e.g., including articles as well as books. Scientific domains often have indirect matches; e.g., in ecology, species density equals species count divided by area, so one schema could have a density column, while another has count and area columns, although both schemas are for biodiversity. Hence a single element in one schema may correspond to multiple elements in another, and even multiple elements in one schema can correspond to multiple elements in another; such matches are called n:m.

Moreover n:m matches may have different meanings in different contexts. For example, `PurchaseOrder1/OrderHeader/Contact` from one company may relate to `PurchaseOrder2/BillTo/Contact` and to `PurchaseOrder2/ShipTo/Contact` from another company, and contact information from the first may be the simple union of two contacts from the second. These are 1:2 matches that can be simplified to two 1:1 matches, `([contact, billTo/contact], sim)` and `([contact, shipTo/contact], sim)`, and hence can be treated like other 1:1 matches. On the other hand, the correspondences of full author name to concatenation of first and last names, and of species density to count divided by area involve conversion functions, and should be represented as 1:2 correspondences, `([name, first concat last], sim)`, `([density, count div area], sim)` respectively. Unfortunately, most existing schema matching tools treat all these cases the same way, considering `[name, firstname]` and `[name, lastname]` as correct matches, e.g., called "global n:m matches" where n = 1 and m = 2, in COMA [3, 18], and other systems.

When one schema element is related to multiple elements in another, whether a real multiple match exists, and whether it is appropriate, can be very difficult to determine, due to various forms of context dependency. In the purchase order schema matching problem, `Contact` in the purchase order header of one schema may correspond to both billing `Contact` and shipping `Contact` in another, although the `Contact` for the purchase order is different from the billing or shipping `Contact`. On the other hand, matching `Contact` in the purchase order header to the supplier `Contact` is not desirable. Also, although `Bib/Book/Author` in one schema is very similar to both `arts/article/author` and `arts/book/author` in another, only the `Bib/Book/Author` to `arts/book/author` correspondence makes sense. Such examples strongly suggest that totally automatic matching tool is infeasible, since some correspondences are correct only under subtle conditions that are usually infeasible to infer from just the schema and/or other straightforward auxiliary information.

Existing automatic tools mainly help discover simple 1:1 matches, without considering data semantics, or how the generated mapping will be used, and thus often require significant manual effort to correct wrong matches and add missing matches. In practice, schema matching is still done manually by domain experts, usually with a graphical tool [11, 13], and is very time consuming when there are many data sources or when schemas are large and/or complex.

The goal of schema matching systems is to save manual effort, but the total manual effort includes work performed to prepare schema and auxiliary infor-

mation, as well as guiding the match process, and editing the results to get a correct mapping. However, existing metrics do not measure total manual effort. The most common metrics only compare accuracy and completeness of match results. However, a system that can identify the critical points where input can most help the matching process could save more manual effort than another system having the same quality metrics; a system that asks users for input where matches can not be automatically determined can save more manual work than a system that generates wrong matches at those places, even if it provides some guidance. Moreover, because schemas themselves often evolve, as well as schemas being added or deleted, a matching system that incrementally updates mappings based on existing mappings can save much more effort than a system that always treats matching as a new task.

Our approach is to discover simple correspondences using a combination of existing matching techniques, and prompting the user to provide input at critical points when no adequate match can be determined, e.g., when there is more than one almost equally good match, or when the contexts embedded in possibly corresponding paths are not consistent with each other, or where n:m correspondences and/or functions may be needed. Context information provided by the paths of schema elements is explicitly used to improve simple match accuracy and detect critical points where user input may be most helpful for n:m correspondences that may involve functions. A research prototype has been tested on bibliographic, biological, and purchase order data, with encouraging results, though it should be noted that not all the features we discuss have as yet been fully implemented.

2 Related work

An extensive review of techniques and tools for automatic schema matching is given in [15], which classifies tools by how they use similarity of information in schemas (element name, structure, data type, constraints), data content, and auxiliary information (e.g., general or domain specific common terminologies), and by whether one or several combined techniques are used. LSD and GLUE use machine learning to evaluate data instances and train matchers, and then predict element similarities by combining their results [2, 4]. Cupid combines name and structural matching, and predicts element similarity based on the similarity of the name and data type of their components (sub-elements and attributes in XML DTD or Schema), which can easily lead to incorrectly identifying atomic elements (leaves in schema tree) [5]. SF [12] and Rondo [9] use a versatile graph matching algorithm, also used in our prototype, for structural matching, and introduce more realistic metrics for match accuracy, which are also used by COMA and by us. COMA supports flexibly combining different schema matching techniques, reuse of previous matching results, and user input during matching; it has been carefully tested for real purchase orders schemas, showing that reuse can help a lot [3]. Other approaches [19] emerged after [15]. All these tools only find 1:1 matches, and have great difficulty with matches

