Information Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Tuning the ensemble selection process of schema matchers

Avigdor Gal *, Tomer Sagi

Faculty of Industrial Engineering and Management, Technion - Israel Institute of Technology, Technion City, Haifa 32000, Israel

ARTICLE INFO

Article history:
Received 4 May 2009
Received in revised form
7 January 2010
Accepted 22 April 2010
Recommended by: J. Van den Bussche

Keywords:
Database integration
Schema matching

ABSTRACT

Schema matching is the task of providing correspondences between concepts describing the meaning of data in various heterogeneous, distributed data sources. It is recognized to be one of the basic operations required by the process of data and schema integration and its outcome serves in many tasks such as targeted content delivery and view integration. Schema matching research has been going on for more than 25 years now. An interesting research topic, that was largely left untouched involves the *automatic* selection of schema matchers to an ensemble, a set of schema matchers. To the best of our knowledge, none of the existing algorithmic solutions offer such a selection feature. In this paper we provide a thorough investigation of this research topic. We introduce a new heuristic, Schema Matcher Boosting (SMB). We show that SMB has the ability to choose among schema matchers and to tune their importance. As such, SMB introduces a new promise for schema matcher designers. Instead of trying to design a perfect schema matcher, a designer can instead focus on finding better than random schema matchers. For the effective utilization of SMB, we propose a complementary approach to the design of new schema matchers. We separate schema matchers into first-line and second-line matchers. First-line schema matchers were designed by-and-large as applications of existing works in other areas (e.g., machine learning and information retrieval) to schemata. Second-line schema matchers operate on the outcome of other schema matchers to improve their original outcome. SMB selects matcher pairs, where each pair contains a first-line matcher and a second-line matcher. We run a thorough set of experiments to analyze SMB ability to effectively choose schema matchers and show that SMB performs better than other, state-of-the-art ensemble matchers.

1. Introduction

Schema matching is the task of providing correspondences between concepts describing the meaning of data in various heterogeneous, distributed data sources (e.g., attributes in database schemata, tags in XML DTDs, fields in HTML forms, etc.) Schema matching is recognized to be one of the basic operations required by the process of data and schema integration [1–3], and thus has a great impact on its outcome. The outcome of the matching process can serve in tasks of targeted content delivery, view integration, database integration, query rewriting over heterogeneous sources, duplicate data elimination, and automatic streamlining of workflow activities that involve heterogeneous data sources. As such, schema matching has impact on numerous modern applications from various application areas. It impacts business, where company data sources continuously realign due to changing markets. It also impacts the way business and other information consumers seek information over the Web. It impacts life sciences, where scientific workflows cross system boundaries more often than not. Finally, it impacts the way communities of knowledge are created and evolve.

Schema matching research has been going on for more than 25 years now (see surveys [1,4–6] and online lists,

* Corresponding author. Tel.: +972 4 8294425; fax: +972 4 8253965.
E-mail address: avigal@ie.technion.ac.il (A. Gal).

0306-4379/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2010.04.003

e.g., OntologyMatching,[1] Ziegler,[2] DigiCULT,[3] and SWgr[4]) first as part of schema integration and then as a standalone research. Over the years, a significant body of work was devoted to the identification of *schema matchers*, heuristics for schema matching. Examples include COMA [7], Cupid [8], OntoBuilder [9], Autoplex [10], Similarity Flooding [11], Clio [12], Glue [13], and others [14–16]. Roughly speaking, schema matchers receive as input two or more schemata, compute a measure of similarity between attributes, and suggest as output a possible set of correspondences between attributes. We provide a full specification of the matching process in Section 3. The main objective of schema matchers is to provide schema matchings that will be effective from the user point of view yet computationally efficient (or at least not disastrously expensive). Such research has evolved in different research communities, including databases, information retrieval, information sciences, data semantics and the semantic Web, and others.

Choosing among schema matchers is far from being trivial. First, the number of schema matchers is continuously growing, and this diversity by itself complicates the choice of the most appropriate tool for a given application domain. Second, as one would expect, empirical analysis shows that there is no (and may never be) single dominant schema matcher that performs best, regardless of the data model and application domain [17]. Therefore, several tools, e.g., [7–9,18,19] have combined different schema matchers to determine the outcome of the matching process. An interesting research topic, that was largely left uncovered, involves the selection of schema matchers to an ensemble. Lee et al. suggested in [20] that the tuning of an ensemble involves the selection of "the right component to be executed." However, to the best of our knowledge, none of the existing algorithmic solutions offer an *automatic* selection feature. In this paper we provide a thorough investigation of this research topic.

Research has shown that many schema matchers perform better than random choice. In [17] we have introduced the monotonicity principle. We connect, in this work, this monotonicity property to the notion of weak classifiers in the AI literature [21]. The theory of weak classifiers has led to the introduction of a class of *Boosting* algorithms (e.g., [21]). This class of algorithms can strengthen weak classifiers to achieve arbitrarily high accuracy and has been shown to be effective in the construction of successful classifiers. Given a set of weak classifiers, the algorithm iterates over them while re-weighting the importance of elements in the training set. There exist many versions of boosting to-date. In this paper we build upon the AdaBoost algorithm [22] to introduce a new heuristic, Schema Matcher Boosting (SMB). We show that SMB has the ability to choose among schema matchers and to tune their importance. As such, SMB introduces a new promise for schema matcher designers. Instead of trying to design a perfect schema matcher, a designer can instead focus on finding better than random schema matchers.

For the effective utilization of SMB, we propose a complementary approach to the design of new schema matchers. We separate schema matchers into first-line and second-line matchers. First-line schema matchers were designed by-and-large as application of existing works in other areas (e.g., machine learning and information retrieval) to schemata. Second-line schema matchers operate on the outcome of other schema matchers to improve their original outcome. Existing examples of second-line matchers include similarity flooding [23], matching ensembles [7], and schema matching verification [24]. Although second-line schema matchers exist, we aim to show the benefit in this classification to the design of new schema matchers. Therefore, we generalize the notion of second-line matchers and discuss their properties.

We run a thorough set of experiments to analyze SMB ability to effectively choose schema matchers and show that SMB performs better than other, state-of-the-art ensemble matchers. The rest of the paper is organized as follows. Section 2 provides an overview of schema matching research efforts, focusing on selection and tuning of matchers. A model of the schema matching process, based on similarity matrices, is introduced in Section 3, followed by the introduction of second-line schema matchers (Section 4). Section 5 introduces SMB and our empirical analysis is given in Section 6. We conclude in Section 7.

2. Background and related work

For completeness sake, we provide first a brief overview of major achievements in schema matching modeling. We highlight the limitations of current works and identify the gaps to be filled by this work. The body of research on the topic of schema matching (and mapping) is vast and we do not attempt to cover all of it here.

Several attempts were made to classify schema matchers into broad classes [5,25,17,26]. In this work we propose another classification, separating first-line from second-line matchers, which adds another dimension on top of existing ones. We defer a comparison of our classification to existing ones to Section 4.

In 2004, Melnik and Bernstein offered a unique contribution to the understanding of the foundations of schema matching [3,27]. Their work proposes a conceptual framework of three layers, namely *applications*, *design patterns*, and *basic operators*. Of particular interest to our work is the *Match* basic operator, the operator that performs the schema matching. This operator precedes *mapping* operators, which represent the functional relationships between attributes. Schema mapping typically incorporates the data transformation semantics that is required to migrate data from a source schema to a target schema. Schema mapping is also researched heavily

[1] http://www.ontologymatching.org/
[2] http://www.ifi.unizh.ch/~pziegler/IntegrationProjects.html
[3] http://www.digicult.info/pages/resources.php?t=10
[4] http://www.semanticweb.gr/modules.php?name=News&file=categories&op=newindex&catid=17

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

[28–31]. Following [3,27], we shall separate the two research areas and focus on schema matching alone in this work.

Somewhat surprisingly, up until recently there was a little fundamental research that can lead to a theoretically rigorous generic infrastructure for further development of algorithmic solutions to the problem of schema matching. After more than 25 years of research, the research area of schema matching is still borrowing isolated bits and pieces of research from other areas, including information retrieval, machine learning, graph theory, and logic. Some of these efforts have been successful while others have proven to be useful in limited domains only. Inquiring about the roots of success and failure in schema matching will, in many cases, sum up to guesswork, simply because there is no single theoretical foundation in place [5,26]. Having a theoretical basis in place, one could start and design a set of algorithms to support the design of schema matching, enhancing user effectiveness. In this work we provide a small step towards such foundation, promoting the design of less-than-perfect matchers to which our boosting heuristic can be applied.

Model theory for attribute correspondences was investigated in [32,33,25]. In [32] correspondences were represented using morphisms in categories [34]. The work in [33] provides explicit semantics to matchings using logical models and model satisfaction. Ref. [25] provides a formal model of schema matching for topic hierarchies, rooted directed trees, where a node has a "meaning," generated using an ontology. A matching connects topic hierarchies by some relation (e.g., subsumption).

The limitations in this line of models, as well as some work in the semantic Web area (e.g., semantic schema matching [35]), were presented both in [33,25]. The former argues for the need "to incorporate *inaccurate* mappings and handle *uncertainty* about mappings. Inaccuracy arises because in many contexts there is no precise mapping…mappings may be inaccurate [since] the mappings language is too restricted to express more accurate mappings." Benerecetti et al. [25] went even further, arguing philosophically that even if two schemata fully agree on the semantics and the language is rich enough, schemata may still not convey the same meaning, due to some hidden semantics, beyond the scope of the schemata. Therefore, [33] argues that "[w]hen no accurate mapping exists, the issue becomes choosing the *best* mapping from the viable ones." The latter approach was later extended to handle top-K schema matchings [17,24,36,37].

