A SURVEY ON IMPLEMENTATION OF AUTOMATED AND INTEGRATED DATA SOURCES IN KEY WORD BASED WEB INFORMATION RETRIEVAL

P. SUNIL KUMAR REDDY, K.SUNEETHA

Research Scholar, Dept. of Comp. Science, S.V. University, Tirupathi, Andhra Pradesh- INDIA
Sr. Lecturer, Dept. of M.C.A, SVEC, Tirupathi, Andhra Pradesh- INDIA
pg.sunilkumar@gmail.com, umasuni.k@gmail.com

ABSTRACT

Rapid growth in data integration result in automation of Experimental data that can be revised interlinked and analyzed from different perspectives. From the users point of view retrieving relevant information is a challenging task. In this survey, we present innovative techniques for creating integrated views over data sources using keyword search techniques, ranked answers, and user feedback [32] to investigate how to automatically discover when a new data source has content relevant to a user’s view, performing automatic data integration for incoming data sets which results in incorporating innumerous methods to discover related attributes, including propagation algorithms from the machine learning community [2] and existing schema matchers [11]. The user may provide feedback on the suggested new results, helping the system to increase the cost of including a new source that is not relevant. In this survey, we discuss how data sources can be adapted to more relevant information with global schema.

Keywords: Relevance feedback, user feedback, schema matching, schema alignment, keyword search, data integration. Query system.

1. INTRODUCTION

Data integration is one of the most difficult challenges in information technology, largely due to the ambiguities involved in trying to semantically merge different data sources. In an ideal world, the data needs of science, medicine, and policy would be met by discovering new data sets and databases the moment they are published, and automatically conveying their contents to users with related information needs, in the form relevant to those users. In order to address the difficulties involved in retrieving relevant information, we need to define a global schema (mediated schema) for their field so that individual sources can be mapped into a common representation. There has been extreme growth in automating the mapping between alignment and retrieval of relevant tasks [29]. It is difficult to develop a global schema that can captures the diverse needs of a large user base and discover new concepts, methods, and types of experimental result. There are only few mechanisms exist that discover relevant data sources which have their data automatically to be in use. Schema alignment tools exist which can map to large numbers of schemas and relations, and it can be difficult to determine when they have produced the right mappings.

Query system [32] develops an information need driven paradigm for data integration, which addresses the above problems. Query system comprises a set of databases that contain known cross-references, links, and correspondence or cross-reference tables which does not require a global mediated schema or full schema mappings. A user specifies an information need through a keyword query. Keyword search in databases [4, 5, 17, 20] define a ranked view consisting of a union of conjunctive queries over different combinations of the data sources which are defined through relevant feedback techniques. In this survey, Query’s information need-driven integration is discussed which can address automatically appending new data sources and relating them to the existing ones. When user registers a new database, then relevance between source and ranked information is considered which uses information about overlapping of data and data alignment costs from existing schema matchers.

Query system can be extended to combine and integrate weighted outputs from different schema matcher [32]. If the source is most relevant to a ranked view, then query results are considered to be appropriate. When the system receives feedback about irrelevant results, then cost incurred to map relevant document is adjusted in order to avoid errors. Query system can adjust weights for individual alignments, including how much to favor the outputs from different schema matchers. This system results in interactive, user-driven integration (e.g., data spaces [14] and best-effort integration [30]) by automatically discovering semantic links among data sources, and using a data-driven approach to providing feedback to the system. Any form of automatic schema alignment can result in errors, which needs to determine all possible errors. When we analyze the data from the perspective of a particular information need, we need to (1) Incorporate efficient methods in order to ensure the quality of results, (2) identify irrelevant results. Below architecture allows creating alternative alignments, and align tables against the new source that affects top-k query results. This system is a unified representation for data values and attributes that facilitate ranked querying and learning [11]. Alignment components from
database [11] and machine learning [33] literature show how to combine the output of alignment components. Algorithm namely Modified Adsorption (MAD)[31] detects schema alignments, and studies its effectiveness in combination with, the COMA++ tool [11]. Machine learning algorithm called MIRA [10] is applied to learn attribute alignments and combine information from multiple matching tools. All these techniques are based on feedback over answers.