that involve conditions, or conversion functions. [17] and [16] use ontologies to discover some indirect matches involving composition or decomposition of multiple elements based on their ontological descriptions, e.g., sample values and keywords, and require users to provide the ontologies. But since users can differ greatly in their match judgments, it is very likely that different individuals would provide very different ontologies, resulting in variable effectiveness of the technique. Moreover, ontologies are harder to write and debug than schemas, and are subject to evolution almost as much as schemas. [16] also argued that user interactions are desirable for deciding many issues in its source schema to target ontological view mapping framework.

3 Our approach

The most novel aspects of our approach are its support for n:m matches that involve semantic functions, and its focus on providing maximum support to users, rather than total automation. It enhances, combines, and reuses algorithms for linguistic, structural, and graph matching from prior work to discover as many good matches automatically as possible, and more importantly, to detect where user input may be most valuable and identify what specific input to request from users. Its structural matching considers closely connected nodes, as in ARTEMIS, COMA, Cupid, DIKE, and Rondo. It has a series of matching stages using different methods, as in COMA and Cupid, and results from different methods are combined by weighted summation. L linguistic similarity of elements is based not just on their tag strings, but also on their path strings, as in COMA, so that hierarchical schema structure is treated as linguistic, and paths provide context dependent matches for shared elements, as in Cupid. We represent n:m match as pairs `([exp1, exp2], sim)`, where `exp1` is an expression over n target paths (typically a single path), `exp2` is a expression over m source paths, `sim` is a value in the range of [0, 1] measuring semantic similarity of `exp1` to `exp2`, with 1 for most similar and 0 for least similar. This is similar to match representation in the DDXMI system [13], though it is not yet integrated with the schema matching software. Examples are:

```
([bib/book/price, bookstore/book/price div (100)], 0.85)
([PO2/contact, PO2/header/contact/firstname concat
  PO2/header/contact/lastname], 0.65)
([Sequences/Sequence/Segment, /Bsml/Definitions/Sequences/Sequence/
  Feature-tables/Feature-table/Feature], 0.74)
```

The following are some technical differences from prior work:

a. Schemas are represented in both tree and graph formats: the tree format is convenient for context and subtree matching, while the graph format accommodates additional schema information, including element type (attribute, atomic, or context), data type, and constraint.

b. Specific user input is interactively requested at critical points, not just at pre- and/or post-match, and user supplied matches are not modified later; this makes post-match editing much easier, since bad guesses made without user guidance propagate less.

c. Graph matching gives new matches based on user supplied and/or high similarity value matches from previous stages, without changing these "good matches" in its fix-point computation and later stages.

d. Since tag meanings vary with context, and context is given by higher elements, we seek to identify **core** contexts and attributes, the most important contextualizing elements for tags within subtrees, by using heuristics and user input; then threshold and context are checked for them. For example, in matching two book DTDs, the contexts `/arts/book` and `/arts/article` are found to be core contexts, with title, author, publisher as their main attributes. Previous steps found that the best match with `/arts/article` is `/bookstore/book`, but its similarity value is lower than the core threshold, so that matches in its subtree are not reliable, and user input is needed. Matching `/arts/book/publisher/name` with `/bookstore/book/author/name` fails because they are in different contexts, even though they have the same tag, and there is no better match for `/arts/book/publisher/name`.

e. Matching two schema trees is divided into matching their subtrees rooted at core context nodes. This improves manageability, and provides a general framework for discovering and maintaining n:m matches.

f. When refining subtree matches in context checking, if the roots of 2 subtrees are considered a "good" match, then the path of an element in either of the subtrees may be changed into a shorter one starting from the corresponding subtree root instead of starting from the root of the whole tree. This helps reduce the impact of heterogeneous structure on path similarity of elements in the subtrees. For example, in matching two bio-molecular DTDs, after finding that `/Bsml/Definitions/Sequences/Sequence/Feature-tables/Feature-table/Feature` matches `/Sequences/Sequence/Segment`, then for all nodes in the subtree of Feature, the string before `/Feature` in its path is cut off, and for all the nodes in subtree of Segment, the string before `/Segment` in its path is also cut off. This should yield higher path similarity values for `(Feature/id, Segment/seg-id)`, `(Feature/Location/Interval-loc/startpos, Segment/seg-start)`.

g. Pre-order traversal gives top-down matching, which is more efficient than bottom-up, and our combination of techniques, earlier identification of core contexts, and user input, all help to improve performance on schemas that differ at higher levels.