Given the discussion above, two main approaches were proposed to handle uncertainty. The first involves the use of semantically oriented methods for schema matching, e.g., S-Match [35], with logical reasoning and the use of external resources (mainly ontologies) to reduce uncertainty. The alternative is to quantify the amount of uncertainty (e.g., [38]) and use various techniques to improve the results, given uncertainty. In our earlier work [17] we used fuzzy logic to model uncertainty. In [39], probability theory was used for the same purpose. Examples for using uncertainty quantification include [24,37]. In the former we proposed the use of the top-K best matchings to improve the accuracy of a matching. The latter proposes to use several matchings and their associated probabilities to provide an estimated response to queries.

The uncertainty involved in the outcome of this or that matcher made the selection process among schema matchers a complex task. First, the number of schema matchers is continuously growing, and this diversity by itself complicates the choice of the most appropriate tool for a given application domain. Second, as one would expect and based on the discussion above, empirical analysis shows that there is no (and may never be) a single dominant schema matcher that performs best, regardless of the data model and application domain [17]. Therefore, several tools, e.g., [7–9,18,19] have combined different schema matchers into *ensembles* to determine the similarity between concepts. The main novelty of this work lies in the ability to automatically determine an effective set of matchers for an ensemble. Works such as [7,40], perform the selection task manually. APFEL [41] learns a threshold that chooses whether to use certain scores, yet does not provide a matcher selection mechanism.

Attribute similarity, as determined by schema matchers in an ensemble can be combined either linearly [42,7] or non-linearly [43], sequentially [40], or in-parallel [42,7]. Up until now, Meta-Learner of LSD (also used in eTuner [20]) is the only matcher that determines matcher weights automatically. In this work, we propose a mechanism (called SMB) for automatically determining linear, in-parallel, weights of ensemble matchers. Our method is therefore similar to that of Meta-Learner. In both approaches a set of matchers is selected and a weighted average of the decisions taken by these matchers determine the matching outcome. However, our technique is different from that of Meta-Learner in two main ways. First, we combine decision-making matchers (see Section 4 for exact definition of decision-making matchers) while Meta-Learner combines attribute similarity measures. Secondly, the weights of Meta-Learner were set using a least-square linear regression analysis while we use the boosting mechanism. The literature shows the connection between boosting and logistic regression [44], yet there is no evident connection to linear regression. Our empirical analysis shows that SMB is superior to eTuner over a wide range of schemata.

Our proposed algorithm is based on a machine learning technique called boosting. Machine learning has been used for schema matching in several works. Autoplex [10] and LSD [42] use a Naïve Bayes classifier to learn attribute mapping probabilities using instance training set. SEMINT [45] use neural networks to identify attribute mappings. APFEL [41] determines heuristic weights in an ensemble and threshold levels using various machine learning techniques, namely decision trees (e.g., C4.5), neural networks, and support vector machines. C4.5 was also used in [46], using WordNet relationships as features. sPLMap [39] use Naïve Bayes, kNN, and KL-distance as content-based classifiers. All these works applied machine learning directly to the schemata, while our approach uses the outcome of other matchers for

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

learning and improvement. In particular, the use of boosting was never applied to schema matching, to the best of our knowledge. Ref. [40] uses a decision tree to determine a sequential activation of matchers. There, decisions are taken manually while our algorithm applied a fully automatic method for determining on the participation of a matcher in an ensemble.

A preliminary version of this work has appeared in [47]. In this work we analyze in greater details the unique ability of SMB to select matchers. Also, the empirical analysis now covers many more aspects that relate to the matcher selection process.

3. Model

We now provide a model for schema matching that will serve us in this work. We shall accompany the description with an example, taken from two hotel reservation Web sites.

3.1. Schema and attributes

Let *schema* $S=\{A_1,A_2,\ldots,A_n\}$ be a finite set of *attributes*. Attributes can be both simple and compound, compound attributes should not necessarily be disjoint, etc. For example, an attribute in a schema of a hotel reservation Web site may be Last Name, First Name, etc. A compound attribute may be Credit Card Info combining three other attributes, Type, Card Number, and Expiry (which could also be a compound attribute, representing month and year of expiration). We note that our model captures the essence of schema matching, namely matching of schema elements, and therefore a richer representation of data models is not needed. Therefore, if we aim at matching simple elements (such as Last Name and First Name) we need not represent their composition into a compound attribute called Name. If the goal of our schema matching process is to match XML paths (see, e.g., [48]), then XML paths are the elements we define as attributes (and hence our matrix elements, see next) in our schemata.

3.2. Attribute matchings and the similarity matrix

For any schemata pair S and S', let $\mathcal{S}=S\times S'$ be the set of all possible *attribute matchings* between S and S'. $\mathcal{S}$ is a set of attribute pairs (e.g., (Please specify your arrival date, check in day)). Let $M(S,S')$ be an $n\times n'$ *similarity matrix* over $\mathcal{S}$, where $M_{i,j}$ represents a degree of similarity between the i-th attribute of S and the j-th attribute of S'. The majority of works in the schema matching literature define $M_{i,j}$ to be a real number in [0,1]. $M(S,S')$ is a *binary similarity matrix* if for all $1\le i\le n$ and $1\le j\le n'$, $M_{i,j}\in\{0,1\}$. That is, a binary similarity matrix accepts only 0 and 1 as possible values.

Similarity matrices are generated by schema matchers. *Schema matchers* are instantiations of the schema matching process [26,49]. They differ mainly in the measures of similarity they employ, yielding different similarity matrices. These measures can be arbitrarily complex, and may use various techniques for name matching, structure matching (such as XML hierarchical representation), etc. Schema matchers use the application semantics in many different ways. Some matchers (e.g., [50,51]) assume similar attributes are more likely to have similar names. Other matchers (e.g., [8,9]) assume similar attributes share similar domains. Others yet (e.g., [10,42]) take instance similarity as an indication to attribute similarity.

Example 1. To illustrate our model and for completeness sake we now present a few examples off schema matchers, representative of many other, similar matchers. Detailed description of these matchers can be found in [9,52][5]:

Term: Term matching compares labels and names to identify syntactically similar terms. To achieve better performance, terms are preprocessed using several techniques originating in IR research. Term matching is based on either complete word or string comparison. As an example, consider the terms `airline information` and `flight airline info`, which after concatenating and removing white spaces become `airlineinformation` and `flightairlineinfo`, respectively. The maximum common substring is `airlineinfo`, and the similarity of the two terms is *length*(`airlineinfo`)/*length* (`airlineinfomation`) $=\frac{11}{18}=61\%$.

Value: Value matching utilizes domain constraints (e.g., drop lists, check boxes, and radio buttons). It becomes valuable when comparing two terms that do not exactly match through their labels. For example, consider attributes `Dropoff Date` and `Return Date`. These two terms have associated value sets {(*Select*),1,2,…,31} and {(*Day*),1,2,…,31}, respectively, and thus their content-based similarity is $\frac{31}{33}=94\%$, which improves significantly over their term similarity (4(`Date`)/11(`Dropoff-Date`)=36%).

Composition: A composite term is composed of other terms (either atomic or composite). Composition can be translated into a hierarchy. This schema matcher assigns similarity to terms, based on the similarity of their neighbors. The Cupid matcher [8], for example, is based on term composition.

Precedence: The order in which data are provided in an interactive process is important. In particular, data given at an earlier stage may restrict the options for a later entry. For example, a hotel chain site may determine which room types are available using the information given regarding the check-in location and time. Therefore, once those entries are filled in, the information is sent back to the server and the next form is brought up. Such precedence relationships can usually be identified by the activation of a script, such as the one associated with a SUBMIT button. Precedence relationships can be translated into a precedence graph. The matching algorithm is

[5] OntoBuilder algorithm description is also available online at http://iew3.technion.ac.il/OntoBuilder/Data/10.OntoBuilder_Papers/dis.pdf.

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

based on a technique we dub *graph pivoting*, as follows. When matching two terms, we consider each of them to be a pivot within its own ontology, thus partitioning the graph into a subgraph of all preceding terms and all succeeding terms. By comparing preceding subgraphs and succeeding subgraphs, we determine the confidence strength of the pivot terms. Precedence was used in [53] to determine attribute correspondences with a holistic matcher.

When encoding the application semantic in a similarity matrix, a matcher would be inclined to put a value of 0 for each pair it conceives not to match, and a similarity measure higher than 0 (and probably closer to 1) for those attribute pairs that are conceived to be correct. This tendency, however, is masked by "noise," whose sources are rooted in missing and uncertain information. This argument was also raised in [54]: "the syntactic representation of schemas and data do not completely convey the semantics of different databases," i.e., the description of a concept in a schema can be semantically misleading. Therefore, instead of expecting a binary similarity matrix, with a 0 score for all incorrect attribute mappings and a unit score for all correct attribute mappings, we would expect the values in a similarity matrix to form two probability distributions over [0,1], one for incorrect attribute mappings (with higher density around 0), and another for correct mappings [55].

Schema matchers use data model semantics when determining the similarity between attributes. For example, XML structure has been used in Cupid [8] to support or dispute linguistic similarities. Also, the similarity flooding algorithm [23] uses structural links among attributes to update linguistic similarities. However, once this similarity has been determined and recorded in the similarity matrix, it is no longer needed. Therefore, the matrix representation as given above is sufficient as a data model, representing the uncertainty involved in the matching process.