2. DATA INTEGRATION THROUGH SEARCH BASED TECHNIQUES

In the Search based techniques. Keyword search query model [4, 17, 20, 32] is adapted in which keywords are matched against elements in one or more relations in different data sources. This system finds links between the relations matching the given keywords and these links are identified by different kinds of associations like foreign key relationships, value overlaps or global identifiers, similarity predicates, or hyperlinks. Multiple relations that match a search keyword may exist and multiple attribute pairs may align between relations, suggesting many possible ways to join relations in order to answer the query. The Query system starts with an initial search graph generated from existing data source relations and the associations among them. During the view creation and output stage, a keyword search is posed against this search graph, and results in a top-k view containing answer relevant to the user. The definition and contents of this view are maintained continuously and the top-scoring queries and their results may need to be updated in response to changes to the underlying search graph developed by the user, and the user may provide feedback that changes the costs of certain queries and when new data sources are discovered by the system, and their attributes are aligned with the existing relations in the search graph, that results in new top-k answers for the user’s view. The process of updating the schema graph’s nodes and associations is referred as search graph maintenance and there is interplay between the two graph maintenance mechanism and the view creation and output stage, as the system may propose an alignment, the view’s contents may be updated, the user may provide feedback on these results, and the view output may be updated. These focused around alignments that are relevant to the user’s ongoing information need.

2.1 Constructing Initial Search Graph

Before processing any query, an initial search graph is created to represent the relations and potential join links. Query system initially scans the metadata in each data source, and determines all attribute and relation names, foreign keys, external links, common identifiers, and other auxiliary information. The basic search graph comprises two types of nodes: relations, represented by rounded rectangles, and attributes, represented by ellipses. The graph is extended with bidirectional association edges drawn from the results of schema matching tools. Each tuple in each of the tables is a virtual node of the search graph, linked by zero-cost edges to its attribute nodes.

2.2 Constructing Views from Keyword Queries

Given a keyword query \( Q = \{K_1, \ldots, K_m\} \), we dynamically expand the search graph into a query graph as follows. For each \( K_i \in Q \), a keyword similarity metric is used to match the keyword against all schema elements and all pre-indexed data values in the data sources. A node representing \( K_i \) to the graph and an edge from \( K_i \) to each graph node matching it are added and Each edge is assigned a set of costs, including mismatch cost that is lower for closer matches, and costs related to the relevance of the relations connected by the edge. The edge also contains an adjustable weight that appropriately scales the edge cost to yield an overall edge cost. For each database tuple matching the keyword, a node is added for each value in the tuple, with a similarity edge between the value and the \( K_i \) node. Zero-cost edges are added between tuple value nodes and their corresponding attribute nodes.

From this query graph, each tree with leaf nodes \( K_1..K_m \) represents a possible join query. From each Query tree, a conjunctive SQL query is generated that constructs a list of items for the SQL select, from, and where clauses, and an associated cost expression for the particular query value-based similarity predicates are incorporated to match keywords to data or metadata, not in joining one item with another; hence the cost of each query is independent of the tuples being processed. The individual SQL statements must be combined together to increase order of associated which needs outer join and each query may output different attributes, and we need a single unified table for output and place conceptually “compatible” output attributes from different queries into the same column.