Figure 1 shows our structure for matching a source schema S and a target schema T. One or more iterations may be executed, depending on whether the mode is automatic or interactive. In interactive mode, matching strategy selection and match candidate determination are done interactively, requesting specific user input at critical points, while in automatic mode, everything is done by using default strategies. Each iteration has 5 steps: optional user interaction, linguistic and data type matching, structural matching, optional context check, and combination of match results; the last four steps also have optional user interactions. Match processing and the input preprocessing are described in the following subsections.

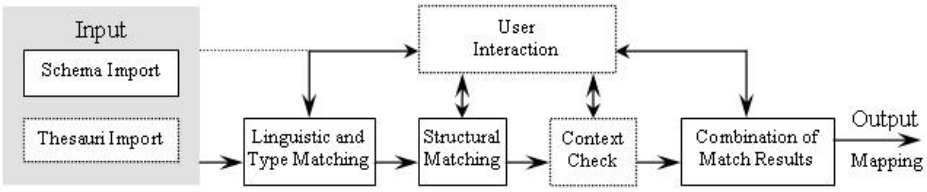

Fig. 1. Diagram of match processing

3.1 Input Preprocessing

The input consists of target and source schemas, and optional domain thesauri. The **Schema Import** module produces a tree and a graph representation of each schema. In the tree representation, each element and attribute of a schema is a node, and sub-elements are child nodes. Each node has its type (attribute, leaf, or parent that is possibly a context), data type, height, constraints, two sets of matches (n_j, *sim*) for node n_j in the other tree, for pre-context-check and post-context-check respectively, etc.

The graph representation uses the RDF (directed labeled graph) model. Each element and attribute that is a node n_i becomes an RDF resource, res_{ni}. Each node type nt_i and data type dt_i become RDF resources res_{nti} and res_{dti} respectively. Corresponding edges are created for each node and added to the graph, two for its parent if it is not the root node, (res_{ni}, $parent\_height_{ni}$, $parent_{ni}$) and ($parent_{ni}$, $child\_height_{ni}$, res_{ni}), one for its node type, (res_{ni}, *nodeType*, res_{ntni}), one for its data type, (res_{ni}, *dataType*, res_{dtni}), etc.

3.2 User interaction

Users can optionally provide some good matches or mismatches, before the initial iteration. If the target element and the source element have significantly different names in a specified good match, then users are asked for input to update the domain thesauri.

3.3 Linguistic and data type matching

In the current prototype, linguistic matching includes: (1) terminological matching, in which element name and path are normalized through domain specific thesauri containing a synonym table, an abbreviation table and a hypernym table; and (2) syntactic matching, which is purely based on string similarity, computed according to the number of substrings they have in common; the implementation in [9] is borrowed. For any pair of nodes (n_i, n_j) in the two trees, both normalized name similarity value $ns_{i,j}$ and normalized path similarity value $ps_{i,j}$ are computed.

$$ns_{i,j} = Coeff \times StringMatch(Normalize(name_{ni}, name_{nj}))$$
$$ps_{i,j} = Coeff \times StringMatch(Normalize(path_{ni}, path_{nj}))$$

where $Coeff = 0.8$ if $name_{ni}$ or $name_{nj}$ happens to be hypernymic to the other, otherwise $Coeff = 1.0$.

$dts_{i,j} = Compatibility(dt_{ni}, dt_{nj})$

The linguistic and data type similarity value, $ls_{i,j}$, is the weighted sum of name, path and data type similarities. $ls_{i,j} = ns_{i,j} \times w_{n\_} + ps_{i,j} \times w_{p\_} + dts_{i,j} \times w_{dt\_}$ where $w_{n\_}$,$w_{p\_}$ and $w_{dt\_}$ are the weight values for name, path and data type similarity respectively, at the linguistic and data type matching steps, and $w_{n\_} + w_{p\_} + w_{dt\_} = 1.0$.

If there are multiple source nodes with big subtrees having almost the same linguistic and data type similarity to a target node that also has a big subtree TST, and if these subtrees are also similar, based on their share of linguistically similar nodes, then this might indicate that the target subtree TST potentially corresponds to multiple source subtrees. Questions about whether there exist multiple match subtrees for this target subtree, which of the similar ones are good ones, and whether there are more, should be posed to users. After user input on matching subtrees, SST_i ($0 \leq$ i $\leq$ k), for target subtree TST, the original problem of matching source schema tree S and target schema tree T is divided into k+1 smaller matching problems: matching $T - TST$ with $S - \sum SST_i$, matching TST with each SST_i for $0 \leq i \leq k$.