3.3. Schema matching

Let the power-set $\Sigma = 2^{\mathcal{S}}$ be the set of all possible *schema matchings* between the schema pair and let $\Gamma : \Sigma \to \{0,1\}$ be a Boolean function that captures the application-specific constraints on schema matchings, e.g., cardinality constraints and inter-attribute mapping constraints. Given a constraint specification Γ, the set of all *valid* schema mappings in Σ is given by $\Sigma_\Gamma = \{\sigma \in \Sigma | \Gamma(\sigma) = 1\}$. We define Γ here as a general constraint model, where $\Gamma(\sigma) = 1$ means that the mapping σ can be accepted by a designer. Γ was modeled in the literature using a special type of matchers, called *constraint enforcers* [20] and their output is recorded in a similarity matrix, as detailed in Section 4. We say Γ is a *null constraint function* (basically accepting all possible matchings as valid with no use of a constraint enforcer) if for all $\sigma \in \Sigma$, $\Gamma(\sigma) = 1$.

Formally, the input to the process of schema matching is given by two schemata S and S' and a constraint Boolean function Γ.[6] The output of the schema matching process is a *schema matching* $\sigma \in \Sigma_\Gamma$, represented as a similarity matrix $M(S,S')$. An attribute pair can be in the output schema matching only if it has a value larger than 0 in M.

4. Second-line schema matchers

As a preamble to presenting our matcher selector heuristic, we now define second-line matchers (2LM). The input to this type of matchers is no longer the schemata S and S', but rather a similarity matrix $M(S,S')$ (together with Γ). Given schemata S and S', we denote by $\mathcal{M}(S,S')$ the (possibly infinite) set of similarity matrices $M(S,S')$. A second-line schema matcher $SM : \mathcal{M}(S,S')^* \times \Gamma \to \mathcal{M}(S,S')$ is a mapping, transforming one (or more) similarity matrices into another similarity matrix.

Second-line schema matchers are different from first-line schema matchers in that they operate solely on similarity matrices. First-line schema matchers operate on the schemata themselves, using the semantics of the application. For example, a linguistic matcher is a first-line schema matcher, using attribute names or description in matching attributes. One can envision a second-line matcher that receives as an input the similarity matrix that was generated by the linguistic matcher and improves it, e.g., by thresholding or by combining it with a matrix of another matcher.

Example 2. In Example 1, we have introduced several matchers, all of which fall into the category of first-line matchers. Two simple examples of second-line matchers are:

Term and Value: A weighted combination of the Term and Value matchers. Here, the input to the matcher involves similarity matrices.

Combined: A weighted combination of the Term, Value, Composition, and Precedence matchers.

A few more second-line matchers that are based on constraint satisfaction are:

- The Maximum Weighted Bipartite Graph (MWBG) algorithm and the Stable Marriage (SM) algorithm, both enforce a cardinality constraint of 1–1. In [52] we have introduced a heuristic we dub Intersection that simply computes and outputs the intersection set of both algorithm outputs. For comparison sake, we also suggest Union, which includes in the output mapping any attribute mapping that is in the output of either MWBG or SM. All four matchers (MWBG, SM, Intersection and Union) are second-line matchers. It is worth noting that neither Intersection nor Union enforce 1–1 matching.
- A variation of the SM matcher is the Dominants matcher. The matcher chooses *dominant pairs*, those pairs in the similarity matrix with maximum value both in their row and in their column. The main assumption guiding this heuristic is that the dominant

[6] For ease of exposition, we constrain our presentation to a matching process of two schemata.

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

pairs are the most probable to be in the exact matching since the two attributes involved in a dominant pair prefer each other most. Note that with this heuristic not all the target attributes are mapped and that an attribute in one schema may be mapped to more than one attribute in another schema, whenever attribute pairs share the same similarity level.

- Finally, in [55] we have introduced 2LNB, a second-line matcher that uses a Naïve Bayes classifier over matrices to determine attribute matchings. Autoplex [10], LSD [42], iMAP [56], and sPLMap [39] also use a Naïve Bayes classifier to learn attribute matching probabilities using instance training set. 2LNB is the only second-line matcher in this group.

We now provide two more examples of second-line matchers, highlighting the differences in their modus operandi from first-line matchers.

Example 3 (*eTuner*). A model of a 1–1 matching system was defined in [20] to be a triple, one element being a library of matching components. This library has four types of components, namely Matcher, Combiner, Constraint Enforcer, and Match Selector. The first type is a first-line schema matcher, in its classical definition. The remaining three types are second-line schema matchers according to our definition.

A *combiner* [7] follows the definition of a schema matcher with a null constraint function. A combination can be done by aggregating elements of the input matrices or by using machine learning techniques such as stacking and decision trees.

A *constraint enforcer* is simply a second-line matcher (note that our definition in Section 3.3 allows adding constraints at the first-line as well).

A *match selector* returns a matrix, in which all elements that are not selected are reduced to 0. Two examples are given in [20], thresholding and the use of the MWBG algorithm for selecting a maximum weighted bipartite graph.

Example 4 (*Top-K*). A heuristic that utilizes the top-*K* best schema matchings to produce an improved schema matching was proposed in [24]. It is a special type of a combiner and a match selector, in which the input does not come from different matchers (as is generally done with ensembles [18,9,57,19]). Rather, the same schema matcher generates multiple matrices that are then evaluated to generate a single similarity matrix by a special form of thresholding.

Comparing Examples 3 and 4 provides interesting observations. First, the modeling of second-line matchers can serve as a reference framework for comparing various research efforts in schema matching. Therefore, while combiners and match selectors are defined to be separate types in [20], they were combined and redefined in [24]. A second observation involves the goal of second-line matchers. Second-line matchers aim at improving on the outcome of first-line schema matchers, striving to increase robustness of first-line matchers. The idea was deemed appealing since complementary matchers can potentially compensate for the weaknesses of each other [57]. In [24] it was shown that the use of a heuristic, based on top-*K* best schema mappings, has increased the precision of mappings by 25% on average, at the cost of a minor 8% reduction in recall.

We now propose a classifications of matchers on two orthogonal dimensions (see Table 1 for classification and example matchers). The first dimension separates first from second-line schema matchers. The second dimension separates those matchers that aim at specifying schema matchings, dubbed *decision makers* from those that compute similarity values yet do not make decisions at the schema level. The most common type is a non-decisive first-line matcher. The OntoBuilder's Term matcher belongs to this class as well as a WordNet-based decision tree technique proposed in [18]. Combiners, in COMA's terminology, are non-decisive second-line schema matchers. They combine similarity matrices of other matchers and hence they are second-line matchers by definition. However, their similarity matrix is not meant to be used to decide on a single schema matching.

Well known decisive second-line matchers are algorithms like MWBG and SM. Both algorithms fall into the category of constraint enforcers in [20] and both enforce a cardinality constraint of 1–1. Finally, the class of first-line decision makers contains few if any matchers. The main reason is that most systems abide by the long conceptual modeling tradition of database schema integration, as summarized in [1]: "The comparison activity focuses on primitive objects first…; then it deals with those modeling constructs that represent associations among primitive objects." This dichotomy was mainly preserved in schema matching as well.

As a concluding remark we compare the proposed classification with the classifications of [5,26]. In [5], matchers are partitioned into *individual matchers* and *combining matchers*. The latter class contains only second-line schema matchers (both decisive and non-decisive). Individual matchers can also serve as second-line matchers. For example, a matcher that takes the outcome of another matcher and apply a threshold condition on it is an *individual*, *second-line* matcher. Combining matchers are further partitioned into composite and hybrid matchers, a classification that is less relevant in our classification, where the secondary partitioning is based on the decisiveness of a matcher. In [26], the *alignments as solutions* class is the same as *second-line decisive* matchers and the class of *alignments as likeness clues* contains the class of *non-decisive matchers* (may they be first-line or second-line). No special treatment is given to the separation of matchers that make use of application semantics to those that rely solely on the outcome of previous matchers.

Table 1
Two dimension matcher classification.

Matcher	First-line matcher	Second-line matcher
Non-decisive	Term	Combined
Decision maker		MWBG

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

5. Boosting schema matching using second-line matchers

In this section we use the notion of second-line matchers to provide a new heuristic to constructing ensembles. In Section 3 we have described how the decision making of schema matchers is masked by "noise." Nevertheless, research has shown that many schema matchers perform better than random choice. In the Appendix we present the monotonicity principle, as introduced in [17] and we argue that any (statistically) monotonic matcher is a weak classifier [21]. A *weak classifier* is a classifier which is only slightly correlated with the true classification and its hypotheses are at least slightly better than random choice. The theory of weak classifiers has led to the introduction of a class of *Boosting* algorithms (e.g., [21]). This class of algorithms can strengthen weak classifiers to achieve arbitrarily high accuracy and has been shown to be effective in the construction of successful classifiers. Given a set of weak classifiers, the algorithm iterates over them while re-weighting the importance of elements in the training set. There exist many versions of boosting to-date. In this paper we build upon the AdaBoost algorithm [22], described in Section 5.1 for completeness sake. AdaBoost is the most popular and historically most significant boosting algorithm. Section 5.2 then introduces our new heuristic, Schema Matcher Boosting (SMB), followed by a discussion on how to improve the training process (Section 5.3).