We start by defining the query output schema QA to match the output schema of the first query’s select-list LA. Then, for each successive query, we iterate over each attribute a in its select-list. Let \( na \) be the node in the query graph with label a. Suppose there exists some similarity edge \( (na; na1) \) with cost below a threshold t, and label \( (na1) \) appears in QA. If the current query is not already outputting an attribute corresponding to label \( (na1) \), then we rename attribute a to label \( (na1) \) in the output. Otherwise, we simply add a as a new attribute to QA. Then we create a multiway disjoint union SQL query, in which each “branch” represents one of the queries produced from a query tree. Each “branch” also outputs a cost (its e term). Finally, we execute the queries and return answers in ranked order, annotated with novel information about their originating queries.
2.3 Maintenance of Search Graph
The novel aspect of Query system is its ability to maintain the search graph and adjust the results of existing user queries accordingly. We assume that a user’s query has described an ongoing information need for that user, and that he or she will make future as well as current use of the query results. Hence we save the results of the query as a view, and we focus on enabling the user to refine the view by giving feedback and adjusting the weights given to various associations, and on incorporating new data sources if good associations can be found with the existing relations in the search graph, and the contents of these new sources affect the contents of the top-k tuples in the user’s view.

3. APPENDING DATA SOURCES
Once a keyword search-based view is defined, Query system switches into search graph maintenance mode. maintenance process, decides how to incorporate new sources into the current view as the system is notified of their availability and comprises a registration service for new tables and data sources and this mechanism can be manually activated by the user, or could ultimately be triggered directly by a Web crawler that looks for and extracts tables from the Web [7] or the deep Web [24, 35].

3.1 Approach
When a new source is registered, the first step is to incorporate each of its underlying tables into the search graph. The search graph is in effect the data model queried by Query system and it contains both metadata (relation and attribute nodes) and data (tuple values), related by edges that specify possible ways of constructing a query.

The lower the cost of an edge, the more likely that the edge will be relevant to answering queries involving one of the nodes it links. When a new source is encountered, the first step is to determine potential alignments between the new source’s attributes and those in existing tables: these alignments will suggest potential joins to be used in query answering, and potential alignments of attributes in query output, such that the same column in the query answers contains results from different sources. Aligned attributes may arise from the same domains or different domains. This task requires a set of alignment primitives (schema matching algorithms) used to match attributes. As the search graph grows in size, the cost of adding new associations becomes expensive: regardless of the specific primitives used, the cost of alignment tends to be at least quadratic in the number of compatible attributes. We must find ways of reducing the space of possible alignements considered.

We develop an information need-driven strategy where we consider only alignments that have the potential to affect existing user queries and then develop techniques for correcting bad alignments through user feedback on the results of their queries.

3.2 Alignment Primitives
Goal of Query system is to develop an architecture and learning methods that are agnostic as to the specific schema matching or attribute alignment techniques used, so that we can benefit from existing methods in databases and machine learning. To demonstrate the architecture’s ability to accommodate different schema matching algorithms, we incorporate two complementary types of matchers in Query system. The first type consists of typical similarity-based schema matchers from the database community that rely on pair wise matches between source and target relations,

Primarily looking at schema rather than instance-level features. The second kind is matchers that globally aggregate the compatibilities between data instances. To that end, we develop a new schema matching technique that looks at “type compatibility” in a way that considers transitivity: if attribute A has 50% overlap in values with attribute B, and attribute B has 50% overlap in values with source C, all three attributes likely come from the same domain even if A and C do not share many values. Here we adapt a technique from the machine learning and Web community called label propagation that exploits transitivity and data properties, which has not previously been applied to schema matching.

3.2.1 Metadata Matcher Alignment
As the name specifies, Meta data Matcher matches multiple data items when determining alignments and COMA++ is used for meta data matching through API [11]. It contains structural relationships and substring matchers to produce proper alignments.