3.4 Structural matching

The Similarity Flooding algorithm for graph matching in [12] was extended to structural matching for our prototype. Taking as input graphs G_t, G_s, and a set of initial similarity values, *InitMap*, between the nodes of the graphs, it iteratively propagates initial similarity values of nodes to surrounding nodes, using the intuition that neighbors of similar nodes are similar, and finally returns the structural similarity $gs_{i,j}$ of any node n_i in G_t and any node n_j in G_s. The original algorithm was modified to freeze input mappings from the user and/or of high similarity from previous steps for some nodes. The initial mapping *InitMap* includes all user input matches, and the best candidate with $max\_ls_i \leq th_{l\_g}$ for each target node from the linguistic and data type matching step, where th_{l_g} is a similarity threshold. Thus

$gs_{i,j} = GraphMatch(G_t, G_s, InitMap)$

The quality of *InitMap* has a very important influence on the output of the structural matcher; users can also improve quality by confirming good matches, denying bad ones, and adding new ones. The following shows how this works. The s_bib.dtd and s_arts.dtd in Figure 2 are simplified to provide more readable graphs. Their representations are shown in Figure 3 (the edge label *cl1* stands for "child at level 1",*dt* for "data type"). Here the *InitMap* contains `([bib/book/title, arts/book/title], 0.77)` from the linguistic matching step, and built-in matches for DTD node and data types, such as ([`parentElem, parentElem`], 1.0), ([`leafElem, leafElem`], 1.0), ([`attribute, attribute`], 1.0), ([`PCDATA, PCDATA`], 1.0), ([`CDATA, CDATA`], 1.0).

Figure 4 (a) shows the pairwise connectivity graph. The positive value on top of a matching pair node is its initial similarity, and is zero if not shown. The propagation graph in Figure 4 (b) is induced from this pairwise connectivity graph (but the added propagation edges of the 7 nodes at lower part are not

<!ELEMENT bib (book*)>
<!ELEMENT book (title)>
<!ATTLIST book year CDATA #REQUIRED >
<!ELEMENT title CDATA>

<!ELEMENT arts (book| article) >
<!ELEMENT book (title)>
<!ATTLIST book date CDATA #REQUIRED >
<!ELEMENT title CDATA>
<!ELEMENT article (title)>
<!ATTLIST article date CDATA #REQUIRED >

Fig. 2. s_bib.dtd and s_arts.dtd

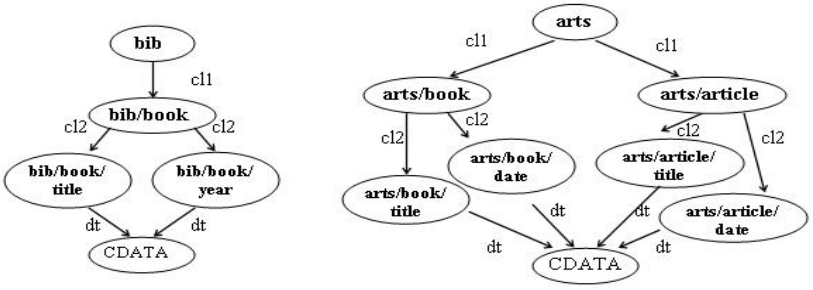

Fig. 3. The graphs for s_bib.dtd and s_arts.dtd

shown, in order to reduce the complexity). For each edge (*source, label, target*), an opposite edge is added for propagating *target* node's similarity to the *source* node. The edge *label* is changed to a *weight* value that indicates how much of the similarity value of the source node will propagate to the target node in each iteration. The *weight* value is determined by the number of outgoing same *label* edges from the *source* node, and whether the different target nodes are treated equally or weighted. Here we treat all targets equally, so the weight for each outgoing same label edge is 1.0 divided by the number of these edges. Several fixpoint formulas were explored in [12]; in the basic one, after each iteration, the map pair node similarity $sim^{n+1} = Normalize(sim^n + \Sigma weight \times sim_{source})$ for each incoming edge). Propagation continues until the Euclidean length of the residual vector $\Delta(sim^{n+1}, sim^n)$ is less than threshold. We made the following modifications: (1) "good" map pair nodes propagate their similarity to neighbors, but neighbors don't propagate to them, which means that the propagation formulae are applied only to "non-good" map pair nodes; and (2) "good" matching pair nodes don't join the normalization process.