5.1. AdaBoost

The input to a boosting algorithm is a **set** of m examples where each example (x_i,y_i) is a pair of an instance x_i and the classification of the instance mapping, y_i. While not necessarily so, y_i typically accepts a binary value in $\{-1,+1\}$, where -1 stands for an incorrect classification and $+1$ stands for a correct classification. Therefore, the algorithm is aimed at binary classifications. The last input element is a hypothesis space $\mathcal{H}$, a set of weak classifiers.

Algorithm 1. Boosting.

1: Input: $S=\{(x_1, y_1), \ldots, (x_m, y_m)\}$, and a space hypotheses $\mathcal{H}$.
2: /* $\forall 1 \le i \le m, x_i \in \mathcal{X}$, and $\forall 1 \le i \le m, y_i \in \{-1,+1\}$ */
3: /* initialization: */
4: **for all** $1 \le i \le m$ **do**
5: $D_1(i) = 1/m$
6: **end for**
7: $t=1$
8: **repeat**
9: /* training phase: */
10: Find the classifier $h_t : \mathcal{X} \to \{-1,+1\}$, $h_t \in \mathcal{H}$ that minimizes the error with respect to the distribution D_t: $h_t = \arg_{h_j} \min \varepsilon_j$.
11: **if** $\varepsilon_t \le 0.5$ **then**
12: Choose $\alpha_t \in R$. $\alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}$
13: Update $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ where Z_t is a normalization factor
14: $t=t+1$
15: **end if**
16: **until** $t=T$ or $\varepsilon_t > 0.5$
17: /* upon arrival of a new instance: */
18: Output the final classifier: $H(x) = \text{sign}(\sum_{k=1}^{\min(t,T)} \alpha_k h_k(x))$

The algorithm works iteratively. In each iteration the input set is examined by all weak classifiers. However, from iteration to iteration the relative weight of examples changes. The common technique in the boosting literature, which we follow here as well, is to place the most weight on the examples most often misclassified in preceding iterations; this has the effect of forcing the weak classifiers to focus their attention on the "hardest" examples. Lines 4–6 of the algorithm assign an initial equal weight to all examples (see Section 5.3 for a revision of this initialization). Weights are updated later in line 13 (see below). An iteration counter t is set to 1 in Line 7 and incremented in Line 14.

Line 10 applies weak classifiers in parallel, looking for the most accurate h_t over the weighted examples. The amount of error of each weak classifier is computed. The error measure may take many forms (see discussion below) and in general should be proportional to the probability of classifying incorrectly an example, under the current weight distribution ($\Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$). At round t, the weak classifier that minimizes the error measure of the current round is chosen.

Lines 11 and 16 provide a stop condition, limiting the amount of error to be no more than 50%. In addition, a restriction on the maximum number of iterations is also part of a stop condition. In Line 12, the amount of change to example weights α_t is determined. In [58], it was shown that for binary classifiers, training error can be reduced most rapidly (in a greedy way) by choosing α_t as a smoothing function over the error. Such a choice minimizes $Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$. In Line 13, the new example weights are computed for the next round $(t+1)$, using Z_t as a normalization factor.

Lines 1–16 of Algorithm 1 serve for training the algorithm weights. These weights are then used in Line 18 to classify a new instance x, by producing $H(x)$ as a weighted majority vote, where α_k is the weight of the classifier chosen in step k and $h_k(x)$ is the decision of the classifier of step k.

5.2. SMB: Schema Matcher Boosting

The Boosting algorithm is trivially simple. However, Algorithm 1 is merely a shell, serving as a framework to many possible instantiations. What separates a successful instantiation from a poor one is the selection of three elements, namely the instances (x_i), the hypothesis space $\mathcal{H}$, and the error measure ε_t. The design of SMB is affected by the need to select prominent matchers from a pool of many possible matchers. SMB performs the selection process sequentially. It starts by greedily choosing those matchers that provide a correct solution to a major part of the schema matching problem. Then, it adds matchers that can provide insights to solving the harder problems. Those matchers that are left out will not be part of the ensemble. We next show the SMB heuristic as a concrete instantiation of Algorithm 1, tailor-made to our specific problem domain of schema matching.

The example set $\{(x_i,y_i)\}$ consist of a set of attribute pairs (x_i is a pair!), one attribute from each schema, and of the classification of the instance mapping y_i. Such a pair represents an attribute matching. This approach can be

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

easily extended to select multiple attributes from each schema, as long as the matcher itself can assess the similarity measure of multiple attributes. Also, examples can be designed to be sets of attributes from multiple schemata rather than a pair. Each instance x_i can be correct (i.e., belongs to the exact matching) or incorrect. Therefore, y_i can have two possible values (+1) (for a correct matching) and (−1) (for an incorrect matching).

Choosing the hypothesis space is more tricky. We note that the boosting algorithm we chose deals with binary classifications. Hence, a given attribute pair is either matched or not. A natural choice of hypotheses will therefore contain a set of decision makers (see discussion in Section 4), whose decision is based on the outcome of some first-line schema matcher. Therefore, we define SMB to be a second-line matcher, $\mathrm{SMB}: \mathcal{M}(S,S')^* \times \Gamma \rightarrow \mathcal{M}(S,S')$, and the elements of the hypothesis space are binary matrices. For example, a hypothesis h in $\mathcal{H}$ is (Term, Dominants), where the Dominants second-line matcher is applied to the outcome of the Term first-line matcher. Among other things, Dominants serves in enforcing the domain constraints, as expressed by Γ. It is worth noting that SMB is also a decision maker and the outcome of SMB is a binary matrix.

Finally, we address the form of the error measure ε. A matcher can either determine a correct attribute matching to be incorrect (false negative) or it can determine an incorrect attribute matching to be correct (false positive). Let A_t denote the total weight of the false negative examples, C_t denotes the total weight of the false positive examples, and B_t denotes the total weight of the true positive examples, all in round t. Typically, one would measure error in schema matching in terms of precision and recall, translated into boosting terminology as follows:

$$P(t) = \frac{B_t}{C_t + B_t}, \quad R(t) = \frac{B_t}{A_t + B_t} \tag{1}$$

Precision and recall may be combined in many ways, one of which is the *F*-Measure, their harmonic mean:

$$F(t) = \frac{2B_t}{A_t + C_t + 2B_t} \tag{2}$$

and therefore, a plausible error measure for the SMB heuristic is

$$\varepsilon_t = 1 - F(t) = 1 - \frac{2B_t}{A_t + C_t + 2B_t} = \frac{A_t + C_t}{A_t + C_t + 2B_t} \tag{3}$$

This is indeed the measure we present in this paper. It is worth noting, however, that this is not the only measure possible. Our empirical evaluation (not shown in this work) suggests that Eq. (3) performs better than other error measures.

Example 5. The example is taken from one of our experiments, described in Section 6. Given the hypotheses space $\mathcal{H}$ as described above, and given a data set of size 70, the SMB heuristic performs five iterations: It started by creating a data set with equal weight for each mapping. Table 2 presents the results of each run. The first column contains the number of the run, followed by the selected hypotheses. The values of the error (ε) and the amount of change to example weights (α) are given in the last two columns. In the sixth iteration no hypothesis had error of less than 50% so the training phase is terminated having five iterations, each with its strength α_t. The outcome classification rule is a linear combination of the five weak matchers with their strength as coefficients. So, given a new mapping (a,a') to be classified, each one of the weak matchers contributes to the final decision with its decision weighted by its strength and if the final decision was positive then the given mapping would be classified as a correct mapping. Otherwise, it would be classified as an incorrect one.

Table 2
An example run of SMB.

Iteration number	Selected matcher	ε	α
1	(Composition,Dominants)	0.328	0.359
2	(Precedence, Intersection)	0.411	0.180
3	(Precedence, MWBG)	0.42	0.161
4	(Term and Value, Intersection)	0.46	0.080
5	(Term and Value, MWBG)	0.49	0.020

Let $\tilde{h}_{\max}$ be the maximum execution time of a matcher in $\mathcal{H}$ and $t_{\max}$ be the number of iterations performed by SMB. The training time of SMB is $O(\tilde{h}_{\max} \cdot t_{\max})$. Given a new schema pair, let $n_{\max}$ be the maximum number of attributes in each schema. The cost of using SMB is $O(n^2_{\max})$, the cost of generating the output matrix.

Two comments about α_t: First, our choice of α_t limits it to be non-negative, since ε_t is restricted not to exceed 0.5 (see lines 11 and 12 of Algorithm 1). This is one characteristic that differentiates SMB from the Meta-Learner of LSD, which uses a least-square linear regression on the training data set. We shall elaborate on this difference more in Section 6.7. Secondly, if a hypothesis is chosen more than once during the training phase, its total weight in the decision making process is the sum of all the weights α_t with which it has been assigned.

5.3. Preprocessing the training data set

Tracing the evolution of the error computed in each iteration of SMB, we observe that error increases rapidly. Recall that the error is the sum of all the weights of the incorrectly classified examples. Therefore, we hypothesize that this phenomenon is a result of outliers, i.e., examples that are inherently ambiguous and hard to categorize. Therefore, *no* matcher in the ensemble classifies them correctly. Even with a small number of outliers, the emphasis placed on the incorrectly classified examples becomes detrimental to the performance of SMB. Such examples receive increasing weights with each new iteration. Consequently, error accumulation accelerates rather than subsides. Regardless of which classifier is chosen, such examples will be misclassified and their weights will increase.