3.2.2 Alignment with Label Propagation
This focuses on which attributes are type compatible at the instance level. This is used in recent machine learning work for finding associated metadata based on weighted transitive relationships across many sources. Informally, this work represents a generalization of some of the ideas in similarity flooding [26] or the Cupid algorithm[23].This contains graph G = (V;E;W) with nodes V , directed edges E, and a weight function W : E -> R that assigns a weight to each edge. Labels are propagated from each node along its out-edges to its neighboring nodes with a probability proportional to edge weight, eventually yielding a label probability distribution Li for each node. this model is similar to Page Rank [6].In this work, we use the Modified Adsorption (MAD)[31] label propagation algorithm. MAD is used in several areas of research [36].
Let $G_r = (V, E_r, W_r)$ be the edge-reversed version of the original graph $G = (V, E, W)$, where $(a; b) \in E_r$ iff $(b; a) \in E$, and $W_r(a; b) = W(b, a)$. Now, choose a node of interest $q \in V$. To estimate $L_q$ for $q \in V$, we perform a random walk on $G_r$ starting from $q$ to generate samples for a random label variable $L$. After reaching a node $i$ during the walk, the following steps are performed.

1. With probability $p_{cont}$, continue the random walk to a neighbor of $i$.
2. With probability $p_{aband}$, abandon the random walk. This abandonment probability makes the random walk stay relatively close to its source when the graph has high-degree nodes. When the random walk passes through such a node, then that further transitions are unrelated to the source and they are abandoned.
3. With probability $p_{pinj}$, stop the random walk and emit either $L_i$ if $i$ is one of the initially labeled nodes.

$L_q$ will converge to the distribution over labels $L$ emitted from random walks initiated from node $q$. In the below proposed algorithm, $I_v$ is the injected label distribution that a node is seeded with; $R_v$ is a label distribution with a single peak corresponding to a separate “none of the above” label $T$. This dummy label allows the algorithm to give low probability to all labels at a node if the evidence is irrelevant.

### Algorithm 1 Modified Adsorption (MAD) Algorithm

**Input:** Graph: $G = (V, E, W)$, Seed labeling: $I_v \in \mathbb{R}^{n+1}$ for $v \in V$. Probabilities: $p_{cont}^v, p_{aband}^v, p_{pinj}$ for $v \in V$, Label priors: $R_v \in \mathbb{R}^{n+1}$ for $v \in V$. Output: Label Scores: $L_v$ for $v \in V$

1. $L_v \leftarrow I_v$ for $v \in V$ [Initialization]
2. $M_v \leftarrow \mu_1 \times p_{cont}^v + \mu_2 \times p_{aband}^v + \sum_w W_{vw} + \mu_3$
3. repeat
4. $D_v \leftarrow \sum_i (p_{cont}^i \times W_{vi} + p_{aband}^i \times W_{vi}) \times I_i$
5. for all $v \in V$ do
6. $L_v \leftarrow \frac{1}{M_v} \times (\mu_1 \times p_{cont}^v \times I_v + \mu_2 \times D_v + \mu_3 \times p_{aband} \times R_v)$
7. end for
8. until convergence

### 3.2.3 Integrating Matchers in Query System

**Matching query results with COMA++**

COMA++ matcher typically performs pair wise schema matching where each new source attribute gets aligned with only a single attribute in the existing set of data sources. matchers only output their top alignment, even when other potential alignments are considered. Goal of Query system is to determine the top-$Y$ (where $Y$ is typically 2 or 3) candidate alignments for each attribute, unless the top alignment has very high confidence: this way we can later use user feedback to “suppress” a bad alignment and see the results of an alternative. In order to receive alignments between the new source’s attributes and all sources, a pair wise schema alignment between the new source and each existing source is done and we obtain what COMA++ assumes to be the top attribute alignments between each relation pair. Between each pair of schemas, we can first compute the top alignment. Next, for each alignment pair $(A, B)$ that does not have a high confidence level, remove attribute $A$ and re-run the alignment, determining the “next best” alignment . Next re-insert $A$ and remove $B$, and repeat the process. If there are additional schema matching constraints (e.g., no two source attributes may map to the same target attribute), we can again iterate over each alignment pair $(A, B)$. Now remove all attributes from $A$’s schema that are “type compatible” with $A$, except for $A$ itself; and run the alignment. Then replace those attributes, and repeat the process removing attributes type-compatible with $B$ other than $B$ itself. Ultimately, we will have obtained from the matcher a set of associations (equivalent here to the alignments) and their confidence levels. Depending on the matcher used, the confidence scores may need to be normalized to a value between 0 and 1; in the case of COMA++, its output already falls within this range. These confidence scores will be used in forming a new edge cost.