In Figure 4 (b), the final similarity values (*sim1, sim2*) are shown around each map pair node, where *sim1* is from the original algorithm without separating shared elements in the input graph, and *sim2* is from the modified algorithm that fixes high similarity initial mappings and uses paths instead of elements to make shared elements context dependent. The second gives higher similarity values for correct matches and lower similarity values for most wrong matches, although it doesn't differentiate `([bib/book, arts/book], 0.88)` from `([bib/book, arts/article], 0.88)`, the later combination of linguistic and structural similarity makes `(bib/book, arts/book)` win significantly. The first method doesn't differentiate `([year, title], 0.42)` from `([year, date], 0.42)`, since they

have the same similarity value, and it gives too low similarity to `([bib, arts], 0.29)`; both cases cause difficulty on filtering.

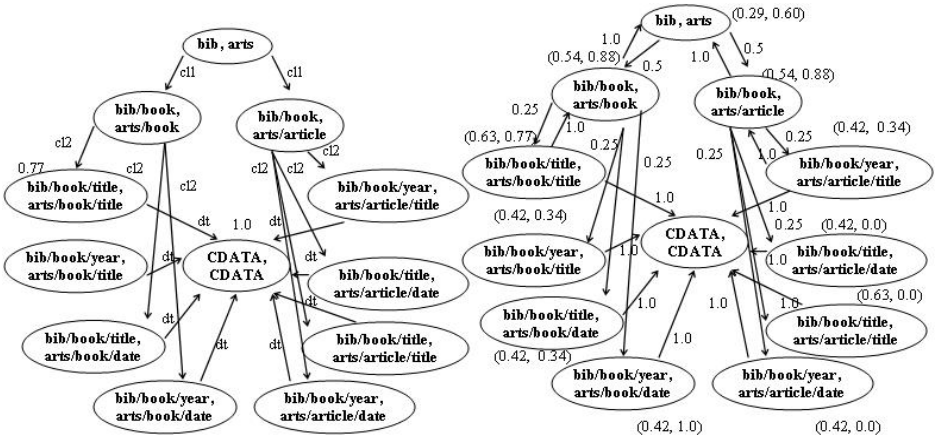

Fig. 4. (a) Pairwise connectivity graph and (b) Induced propagation graph

After this graph matching step, the similarity between two tree nodes is the weighted sum of similarity values of name, path, data type and structure:

$$s_{i,j} = ns_{i,j} \times w_{n\_c} + ps_{i,j} \times w_{p\_c+} + dts_{i,j} \times w_{dt\_c} + gs_{i,j} \times w_{g\_c} \quad (*)$$

where w_{n_c}, w_{p_c}, w_{dt_c}, and $w_{g_c}$are the weight values for combining the name, path, data type and structural similarities to identify the best match candidates for each node right before the following context check step, and $w_{n\_c} + w_{p\_c} + w_{dt\_c} + w_{g\_c} = 1.0$.

3.5 Context check

Matches are refined by checking and re-matching subtrees rooted at core elements, as discussed in (3d) above. After core contexts are identified, they are sorted in ascending order of height from the root, for top-down matching refinement. For each node n_i, check if its best matching node has maximum similarity bigger than a relatively high similarity threshold. If not, for this important target node, there may be no good match from the combined matching steps, so that its subtree matches also might not be good, and user interaction is requested; or in automatic mode, some matches with very low similarity within the subtree are thrown away and child nodes that are core nodes are checked further. If a good match is found, refinement of subtree matches to best matching node n_j's subtree starts by computing initial mappings, $InitMap_{subtree\_ni,subtree\_nj}$ which are the best so far within these two subtrees (but not the best at whole trees instead) and with similarity bigger than a threshold. Then subtree graphs $G_{subtree\_ni}$ and $G_{subtree\_nj}$ are generated as input with $InitMap_{subtree\_ni,subtree\_nj}$ together for $GraphMatch$ to compute again the structural similarities, $gs_{h,k}$ of nodes n_h in n_i's subtree and nodes n_k in n_j's subtree, by

$gs_{h,k} = GraphMatch(G_{subtree\_ni}, G_{subtree\_nj}, InitMap_{subtree\_ni,subtree\_nj})$

More checking is needed to see whether any almost equally good multiple matches with very high similarity values from the linguistic and data type matching step for these core nodes are still competing candidates after context check. If they are very different, this might indicate that some multiple matches would be missed. User input on specifying whether or not they are multiple matches can avoid missing multiple good matches. Then the core node subtree is matched to the multiple matching subtrees as discussed above, and a new similarity value for n_h and n_k is computed using formula (*), but the weight values for similarities from name, path, data type and graph matchers may change. Then for each target node n_i, another set of matches (n_j, *sim*), where n_j is a node in the source tree, is computed as the matching result of the context check matcher.

