Rank Aggregation for Automatic Schema Matching

Carmel Domshlak, Avigdor Gal, *Senior Member*, *IEEE*, and Haggai Roitman

Abstract—Schema matching is a basic operation of data integration, and several tools for automating it have been proposed and evaluated in the database community. Research in this area reveals that there is no single schema matcher that is guaranteed to succeed in finding a good mapping for all possible domains and, thus, an ensemble of schema matchers should be considered. In this paper, we introduce *schema metamatching*, a general framework for composing an arbitrary ensemble of schema matchers and generating a list of best ranked schema mappings. Informally, schema metamatching stands for computing a "consensus" ranking of alternative mappings between two schemata, given the "individual" graded rankings provided by several schema matchers. We introduce several algorithms for this problem, varying from adaptations of some standard techniques for general quantitative rank aggregation to novel techniques specific to the problem of schema matching, and to combinations of both. We provide a formal analysis of the applicability and relative performance of these algorithms and evaluate them empirically on a set of real-world schemata.

Index Terms—Database integration, schema matching, rank aggregation.

1 Introduction

Schema matching is the task of matching concepts describing the meaning of data in various data sources (for example, database schemata, XML DTDs, HTML form tags, and so forth). As such, schema matching is recognized to be one of the basic operations required by the process of data integration [3]. The area of data integration has a rich body of literature on schema matching, summarized in a few surveys [7], [41] and special issues [11], [39]. Examples of algorithmic tools providing means for schema matching are COMA [8], Cupid [31], OntoBuilder [23], Autoplex [1], Similarity Flooding [34], Clio [36], Glue [10], to name a few. Foundational principles of schema matching are also discussed in [3], [22], [30], and [32].

A typical classification of schema-matching tasks relates to the amount of automatic processing required for achieving a task. Due to its cognitive complexity, schema matching has been traditionally performed by human experts [5], [28]. For obvious reasons, manual concept reconciliation in large-scale and/or dynamic environments (with or without computer-aided tools) is inefficient and at times close to impossible. Introduction of the Semantic Web vision [2] and shifts toward machine-understandable Web resources and Web services have made even clearer the vital need for automating schema matching. The move from manual to semiautomatic schema matching has been justified in the literature using arguments of scalability (especially for matching between large schemata [26]) and by the need to speed up the matching process. The motivation for moving to *fully automatic* (that is, unsupervised) schema matching stems from the possible absence of a human expert in the decision process. In particular, such situations characterize numerous emerging applications triggered by the vision of the Semantic Web and machine-understandable Web resources [2], [43]. To illustrate this further, consider the recent Web service challenge competition held in 2006.[1] The teams at this competition were required to discover and compose Web services in a completely unsupervised manner. Although the first competitions are still based on exact string matching of parameters, the next competitions have been declared to involve issues of heterogeneous and constrained schema matching.

Attempting to address the schema-matching problem, numerous heuristics (schema matchers or simply matchers hereafter) have been proposed and evaluated in the database community (for example, see [1], [4], [9], [18], [19], [23], [25], [34], and [42]). However, choosing among this variety of tools 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, recent 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 [22]. In fact, due to effectively unlimited heterogeneity and ambiguity of data description, it seems unavoidable that optimal mappings for many pairs of schemata will be considered as "best mappings" by none of the existing schema matchers.

Striving to increase robustness in the face of the biases and shortcomings of individual matchers, several tools have

• *The authors are with the Faculty of Industrial Engineering and Management, Technion-Israel Institute of Technology, Technion City, Haifa 32000, Israel.*
E-mail: {dcarmel, avigal}@ie.technion.ac.il, haggair@tx.technion.ac.il.

Manuscript received 10 Apr. 2006; revised 13 Aug. 2006; accepted 2 Oct. 2006; published online 24 Jan. 2007.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0168-0406.
Digital Object Identifier no. 10.1109/TKDE.2007.1010.

1. http://insel.flp.cs.tu-berlin.de/wsc06/.

enabled combining principles by which different schema matchers judge the similarity between concepts. The idea is appealing since an ensemble of complementary matchers can potentially compensate for the weaknesses of each other. Indeed, several studies report on encouraging results when using schema matcher ensembles (for example, see [8], [13], [23], [31], and [38]). Given that, the first goal of our work is to formally analyze the applicability and limitations of prior works on ensembling schema matchers and provide a more general ensemble framework that overcomes these limitations.