**Using MAD to identify Compatible Data types**

Matcher module identifies data types across schemas, using techniques described in [31]. Even though matcher implementation is a part of Query system, it does not provide any special interfaces. From Query’s perspective it remains a black box. This matcher first creates an internal label propagation graph that incorporates both metadata and data. From the search graph, we take all relation attributes from all sources, and create a node in the label propagation graph for each attribute, labeled with its canonical name. We also consider all data values and create a label propagation graph node for each unique value. We add to the graph an edge between a value node and each node representing an attribute in which the value appears.
Now we annotate or label each attribute node with its name. We run the MAD algorithm, propagating sets of annotations from node to node. The algorithm runs until the label distribution on each node ceases to change beyond some tolerance value. Alternatively, the algorithm can be run for a fixed number of iterations. Each value node ultimately receives a distribution describing how strongly it “belongs” to a given schema attribute, and each attribute node receives a distribution describing how closely it matches other attribute nodes. MAD does not require direct pair wise comparison of sources which involves multiple sources. We use the label distributions generated by MAD to generate uncertainty levels from which edge costs will be derived for Query’s search graph. For each node \( n \) in the MAD graph, we select the top-\( Y \) attributes from its label distribution, and we add an edge in the search graph between the attribute node for \( I \) and the attribute node for \( n \). The confidence level for each such edge will be \( \ln (I) \).

### 3.3 Identification of Associations between Data Sources

When the number of data sources increase, we need to ensure that alignment algorithms are relevant and the simple approach is to perform exhaustive matching and once new data source is identified, we iterate over all existing data sources in turn, and run our alignment algorithm(s) and this approach is called Exhaustive approach. Identifying associations between data sources is a tedious task. New associations are “visible” to users if they appear in any queries returning top-k results. Hence existing scores of top-k results, restrict the search space of alignments. As new queries are materialized within the system, we incrementally consider further alignments that might affect the results of those queries.

We can implement techniques from network formation in social networks [3], and assume the existence of an alignment prior \( (P) \) over vertices of the existing search graph \( G \), specifying a preference ordering for associations with the existing nodes. This can capture alignments with highly authoritative or popular relations. When we add a new association edge based on an alignment, we set its cost based on the following weighted features:

1. A default feature shared with all edges and set to 1, whose weight comprises a default cost added to all edges.
2. A feature for the confidence value of each schema matcher, whose weight represents how heavily we (dis)favor the schema matcher’s confidence scores relative to the other cost components.
3. A feature for each relation \( R \) connected by the association, whose value is 1 for this relation \( R \), and whose weight represents the negated logarithm of the \( R \)’s authoritativeness.
4. A feature that uniquely identifies the edge itself, whose value is 1, and whose weight comprises a cost added to the edge.

### 4. EVALUATIONS BASED ON USERS FEEDBACK

When the user views a set of results, he or she may notice a few results that seem either clearly correct or incorrect. In Query system, the user may provide feedback by optionally annotating each query answer to indicate a valid result, invalid result, or a ranking constraint (tuple \( tx \) should be scored higher than \( ty \)). Query system first generalizes this feedback by taking each tuple, and, by looking at its provenance, replacing it with the query tree that produced it, using a scheme similar to [32] and yields that our model lower cost which results in higher ranking. The association cost learner converts each tuple annotation into a constraint as follows:

1. A query that produces correct results is constrained to have a cost at least as low as the top-ranked query result. These constraints are fed into an algorithm called MIRA [10], which is effective in learning edge costs from user feedback on query results [32].