Core nodes might not group at only one or two levels in a schema tree, and they might not be identified precisely by heuristics. User input on specifying some or all core nodes can help the system concentrate on checking important subtree matches. When it is hard for the system to decide good matches for a core node, user feedback can significantly improve whole subtree matches, and significantly reduce total user effort.

3.6 Combination of match results

After performing the above matching steps, each target node gets a set of candidates from combining the results from the four individual name, path, data type and structural matchers, and another set of candidates from the context check matcher. Selecting top matches from one set, or from combination of both sets, depends on how they differ from each other. If the context check results win significantly with comparison based on the best match similarity value, the context check results will be selected, otherwise the pre-context-check results are selected. If they are similar, user input is requested, or by default the top candidates from both sets are selected by sorting them together non-interactively.

The direction of matching also affects the quality of match results according to our experiments and prior work [3]. Users can choose selecting match results from a single direction or both directions based on their knowledge of the schemas. Both-direction means matching target to source and also matching source to target, if a correspondence (n_t, n_s) in the target to source mapping is selected only if target node n_t is one of top k (e.g., k = 2) almost equal candidates for source node n_s in the source to target mapping. Both-direction is the default strategy, since it performs better on the average.

User input can also help decide how many best match candidates should be selected (by default only the best one is selected), the threshold for selecting or throwing matches with lower similarity values for each target node, and what delta value for deciding multiple candidates with similar similarity values almost equally in order to detect real multiple matches for output, etc.

It will save significant post-match manual work by asking user input here to organize matches in the right format for further applications, and to identify

necessary operations if they could not be automatically recognized or have not been specified earlier, especially for n:m matches. Since without formal metadata, ontological information or content information, it is nearly always infeasible to recognize precise conversion functions for n:m matches, the effort on developing a high quality user interface to support directly specifying them conveniently seems more helpful than asking users for formal metadata or domain ontologies as hints to discover them. We have not yet provided a good interface for delivering user input conveniently, but this should be done soon.

4 Examples and results

Three application domains have been used to evaluate our approach: 1) three book DTDs obtained from the web, bibliography (*b*), arts (*a*), bookstore (*st*), and a mediated one (*bk*), have been tested in our data integration prototype system DDXMI, using manual matching [13]; 2) three bio-molecular DTDs, GAME (*g*), BSML (*bs*), BIOML (*bi*), which have been trimmed to remove some branches that are huge but irrelevant to sequence encoding, or were caused by unsophisticated design, plus a mediated DTD (*s*) for gene sequence encoding only; and 3) five XML Schemas for purchase orders, CIDX (*c*), Excel (*e*), Noris (*n*), Paragon (*p*), and Apertum (*ap*), used by COMA [3]. The characteristics of relevant schemas are shown in Table 1.

Domain	*Bibliography*				*Biology*				*Purchase Order*				
Schema	*b*	*st*	*a*	*bk*	*bs*	*g*	*bi*	*s*	*c*	*e*	*n*	*p*	*ap*
#Paths	13	7	21	13	69	124	32	30	40	54	65	80	145
#Nodes	11	7	15	13	58	66	21	30	40	35	46	74	80
Max Depth	4	4	5	5	10	7	8	4	4	4	4	6	5

Table 1. Characteristics of the tested schemas

A partial evaluation of the quality of our matching process can be obtained using the same measures used in [3, 12], derived from the information retrieval and data mining fields. However, as argued earlier in the paper, a metric that counts the total user effort would provide a much more appropriate evaluation, since our aim is to minimize this quantity, rather than to compete directly with systems that are less interactive. The manually determined real matches R for a match task are compared with the matches P returned by automatic matching process. The following are counted: the correctly identified matches T, the wrong matches $F = P - T$, and the missed matches $M = R - T$. These correspond to true positives, false positives, and false negatives respectively. Then the following three measures are computed:

$Precision = T/P = T/(T + F)$

$Recall = T/R$

$Overall$ = 1-$(F + M)/R = (T - F)/R = Recall \times (2 - 1/Precision)$

The first gives the reliability of the mapping, the second gives the share of real matches found, and the third is a combined measure for mapping quality, taking account of the post-match effort needed for both removing wrong and adding missed matches.