To avoid this phenomenon, we introduce a preprocessing phase to the training phase of SMB, in which we identify and filter outliers. These examples are de-emphasized by eliminating them from the training set to avoid rapid error accumulation. We remove all examples that no matcher classifies correctly. Eliminating

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

examples from the training set is equivalent to presetting small weights to outliers. Our empirical analysis has shown that the preprocessing stage yields a significant improvement in performance.

6. Experiments with SMB

6.1. Experiment setup

In our experiments we have used 30 matcher combinations (recall that our hypothesis space is made of matching pairs), combining Term, Value, Composition, Precedence, Term and Value, and Combined with MWBG, SM, Dominants, Intersection, Union, and 2LNB. All matchers were described in Sections 3.2 and 4. All algorithms were implemented using Java 2 JDK version 1.5.0_09 environment, using an API to access OntoBuilder's matchers and get the output matrices. The experiments were run on a laptop with Intel Centrino Pentium, 1.50 GHz CPU, 760 MB of RAM Windows XP Home edition OS.

The Term and Combined matchers were shown in [17] to be monotonic. Our preliminary experiments show that Value was not monotonic, and is brought here as a baseline case. To demonstrate the potency of our matchers, we have experimented with the OAEI 2006 Directory benchmark.[7] Our empirical analysis yields that the pair (Term, MWBG), for example, achieved on average Precision of 61%, Recall of 96%, and *F*-Measure of 72%, on a set of 110 randomly selected tasks. This is better than other known results on this data set.

6.2. Data set

For our experiments, we have selected 230 Web forms from different domains, such as job hunting, Web mail, and hotel reservation. We extracted a schema from each Web form using OntoBuilder. We have matched the Web forms in pairs (115 pairs), where pairs were taken from the same domain, and generated manually the exact matching for each pair.[8] The schemata vary in size, from 8 to 116 attributes with about two-thirds of the schemata have between 20 and 50 attributes. They also vary in the proportion of number of attribute pairs in the exact matching relative to the target schema.[9] This proportion ranges from 12.5% to 100%; the proportion in about half of the ontologies is more than 70%, which means that about 70% of the schema attributes can be matched. Another dimension is the size difference between matched schemata, ranging from equal size schemata to about 2.2 times difference between schemata. In about half of the pairs, the difference was less than 50% of the target schema size.

We ran the six schema matchers (Term, Value, Composition, Precedence, Term and Value, and Combined) on the 115 pairs, generating 690 matrices. These matrices used the second-line matchers (MWBG, SM, Dominants, Intersection, Union, and 2LNB) to generate new matrices. 2LNB was paired only with the Combined matcher. All in all, we have analyzed 3565 pairs of real-world schemata. SMB was tested in two different settings. In the first, dubbed SMB-All, SMB was trained with all 31 matcher pairs. In the second setting, dubbed SMB-Recall, we have trained SMB on 12 matcher pairs whose individual performance is more geared towards higher Recall, and share the second-line matchers Union and MWBG. The latter setting was developed to test SMB abilities in an environment that promotes Recall.

6.3. Evaluation methodology

We have repeated experiments with a varying size of training data set. Here, we report on experiments with a training set of 60 randomly selected schema pairs and a test set of size 30 (schema pairs) that was also selected randomly from the remaining matrices. We have repeated each experiment three times. Preprocessing to avoid outliers was applied.

To evaluate the various heuristics, we use Precision and Recall. Lower Precision means more false positives, while lower Recall suggests more false negatives. To extend Precision and Recall to the case of non-1–1 mappings, we have adopted a correctness criteria according to which any attribute pair that belongs to the exact mapping is considered to be correct, even if the complex mapping is not fully captured.

6.4. Comparative performance analysis

We first analyze comparatively the performance of SMB with the 31 matcher pairs. Fig. 1 (top) positions all 32 matchers on a Precision (*x*-axis) vs. Recall (*y*-axis) scatter plot. The expected Precision/Recall trade-off is evident here, with no single dominating matcher. Matchers are partitioned into two groups. To the right there are all those matchers that qualify as weak classifiers, which we define to be those whose *F*-Measure is higher than 50%. To the left, there are five matchers that cannot be considered weak matchers. In common to all five matcher pairs is the use of the Value matcher. This matcher is not statistically monotonic, since it cannot differentiate between pairs that share the same attribute domain. However, when combined with the Term matcher, Value generally adds 1–2% to the Term matcher performance.

SMB is clearly the winner in terms of Precision, balancing Precision with Recall. Fig. 1(bottom) illustrates the percentage of improvement SMB provides in terms of Precision. The *x*-axis represents the different matcher pairs while the *y*-axis shows the percentage of improvement. It ranges from 5.4% to 66% for weak matchers. For

[7] The benchmark is publicly available at http://keg.cs.tsinghua.edu.cn/project/RiMOM/oaei2006/oaei2006.html. Our results are based on a private evaluation of the exact matching, since the OAEI organizers do not provide the exact matching.

[8] All ontologies and exact matchings are available for download from the OntoBuilder Web site, http://ie.technion.ac.il/OntoBuilder.

[9] In OntoBuilder, one of the schemata is always chosen to be the target schema, the schema against which comparison is performed.

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

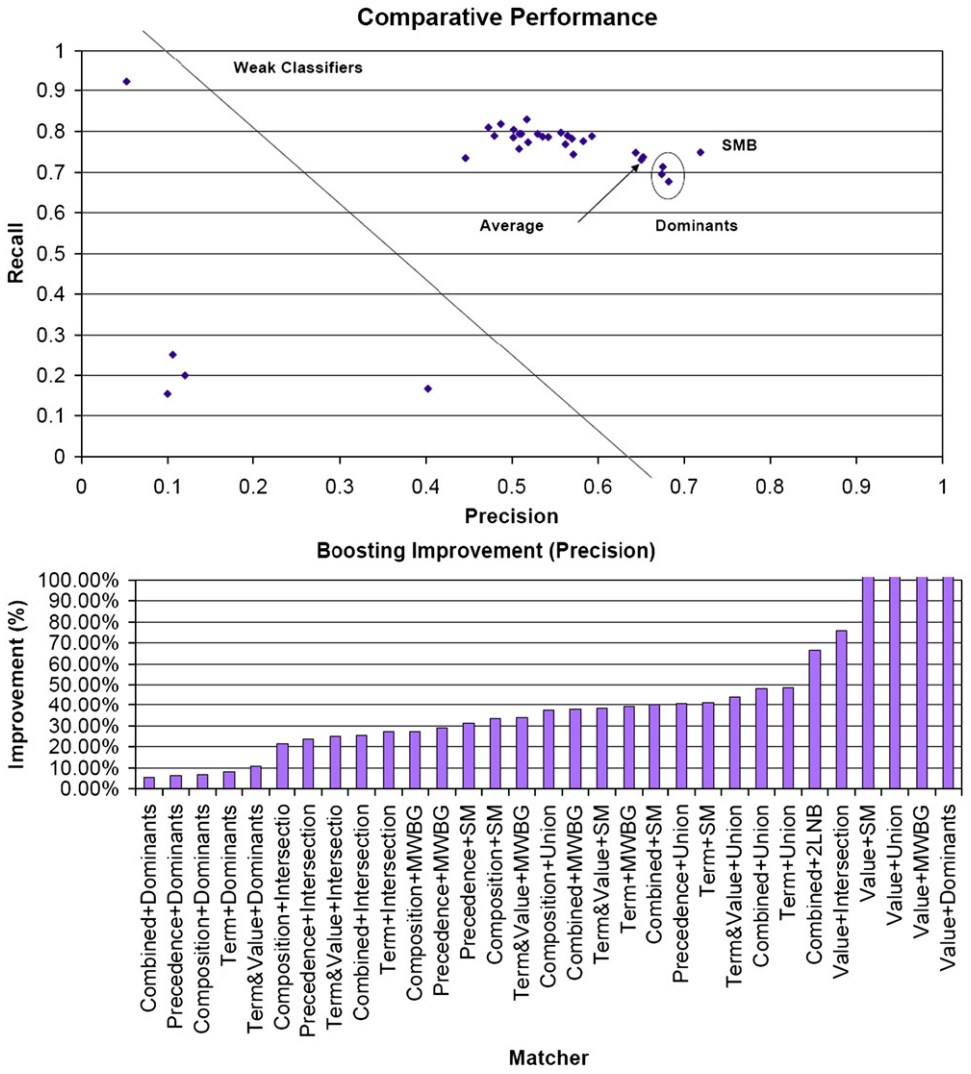

Fig. 1. Performance analysis.

example, SMB improved by 39% over the pair (Term, MWBG), illustrated earlier to have good outcomes on tough data sets (such as the directory data set of OAEI'2006). In terms of *F*-Measure, we observe an improvement of 4.3–34.3% for weak matchers.

For comparison, Fig. 1(top) also contains another matcher weighing technique, Average, which is discussed in detail later in this section. While Average blends in with other matchers (mainly from the (*, Dominants) set), SMB stands out in its Precision performance, while not compromising much its Recall. The group of (weak classifiers) matchers that dominate the Recall measure all use Union as a second line matcher. The performance of SMB with respect to Recall is analyzed in Section 6.6.