**Relationship between Edge Costs and Features.** Each edge is initialized with a cost composed of multiple weighted features: the product of the weight and the feature value comprise a default cost given to every edge, a weighted confidence score from each schema alignment algorithm, the authoritativeness of the two relations connected by the edge, and an additional cost for the edge itself. Query’s association cost learner takes the constraints from user feedback and determines a weight assignment for each feature — thus assigning a cost to every edge.

**Learning Algorithm.** The learning algorithm reads examples sequentially and updates its weights after receiving each of the examples based on how well the example is classified by the current weight vector. The algorithm, which was first used in [32], is a variant of the Margin Infused Ranking Algorithm (MIRA) [10]. We need to include real-valued features from similarity costs. Using real-valued features directly in the algorithm results in poor learning because of the different ranges of different real-valued and binary features. Therefore the real-valued features are then replaced by features indicating membership. The weights are all zero. After receiving feedback from the user on the \( r \)th query \( Sr \) about a top answer, the algorithm retrieves the list \( B \) of the \( k \) lowest-cost Steiner trees using the current weights. The user feedback for interaction \( r \) is represented by the keyword nodes \( Sr \) and the target tree \( Tr \) that yielded the query answers most favored by the user. The
algorithm then updates the weights so that the cost of each tree T ∈ B is worse than the target tree T_r by a margin equal to the mismatch or loss L(T_r; T) between the trees. If T_r ∈ B, because L(T_r; T_r) = 0, the corresponding constraint in the weight update is trivially satisfied. The update also requires that the cost of each edge be positive, since non-positive edge costs will result in non-meaningful Steiner tree computations.

5. INCORPORATING NEW DATA SOURCES
When a huge data set is supplied to the Query system. Query logs s for pairs of SQL queries are scanned, where one query represented an expansion of the other, base, and query: i.e., the expanded query either joined or unjoined additional relationships with the base query in order to identify new data sources. The expanded query identifies new sources that would be useful to add to an existing search graph that had been capable of answering the base query.

When the expanded query represents the union of the base query with a new query sub expression, then clearly the new data source results in new association edges that provide further data for the user’s view. When the expanded query represents an additional join of the base query with new data, this also affects the contents of the existing view if the additional join represents a segment of a new top-scoring Steiner tree for the same keyword query.

For each base query, e constructed a corresponding keyword query is constructed, whose Steiner trees included the relations in the base query. Next, the search graph can be initialized to include all sources except the ones unique to the expanded query. Weights in the search graph can be initialized to default values, then feedback can be provided on the keyword query results, such that the SQL base query from the query log is returned as the top query.

5.1 Evaluating Matchings
Query matchings can be evaluated not only on the cost of running alignment but also on at their quality. Query system considers the suggested alignments from the individual alignment algorithms, as well as user feedback on query answers, to get the correct associations. These experiments were conducted over the from the original schema specifications, we identify that there are 8 semantically meaningful join or alignment edges among these relations, but we remove this information from the metadata. This process starts with a schema graph that contains the tables and then to run the association generation step (using COMA++ and/or MAD) to generate a new graph in the Y most promising alignments (for different values of Y) are recorded for each attribute. Next we execute the set of keyword queries obtained from the databases’ documentation. For each query, we generate one feedback response, marking one answer that only makes use of edges in the gold standard. Since the gold standard alignments are known during evaluation, this feedback response step can be simulated on behalf of a user. Our goal is to “recover” all of the links which form gold standard. Metrics are computed with respect to the search graph, as opposed to looking at query answers. For different values of Y, we compare the top Y alignment edges in the search graph (that also fall under a cost threshold) for each attribute, versus the edges in the gold standard. Clearly, if the alignment edges in the schema graph exactly match the gold standard, then they will result in correct answers.