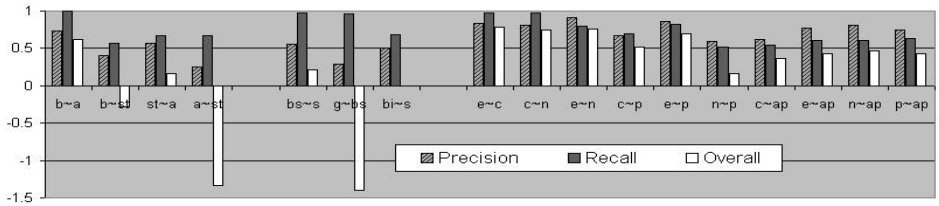

Fig. 5. Measure values: no context check, non-interactive mode, 1-direction

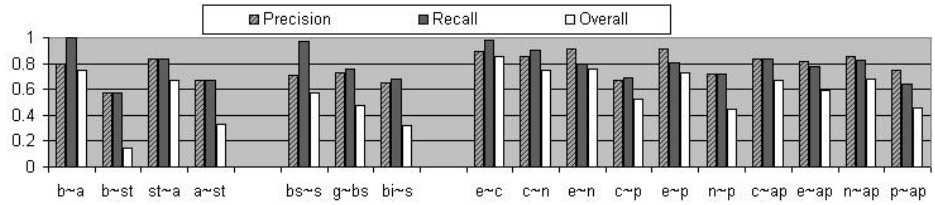

Fig. 6. Measure values: with context check, non-interactive mode, 1-direction

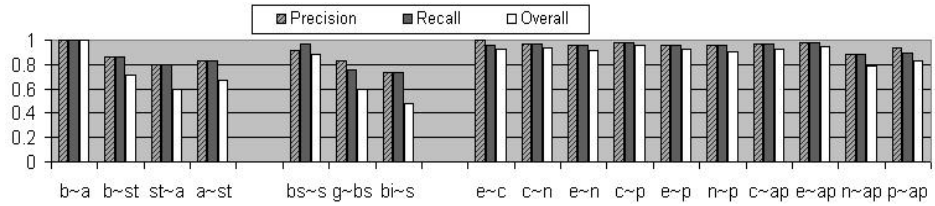

Fig. 7. Measure values: with context check, interactive mode, 1-direction

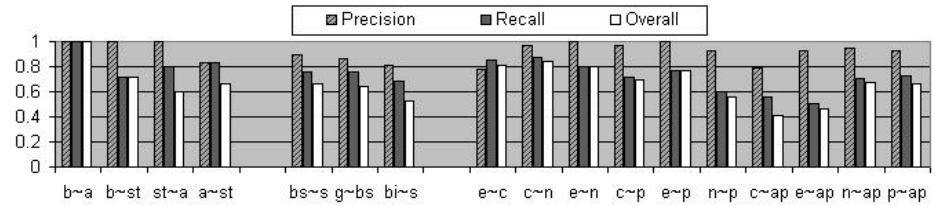

Fig. 8. Measure values: with context check, non-interactive mode, 2-direction

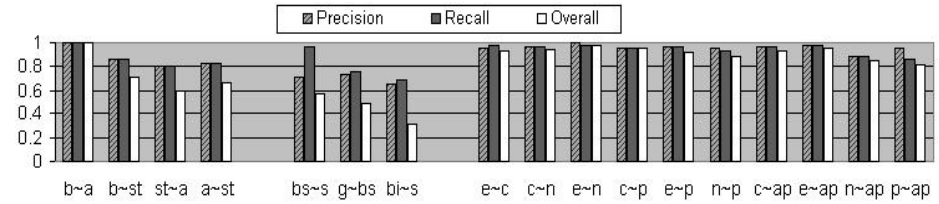

Fig. 9. Measure values: no context check, interactive mode, 2-direction

5 Summary and future work

The test results in Figures 5–9 show that our approach works well, in comparison with the purchase order schema tests of COMA [3] and Cupid [5]. The context check step seems especially helpful in 1-direction match, improving the overall score significantly. For example, in the task of matching target **arts** dtd with source **bookstore** dtd ($a{\sim}st$ in Figure 5–6) in only one direction (from the target to the source, trying to match target nodes as much as possible), **arts** is about both book and article, context check help prune all mismatches for the nodes under article subtree. But without context check, `arts/article/author`, `arts/article/title` get matched to `bookstore/book/author`, `bookstore/book/title` respectively, with relatively high similarity values. For simpler tasks with few multiple-to-multiple matches, in non-interactive mode, both-direction works as well as context check. In the above example, `arts/article/author` is not almost equal candidate as `arts/book/author` to `bookstore/book/author` in **bookstore** to **arts** mapping, so it and the others in article subtree are filtered out. User interactions help identify n:m matches with semantic functions where good 1:1 matches can not be determined, and certify many matches with relatively low similarity values, during matching process. Some of the overall measures are still very low because the difference of price units for book DTDs can't be detected without analyzing the data content or having more semantic metadata, or because necessary conditions can't be found, or even determined to be needed for matching, using only the available information.