Such an improvement is nice, yet not unheard of. For example, LSD has shown an improvement of 5–22% in Precision [42]. We defer a comparison with LSD to later in the section, arguing here that these results are comparable. While SMB may be better than the other individual matchers on average, how often does it manage to outperform all other matchers? we have analyzed the data and the results are illustrated in Fig. 2. For each matcher, we record the percentage of schema pairs, where it was not outperformed by any of the other matchers in terms of Precision, Recall, and *F*-Measure. The figure provides a comparison of all weak classifiers and SMB. SMB clearly performs the best in terms of Precision and *F*-Measure. In 43% of the schema pairs, its Precision performance was not dominated by any other matcher. The next best matcher in this category was (Precedence, Dominants), non-dominated in only 28% of the cases.[10] Similar results are observed for the *F*-Measure, where SMB leads with 38%, followed by (Term&Value,

[10] Note that the sum of non-dominance percentage exceeds 100% since matchers may reach the same level of precision for some instances, counting both to be non-dominated for this instance.

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

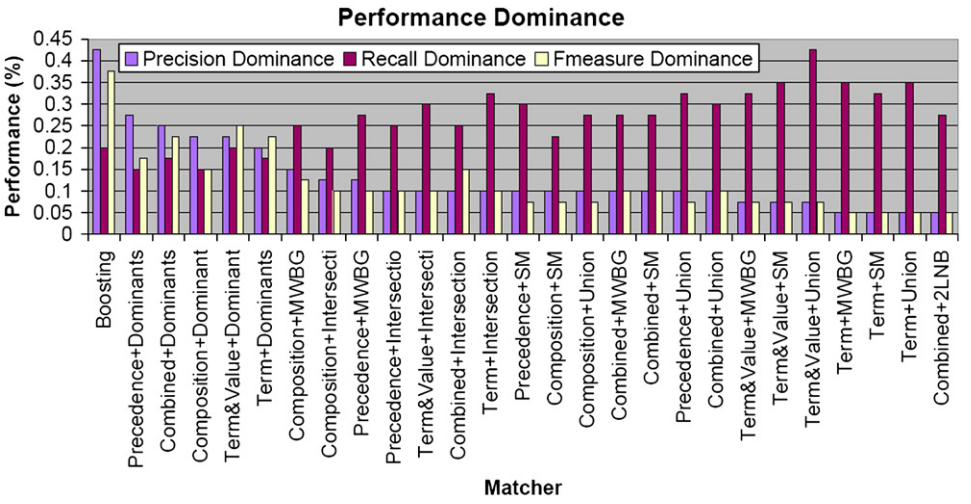

Fig. 2. Dominance analysis.

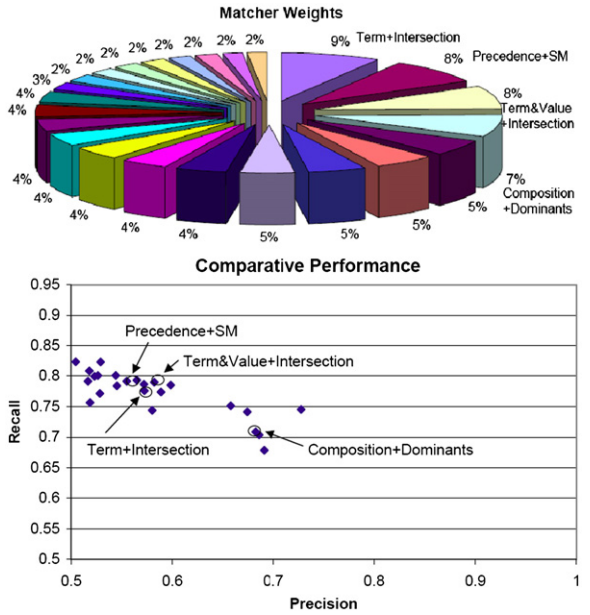

Fig. 3. Relative matcher weights in SMB and individual performance.

Dominants) with 25%. For Recall, SMB is non-dominated for 20% of the schema matching pairs.

6.5. Matcher selection

We now analyze the decision making process of SMB. Given the individual performance of each matcher, one could expect that those matchers with the highest weight in the decision making of SMB will be those that perform best individually. In our case, the top four matchers, in terms of Precision and *F*-Measure are pairs in which the second-line matcher is Dominants. Fig. 3 presents the relative matcher weights in SMB. The higher the weight, the more important is the vote of a matcher regarding each attribute matching. Only 24 out of the 31 matchers participate in the decision making of SMB.

Surprisingly, the top four matchers in Fig. 3(left) include only one pair with Dominants ((Composition, Dominants)). The pair (Combined, Dominants) is the leading pair in terms of Precision when observing individual performance, yet is not even part of the SMB decision making! The most important matcher for SMB is (Term, Intersection), ranked 11-th according to the *F*-Measure individual performance and 10-th according to Precision. (Precedence, SM), ranked second for SMB, has a mediocre individual performance. Fig. 3(right) is a zoomed-in version of Fig. 1(left), highlighting the four top matchers of SMB.

Our first observation is that the decision making of SMB is not linear in the individual performance of matchers, and therefore the training process of SMB is valuable. Secondly, we observe that SMB seeks diversity in its decision making. It uses Term, Value (combined with Term due to its individual poor performance), Composition, and Precedence. Given these four matchers, SMB has no need for the Combined matcher, which provides a weighted average of the four. This explains the absence of (Combined, Dominants).

SMB has the ability to *choose* matchers for an ensemble. eTuner suggests a method for tuning "knobs" given an ensemble but does not provide a method for constructing it. LSD also applies the Meta-Learner to an existing ensemble. In our experiments, SMB includes only 24 out of the 31 matchers. We consider this feature as a main contribution of the proposed algorithm.

6.6. Recall performance analysis

In the experiments summary discussed in Section 6.4, SMB improved much on Precision while sacrificing Recall to a certain extent. To understand this phenomenon, recall that the error measure that was defined for SMB balances Precision with Recall. SMB chose matchers that improve Precision. For example, Intersection is a conservative

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

matcher that selects an attribute matching only if both MWBG and SM selects it. Dominants is yet another example of a conservative matcher that aims at higher Precision, paying with Recall. By choosing matchers that emphasize Precision, SMB itself will demonstrate a higher level of Precision.

To isolate the impact second-line matchers have on Precision and Recall, we conducted another experiment, in which SMB is allowed to use only matcher pairs that put emphasis on Recall. For that purpose, we restricted our set of hypotheses to 12 matcher pairs, all share the Union or MWBG second-line schema matchers. Union is clearly geared towards Recall. MWBG showed empirically good Recall performance and was added here to give SMB the freedom of combining different opinion. We conducted the experiment in the same manner as discussed Section 6.3. Fig. 4 illustrates the outcome of this set of experiments. SMB-Recall improves on the average performance of other matcher pairs in Recall while sacrificing somewhat on Precision. SMB does not perform as well in this experiment as it does in the experiment where all matchers were at its disposal, which indicates that a larger variety leads to better outcomes. In particular, in this experiment SMB also showed its unique ability in choosing matchers, choosing four out of the 12 matchers, including a surprising selection of (Value, MWBG). This matcher, with its poor individual performance, has a weight of 25% in the decision making.

6.7. Weight selection

The outcome of SMB matcher training is a weighted average for matcher voting. Our next set of experiments, summarized in Table 3, compare the outcome of using SMB weights with other weighing schemes.

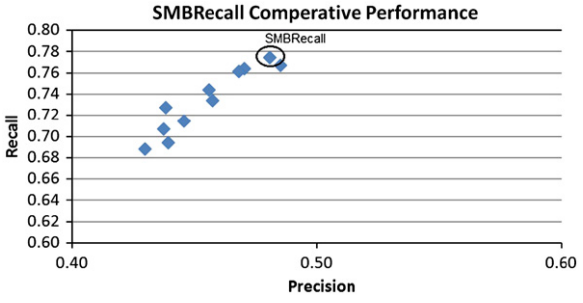

Fig. 4. Comparative performance of matcher sets with Union and MWBG as a second-line matcher and SMBRecall.

Table 3
Comparison of weight schemes.

Weight scheme	Avg. precision	Avg. recall	Avg. *F*-Measure
Boosting	0.73	0.75	0.74
Meta-Learner	0.57	0.69	0.62
F-Measure	0.61	0.72	0.64
Average	0.65	0.73	0.67
Random	0.57	0.65	0.61

The first row represents the average performance of SMB, as presented above (Fig. 1). In the second row, the performance of the LSD's Meta-Learner is presented. Given a set of weak learners $\mathcal{H}$, the Meta-Learner uses a least-square linear regression on the training data set, minimizing the squared error

$$\sum_{i=1}^{m}\left(y_i - \sum_{h\in\mathcal{H}} h(x_i)\cdot w_h\right)^2 \quad (4)$$

where y_i is set to 0 if the pair x_i should not be matched and 1 otherwise. $h(x_i)$ is the decision of learner h regarding pair x_i and w_h are the variables on which the linear regression is applied. The Meta-Learner cannot choose classifiers and therefore, with 31 different learners, a huge space of possibilities exist. To allow a comparison using some common baseline, we have selected the top 16 matchers chosen by SMB to participate in the training of the Meta-Learner. The remaining eight matchers seem to have little impact on the decision making of SMB. The most dominant matcher was (Combined, Intersection), which was ranked 8-th by SMB, demonstrating that the Meta-Learner and SMB reach different decisions regarding matcher importance. For example, the matcher (Term, Dominants) (ranked 9-th by SMB) has received a negative weight. It is worth noting that unlike SMB, the weights of the Meta-Learner can be negative as well. This has the interesting effect of transposing the decision of a matcher.

Once the weights have been set, a set of 30 schema pairs was chosen randomly and generated the outcome using the weights w_h from the training phase. The Meta-Learner reached a Precision of 57%, a Recall of 69%, and an *F*-Measure of 62%. Comparing with the results of SMB, we observe that SMB performs 28% better than the Meta-Learner in terms of Precision, 9% better in terms of Recall and 19% better in terms of *F*-Measure.