5.2 Migrating to Large number of Data Sources
Query system can be migrated to volatile Data Sources and the cost of incorporating the keyword queries on the same can be estimated and performance can be evaluated. Query Optimization plays a vital role in estimating the cost of the queries and this can be done by creation of indexes and ranking the queries based on te cost. Once the Top queries on basis of cost of performance are identified, then each keyword query is executed in sequence and ensures that base query is the top scorer. Costs of edges of the query are calibrated to provide relevant and meaningful results. After this, new Data sources are generated with two attributes, and then they are connected to random nodes in the Search graph of Query System. Once Schema graph of Query is constructed, alignment methods can be used to align the new data sources created in the graph. As the query graph do not contain related attributes, COMA+ tool can be used to run and it focuses on the column comparisons of the queries used in Query system . Number of pair wise comparisons do not change even though the number of Data sources are more.

5.2.1 COMA++ as matcher
COMA++ can be used to match queries in Query system and perform pair wise comparisons which means that new source attribute is aligned with only a single attribute in existing set of data sources and matchers output only a single attribute in existing set of data sources and matchers only output their top alignment even when top alignments are considered and goal in Query system is to determine the top alignments which can be done by COMA++ tool also. Depending on the confidence and the cost, best query match based on the keyword queries is identified. Pair wise schema alignment between new data source and existing data source is identified and top attributes between each relation pair is obtained by COMA++. COMA++ can also be used to effectively perform Query optimization and Query processing in addition to cost estimation and schema alignments.
6. RELATED WORK
This survey discusses about Query system presented in [32], namely, that all alignments were specified in advance. Many systems supporting keyword search over databases [4, 5, 16, 17, 18, 20] use scores based on a combination of similarity between keywords and data values, length of join paths, and node authority [1]. Existing “top-k query answering” [9, 15, 22, 25] provides the highest-scoring answers for ranked queries. Schema alignment or matching is well-studied across the database, machine learning, and Semantic Web communities [29]. General consensus is that methods that incorporate both data- and metadata-based features, and potentially custom learners and constraints, are most effective. Thus, most modern matchers combine output from multiple sub-matchers [11, 12, 26]. Our focus is not on a new method for schema matching, but rather architecture for incorporating the output of a matcher in a complete iterative, end-to-end pipeline where the matches or alignments are incorporated into existing user views, and feedback on answers is used to correct schema matching output. Our approach requires no special support within the matcher, propagating “influences” across node connectivity for schema alignment is used in similarity flooding [26] and the Cupid system [23], among other schema matching studies. However, in the machine learning and Web communities, a great deal of work has been done to develop a principled family of label propagation algorithms [2, 36]. We incorporate this kind of matching method not only to align compatible attributes in the output, but to discover synonymous tables and transitivity related items. This survey builds upon recent observations [33] showing that one could find potential labeling of tables extracted from the Web using a particular label propagation algorithm called Modified Adsorption (MAD). Ranked data model propagates uncertainty from uncertain mappings to output results. Intuitively, this resembles the model of probabilistic schema mappings [13], although we do not use a probabilistic model. Our goal is to learn rankings based on answer feedback, and hence we need a ranking model amenable to this. Our work is complementary to efforts on learning to construct mashups [34], in suggesting potential joins with new sources. Goal here is to exploit user feedback to learn to correct schema matching errors. A method that learns to rank pairs of nodes based on their graph-walk similarity is presented in [27]. In contrast, the learning method used learns to rank trees derived from the query graph, and not just node pairs. The method for incorporating user feedback as presented in [8] requires developers to implement declarative user feedback rules. We do not require any such intermediate rule implementation, and instead learn directly from user feedback over answers.

CONCLUSIONS AND FUTURE WORK
This survey discusses about, automatic, information need driven strategy for automatically incorporating new sources and their information in a data integration setting. Queysysytem represents a step towards the ultimate goal of automated data integration, at least for particular kinds of datasets. This can be extended to automation of information about specific domains, including Web sources with information extraction components.

REFERENCES