Many issues remain to be studied. One is representing the generated mappings to facilitate user editing, for correcting false matches, adding missed matches, attaching restructuring and semantic conversion functions, making sure that all heterogeneities are resolved for the next step, query discovery, which is extremely important for data integration. Second, semantic metadata could make semantic matching more convenient and reliable, for example, to help with finding n:m matches; we are now studying this problem for ecological data integration and analysis problems. A third issue is using formal ontologies to help with schema mapping, assuming an ontology is attached to each schema; we are exploring ontology mappings and developing support techniques [14]. Match composition and reuse also deserve further research for the above issues, and have been touched upon for the 1:1 case by [3, 8]. Finally, we wish to conduct experiments with groups of users, to validate our proposed total user effort metric, and evaluate our system in its interactive mode.

Acknowledgements We thank Bertram Ludäscher for many valuable discussions, and we also thank Hong-Hai Do, Erhard Rahm and Sergey Melnik for generously sharing their data with us. This material is based upon work supported by the National Science Foundation under Grant No. ITR 0225676.

References

1. Bergamaschi, S., S. Castano, and M. Vincini: Semantic Integration of Semistructured and Structured Data Sources. SIGMOD Record **28(1)** (1999) 54–59.

2. Doan. A, P. Domingos, and A. Y. Halevy: Reconciling schemas of disparate data sources: A machine-learning approach. SIGMOD'01 (2001).
3. Do, H. and E. Rahm: Coma - A System for flexible combination of schema matching approaches. Proc. 28th Conf. on Very Large Databases (VLDB) (2002).
4. Doan, A. Thesis. http://anhai.cs.uiuc.edu/home/thesis/anhai-thesis.pdf (2003).
5. Madhavan, J., P. A. Bernstein, E. Rahm: Generic Schema Matching with Cupid. Proc. 27th Int. Conf. on Very Large Data Bases (VLDB) (2001).
6. Embley, D.W. et al: Multifaceted Exploitation of Metadata for attribute Match Discovery in Information Integration. WIIW (2001).
7. Li, W., C. Clifton: SemInt: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural network. Data and Knowledge Engineering **33:1** (2000) 49–84.
8. Madhavan, J., P. A. Bernstein, P. Domingos, & A.Y. Halevy. Representing and Reasoning About Mappings between Domain Models. In *18th National Conference on Artificial Intelligence (AAAI)*, 2002.
9. Melnik, S., E. Rahm, P. A. Bernstein: Rondo: A Programming Platform for Generic Model Management. Proc. ACM Intl. Conference on Management of Data (SIGMOD) (2003) San Diego, CA.
10. Milo, T. and S. Zohar: Using Schema Matching to Simplify Heterogeneous Data Translation. VLDB (1998) 122–133.
11. Miller, R., L. Haas, and M.A. Hernandez: Schema Mapping as Query Discovery. VLDB (2000) 77–88.
12. Melnik, S., H. Garcia-Molina, E. Rahm: Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. ICDE (2002).
13. Nam, Y. K, J. Goguen, and G. Wang: A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases. Proc. Intl. Conference on Ontologies, Databases, and Applications of Semantics for Large Scale Information Systems, Springer, Lecture Notes in Computer Science, **2519** (2002) 1332–1344.
14. Noy, N. F. and M. A. Musen: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. Proc. 17th National Conference on Artificial Intelligence (AAAI-2000), Austin, TX. Available as SMI technical report SMI-2000-0831 (2000).
15. Rahm, E. and Bernstein, P. A.: On Matching Schemas Automatically. Techn. Report (2001) Dept. of Comp. Science, Univ. of Leipzig. http://dol.uni-leipzig.de/pub/2001-5/en
16. Biskup, J. and D. W. Embley: Extracting information from heterogeneous information sources using ontologically specified target view. Information Systems **28** (2003) 169–212.
17. Li Xu and D. Embley: Using Domain Ontologies to Discover Direct and Indirect Matches for Schema Elements. Proc. Semantic Integration Workshop (2003) Sanibel Island, Florida.
18. Do, H.H., Melnik, S., Rahm, E.: Comparison of Schema Matching Evaluations. Proc. GI-Workshop "Web and Databases", Erfurt, LNCS 2593, Springer-Verlag (2002).
19. He, B. and K. C. Chang: Statistical Schema Matching across Web Query Interfaces. In ACM Intl. Conference on Management of Data (SIGMOD) (2003) San Diego, CA.