In the third line we present the results of matching 30 randomly chosen schema pairs, where matchers are assigned a weight equivalent to their *F*-Measure. For example, (Precedence, Dominants) is assigned a weight of 0.69 while (Value, Dominants) is assigned a weight of 0.11. This weighing scheme reduces Precision by about 20% on average, Recall by about 4% on average, and *F*-Measure by about 16% on average. The performance of weighing using *F*-Measure is worse (!) than those of assigning equal weights to the various matchers (fourth line of Table 3; also presented in Fig. 1(left)). SMB improves Precision by 12%, Recall by 3%, and *F*-Measure by 11%.

The fifth row provides the average result of three random weight selections, each time testing the random weight over 30 randomly selected schema pairs. SMB improves Precision by 28%, Recall by 15%, and *F*-Measure by 21%.

To conclude, in this set of experiments, SMB is shown to dominate all other tested weighing scheme, in terms of Precision, Recall, and *F*-Measure. This, together with the ability of SMB to select matchers for an ensemble, sums up to show SMB to be the best choice for ensemble design.

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

7. Conclusions

In this work we have focused on the selection process of schema matchers into ensembles. We presented the Schema Matcher Boosting (SMB) heuristic to efficiently use an ensemble of matchers. SMB uses the separation of first and second-line matchers in devising an effective mechanism to improve on existing methods. We have analyzed, both conceptually and empirically, the properties of SMB, discussing its benefit and analyzing its performance. SMB has the unique ability to choose from a pool of matchers. Its decision making is based on diversity of matchers, taking their best combination. Our empirical analysis also shows that SMB provides a major increase in Precision with no or minimal loss of Recall. SMB performance improves on any individual matcher pair with which we have experimented and was shown to dominate other weighing schemes, including that of LSD Meta-Learner.

In our future work, we aim at improving SMB even more. We intend to look at existing works in boosting, involving multiclass classification (e.g., AdaBoost.M1 and AdaBoost.M2, [58]) and error-correcting output codes [59], possibly identifying new ties to the schema matching problem. Another direction will be to incorporate human knowledge, as was suggested by several schema matching papers in the past. We shall look into works such as [60], where human judges construct estimated probability. This approach was argued to be too hard for experts to deal with in schema matching, so we shall look into indirect methods for building such estimated probability functions.

Appendix A. Monotonicity

The evaluation of schema matchings is performed with respect to an *exact matching*, based on expert opinions. *Precision* and *recall* are used for the empirical evaluation of performance. Assume that out of the $n \times n'$ attribute matchings, there are $c \leq n \times n'$ correct attribute matchings, with respect to the exact matching. Also, let $t \leq c$ be the number of matchings, out of the correct matchings, that were chosen by the matching algorithm and $f \leq n \times n' - c$ be the number of incorrect such attribute matchings. Then, precision is computed to be $t/(t+f)$ and recall is computed as t/c. Clearly, higher values of both precision and recall are desired. From now on, we shall focus on the precision measure, denoting by $p(\sigma)$ the precision of a schema matching σ.

We first create equivalence schema matching classes on 2^S. Two matchings σ' and σ'' belong to a class p if $p(\sigma') = p(\sigma'') = p$, where $p \in [0,1]$. For each two matchings σ' and σ'', such that $p(\sigma') < p(\sigma'')$, we can compute their schema matching level of certainty, $\Omega(\sigma')$ and $\Omega(\sigma'')$. We say that a matching algorithm is *monotonic* if for any two such matchings $p(\sigma') < p(\sigma'') \rightarrow \Omega(\sigma') < \Omega(\sigma'')$. Intuitively, a matching algorithm is monotonic if it ranks all possible schema matchings according to their precision level.

A monotonic matching algorithm easily identifies the exact matching. Let σ^* be the exact matching, then

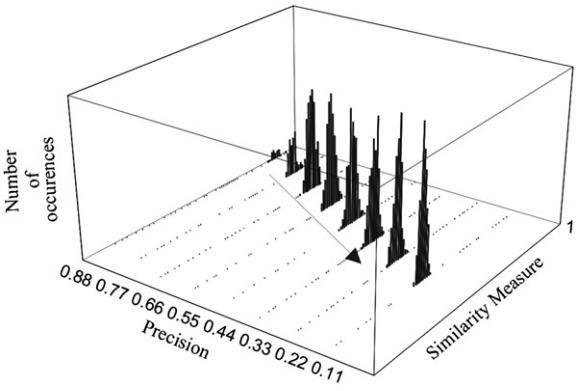

Fig. 5. Illustration of the monotonicity principle.

$p(\sigma^*) = 1$. For any other matching σ', $p(\sigma') < p(\sigma^*)$. Therefore, if $p(\sigma') < p(\sigma^*)$ then from monotonicity $\Omega(\sigma') < \Omega(\sigma^*)$. All one has to do then is to devise a method for finding a matching σ^* that maximizes Ω.[11]

Fig. 5 provides an illustration of the monotonicity principle using a matching of a simplified version of the Web forms in "Absolute Agency" with "Adult Singles" Web sites, both taken from the dating and matchmaking domain. Both schemata have nine attributes, all of which are matched under the exact matching. Given a set of matchings, each value on the x-axis represents a class of schema matchings with a different precision. The z-axis represents the similarity measure. Finally, the y-axis stands for the number of schema matchings from a given precision class and with a given similarity measure.

Fig. 5 provides two main insights. First, the similarity measures of matchings within each schema matching class form a "bell" shape, centered around a specific similarity measure. Such a behavior indicates a certain level of robustness of a schema matcher, assigning close similarity measures to matchings within each class. Second, the "tails" of the bell shapes overlap. Therefore, a schema matching from a class of a lower precision may receive a higher similarity measure than a matching from a class of a higher precision. This, of course, contradicts the monotonicity definition. However, the first observation serves as a motivation for a definition of a statistical monotonicity, first introduced in [17]:

Definition 1 (*Statistical monotonicity*). Let $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ be a set of matchings over schemata S_1 and S_2 with n_1 and n_2 attributes, respectively, and define $n = \max(n_1, n_2)$. Let $\Sigma_1, \Sigma_2, \ldots, \Sigma_{n+1}$ be subsets of Σ such that for all $1 \leq i \leq n+1$, $\sigma \in \Sigma_i$ iff $(i-1)/n \leq p(\sigma) < i/n$. We define M_i to be a random variable, representing the similarity measure of a randomly chosen matching from Σ_i. Σ is *statistically monotonic* if the following inequality holds for

[11] In [17], where the monotonicity principle was originally introduced, it was shown that while such a method works well for fuzzy aggregators (e.g., weighted average) it does not work for t-norms such as min.

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

any $1 \le i < j \le n+1$:

$$\overline{\Omega}(M_i) < \overline{\Omega}(M_j) \quad (5)$$

where $\overline{\Omega}(M)$ stands for the expected value of M.

References

[1] C. Batini, M. Lenzerini, S. Navathe, A comparative analysis of methodologies for database schema integration, ACM Computing Surveys 18 (4) (1986) 323–364.
[2] M. Lenzerini, Data integration: a theoretical perspective, in: Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 2002, pp. 233–246.
[3] P. Bernstein, S. Melnik, Meta data management, in: Proceedings of the IEEE CS International Conference on Data Engineering, IEEE Computer Society, Boston, MA, USA, 2004.
[4] A. Sheth, J. Larson, Federated database systems for managing distributed, heterogeneous, and autonomous databases, ACM Computing Surveys 22 (3) (1990) 183–236.
[5] E. Rahm, P. Bernstein, A survey of approaches to automatic schema matching, VLDB Journal 10 (4) (2001) 334–350.
[6] P. Shvaiko, J. Euzenat, A survey of schema-based matching approaches, Journal of Data Semantics 4 (2005) 146–171.
[7] H. Do, E. Rahm, COMA—a system for flexible combination of schema matching approaches, in: Proceedings of the International Conference on Very Large Data Bases (VLDB), 2002, pp. 610–621.
[8] J. Madhavan, P. Bernstein, E. Rahm, Generic schema matching with Cupid, in: Proceedings of the International Conference on Very Large Data Bases (VLDB), Rome, Italy, 2001, pp. 49–58.
[9] A. Gal, G. Modica, H. Jamil, A. Eyal, Automatic ontology matching using application semantics, AI Magazine 26 (1) (2005) 21–32.
[10] J. Berlin, A. Motro, Autoplex: automated discovery of content for virtual databases, in: C. Batini, F. Giunchiglia, P. Giorgini (Eds.), Cooperative Information Systems, 9th International Conference, CoopIS 2001, September 5–7, 2001, Proceedings, Lecture Notes in Computer Science, vol. 2172, Springer, Trento, Italy, 2001, pp. 108–122.
[11] S. Melnik, E. Rahm, P. Bernstein, Rondo: a programming platform for generic model management, in: Proceedings of the ACM-SIGMOD Conference on Management of Data (SIGMOD), ACM Press, San Diego, CA, 2003, pp. 193–204.
[12] R. Miller, M. Hernàndez, L. Haas, L.-L. Yan, C. Ho, R. Fagin, L. Popa, The Clio project: managing heterogeneity, SIGMOD Record 30 (1) (2001) 78–83.
[13] A. Doan, J. Madhavan, P. Domingos, A. Halevy, Learning to map between ontologies on the semantic web, in: Proceedings of the 11th International Conference on World Wide Web, ACM Press, Honolulu, Hawaii, USA, 2002, pp. 662–673.
[14] S. Bergamaschi, S. Castano, M. Vincini, D. Beneventano, Semantic integration of heterogeneous information sources, Data & Knowledge Engineering 36 (3) (2001).
[15] S. Castano, V.D. Antonellis, S.D.C. di Vimercati, Global viewing of heterogeneous data sources, IEEE Transactions on Knowledge and Data Engineering 13 (2) (2001) 277–297.
[16] K. Saleem, Z. Bellahsene, E. Hunt, Performance oriented schema matching, in: 18th International Conference on Database and Expert Systems Applications (DEXA 2007), Springer, Regensburg, Germany, 2007, pp. 844–853.
[17] A. Gal, A. Anaby-Tavor, A. Trombetta, D. Montesi, A framework for modeling and evaluating automatic semantic reconciliation, VLDB Journal 14 (1) (2005) 50–67.
[18] D. Embley, D. Jackman, L. Xu, Attribute match discovery in information integration: exploiting multiple facets of metadata, Journal of Brazilian Computing Society 8 (2) (2002) 32–43.
[19] P. Mork, A. Rosenthal, L. Seligman, J. Korb, K. Samuel, Integration workbench: Integrating schema integration tools, in: Proceedings of the 22nd International Conference on Data Engineering Workshops, ICDE 2006, 3–7 April 2006, Atlanta, GA, USA, 2006, p. 3.
[20] Y. Lee, M. Sayyadian, A. Doan, A. Rosenthal, eTuner: tuning schema matching software using synthetic scenarios, VLDB Journal 16 (1) (2007) 97–122.
[21] R. Schapire, The strength of weak learnability, Machine Learning 5 (1990) 197–227 URL: citeseer.ist.psu.edu/schapire90strength.html.
[22] Y. Freund, R. Schapire, A short introduction to boosting, 1999. URL: iteseer.ist.psu.edu/freund99short.html.
[23] S. Melnik, H. Garcia-Molina, E. Rahm, Similarity flooding: a versatile graph matching algorithm and its application to schema matching, in: Proceedings of the IEEE CS International Conference on Data Engineering, 2002, pp. 117–140.
[24] A. Gal, Managing uncertainty in schema matching with top-k schema mappings, Journal of Data Semantics 6 (2006) 90–114.
[25] M. Benerecetti, P. Bouquet, S. Zanobini, Soundness of schema matching methods, in: Proceedings of ESWC 2005, 2005, pp. 211–225.
[26] J. Euzenat, P. Shvaiko, Ontology Matching, Springer-Verlag, Heidelberg (DE), 2007.
[27] S. Melnik, Generic Model Management: Concepts and Algorithms, Springer-Verlag, Berlin, Heidelberg, New York, 2004.
[28] D. Barbosa, J. Freire, A. Mendelzon, Designing information-preserving mapping schemes for xml, in: Proceedings of the International Conference on Very Large Data Bases (VLDB), 2005, pp. 109–120.
[29] P. Bohannon, W. Fan, M. Flaster, P. Narayan, Information preserving xml schema embedding, in: Proceedings of the International Conference on Very Large Data Bases (VLDB), 2005, pp. 85–96.
[30] R. Fagin, Inverting schema mappings, in: Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 2006, pp. 50–59.
[31] R. Fagin, P. Kolaitis, L. Popa, W. Tan, Quasi-inverses of schema mappings, in: Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 2007, pp. 123–132.
[32] S. Alagic, P. Bernstein, A model theory for generic schema management, in: Database Programming Languages, 8th International Workshop, DBPL 2001, Frascati, Italy, September 8–10, 2001, pp. 228–246.
[33] J. Madhavan, P. Bernstein, P. Domingos, A. Halevy, Representing and reasoning about mappings between domain models, in: Proceedings of the 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI), 2002, pp. 80–86.
[34] S.M. Lane, Categories for the Working Mathematician, second ed., Springer, 1998.
[35] F. Giunchiglia, P. Shvaiko, M. Yatskevich, Semantic schema matching, in: Proceedings of the 10th International Conference on Cooperative Information Systems (CoopIS 2005), Agia Napa, Cyprus, 2005, pp. 347–365.
[36] C. Domshlak, A. Gal, H. Roitman, Rank aggregation for automatic schema matching, IEEE Transactions on Knowledge and Data Engineering 19 (4) (2007) 538–553.
[37] X. Dong, A. Halevy, C. Yu, Data integration with uncertainty, in: Proceedings of the International Conference on Very Large Data Bases (VLDB), 2007, pp. 687–698.
[38] E. Mena, V. Kashayap, A. Illarramendi, A. Sheth, Imprecise answers in distributed environments: estimation of information loss for multi-ontological based query processing, International Journal of Cooperative Information Systems 9 (4) (2000) 403–425.
[39] H. Nottelmann, U. Straccia, Information retrieval and machine learning for probabilistic schema matching, Information Processing and Management 43 (3) (2007) 552–576.
[40] F. Duchateau, Z. Bellahsene, R. Coletta, A flexible approach for planning schema matching algorithms, in: Proceedings of the 13th International Conference on Cooperative Information Systems (CoopIS 2008), 2008, pp. 249–264.
[41] M. Ehrig, S. Staab, Y. Sure, Bootstrapping ontology alignment methods with apfel, in: The Semantic Web—ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6–10, 2005, pp. 186–200.
[42] A. Doan, P. Domingos, A. Halevy, Reconciling schemas of disparate data sources: a machine-learning approach, in: W.G. Aref (Ed.), Proceedings of the ACM-SIGMOD Conference on Management of Data (SIGMOD), ACM Press, Santa Barbara, CA, 2001, pp. 509–520.
[43] A. Algergawy, R. Nayak, G. Saake, Xml schema element similarity measures: a schema matching context, in: Proceedings of the 8th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2009), 2009, pp. 1246–1253.
[44] R. Schapire, The boosting approach to machine learning: an overview, in: MSRI Workshop on Nonlinear Estimation and Classification, Berkeley, CA, 2001.
[45] W.-S. Li, C. Clifton, SEMINT: a tool for identifying attribute correspondences in heterogeneous databases using neural networks, Data & Knowledge Engineering 33 (1) (2000) 49–84.
[46] L. Xu, D. Embley, A composite approach to automating direct and indirect schema mappings, Information Systems 31 (8) (2006) 697–886.
[47] A. Marie, A. Gal, Boosting Schema Matchers, Springer, Monterey, Mexico, 2008, pp. 283–300.

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003

[48] A. Vinson, C. Heuser, A. da Silva, E. de Moura, An approach to xml path matching, in: WIDM '07: Proceedings of the 9th Annual ACM International Workshop on Web Information and Data Management, ACM, New York, NY, USA, 2007, pp. 17–24 doi:http://doi.acm.org/10.1145/1316902.1316906.

[49] A. Gal, P. Shvaiko, Advances in ontology matching, in: T.S. Dillon, E. Chang, R. Meersman, K. Sycara (Eds.), Web Services and Applied Semantic Web, Springer, Berlin/Heidelberg, 2009, pp. 176–198.

[50] B. He, K.C.-C. Chang, Statistical schema matching across Web query interfaces, in: Proceedings of the ACM-SIGMOD Conference on Management of Data (SIGMOD), ACM Press, San Diego, CA, USA, 2003, pp. 217–228.

[51] W. Su, J. Wang, F. Lochovsky, A holistic schema matching for Web query interfaces, in: Advances in Database Technology—EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26–31, 2006, pp. 77–94.

[52] A. Marie, A. Gal, On the stable marriage of maximumweight royal couples, in: Proceedings of AAAI Workshop on Information Integration on the Web (IIWeb'07), Vancouver, BC, Canada, 2007.

[53] W. Su, Domain-based data integration for web databases, Ph.D. Thesis, Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, December 2007.

[54] R. Miller, L. Haas, M. Hernández, Schema mapping as query discovery, in: A.E. Abbadi, M. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, K.-Y. Whang (Eds.), Proceedings of the International Conference on Very Large Data Bases (VLDB), Morgan Kaufmann, Cairo, Egypt, 2000, pp. 77–88.

[55] A. Marie, A. Gal, Managing uncertainty in schema matcher ensembles, in: H. Prade, V. Subrahmanian (Eds.), 1st International Conference on Scalable Uncertainty Management, SUM 2007, Springer, Washington, DC, USA, 2007, pp. 60–73.

[56] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, P. Domingos, imap: Discovering complex mappings between database schemas, in: Proceedings of the ACM-SIGMOD Conference on Management of Data (SIGMOD), 2004, pp. 383–394.

[57] P. Bernstein, S. Melnik, M. Petropoulos, C. Quix, Industrial-strength schema matching, SIGMOD Record 33 (4) (2004) 38–43.

[58] Y. Freund, R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, Journal of Computer and System Sciences 55 (1) (1997) 119–139.

[59] R. Schapire, Using output codes to boost multiclass learning problems, in: Proceedings of the Fourteenth International Conference on Machine Learning, 1997, pp. 313–321.

[60] G. Ridgeway, D. Madigan, T. Richardson, Boosting methodology for regression problems, in: Proceedings of the International Workshop on AI and Statistics, 1999, pp. 152–161.

Please cite this article as: A. Gal, T. Sagi, Tuning the ensemble selection process of schema matchers, Informat. Systems (2010), doi:10.1016/j.is.2010.04.003