However, even having a good ensemble of complementary schema matchers cannot guarantee that an optimal mapping between the schemata (for example, a mapping that would have been generated by a human expert) will always be identified as the top choice of the ensemble. To address such situations to the largest degree possible, one can adopt the approach in which K (and not just one) top-ranked schema mappings are generated and examined[2] either iteratively or simultaneously [22], [21], [27], [29]. Our second goal is thus to connect between the ensemble approach and the top-K approach, increasing the robustness of the schema-matching process by enjoying the best of these two worlds.

To achieve our goals, here, we introduce a generic computational framework, *schema metamatching*, for computing the top-K prefix of a "consensus" ranking of alternative mappings between two schemata, given the graded valid mappings of schema attributes provided "individually" by the members of an ensemble. A valid mapping in this case is a mapping that satisfied matching constraints (for example, cardinality constraints) specific to the application.[3]

Our starting point is based on rank aggregation techniques developed in the areas of Web search and database middleware [12], [17]. First, we show that the Threshold algorithm, originally proposed in the context of database middleware [17], can be applied to our problem almost as is. Unfortunately, as we show, computing the top-K mappings for schema metamatching using the Threshold algorithm may require time exponential in the size of the matched schemata. Since in the original context of domain-independent rank aggregation the Threshold algorithm has been shown to be optimal in a strong sense, we proceed with developing techniques that exploit the specifics of the schema-matching problem. For a certain wide class of problems, we present a simple algorithm, the Matrix-Direct (MD) algorithm whose time complexity is polynomial in the size of the matched schemata and the required K. Subsequently, we present the Matrix-Direct-with-Bounding (MDB) algorithm, which draws upon both the MD and Threshold algorithms, addressing matching scenarios where the MD algorithm is inapplicable. We show that the Threshold and MDB algorithms are (complexity-wise) mutually undominated—that is, there exist problem instances in which one algorithm performs dramatically better than the other. To enjoy the best of both worlds, and even to improve upon them, we introduce the Cross-Threshold algorithm, a hybrid version of these two algorithms, based on their in-parallel, *mutually enhancing* execution. Our analysis shows the complexity and effectiveness of adopting this hybrid algorithm.

We support our formal analysis with experiments on a real-world data feed. In these experiments, we test the relative performance of the Threshold, MDB, and Cross-Threshold algorithms on numerous sets of various schema matchers. Our empirical findings support the formal results, in particular showing that the CrossThreshold algorithm dominates both Threshold and MDB algorithms.

It is important to note that the schema metamatching framework does *not* define the "consensus" ranking, but only aims at its efficient generation. The "consensus" ranking is defined by the actual choice of ensemble, and this choice is orthogonal to our work. In particular, the relative effectiveness of the "consensus" ranking is *independent* of the choice of the schema metamatching algorithm. Therefore, our formal and empirical analyses are devoted solely to the correctness of the algorithms and their comparative performance.

To summarize, the main contributions of this paper are listed as follows:

- Introduction of schema metamatching, a generic computational framework for combining an ensemble of arbitrary schema matchers for identifying the top-K schema mappings.
- Provision and formal analysis of four algorithms for schema metamatching. In particular, we analyze an existing algorithm (Threshold) for general rank aggregation adapted to our domain and compare its applicability and performance with a generalized version of the COMA [8] approach (MD). We next develop and study two novel *generically applicable* algorithms (MDB and CrossThreshold). In particular, we show that the CrossThreshold algorithm combines the benefits of all the other algorithms, providing the generically most efficient solution to the schema metamatching problem.
- Comparative quantitative evaluation of the algorithms that empirically supports the practical relevance of our formal results.

The rest of the paper is organized as follows: In Section 2, we provide some basic formalism and notation, and introduce the schema metamatching framework. In Section 3, we discuss two basic algorithms that can be used to implement schema metamatching, namely, the Threshold and MD algorithms. In Section 4, we introduce the MDB algorithm and compare it with the Threshold algorithm. In Section 5, we introduce the CrossThreshold algorithm, a hybrid version of the Threshold and MDB algorithms, and discuss its properties. The corresponding experiments and empirical analysis are presented in Section 6. We conclude in Section 7.

2. Automatic examination of alternative schema mappings is beyond the scope of this paper; it is typically tool dependent and may involve analysis of query variations [35], Web server error messages, and so forth.

3. Alternatively, the ensemble members can first provide rankings of only the *attribute-level* mappings, while ignoring the application constraints posed on the schema matching process. It is apparent that such an approach would significantly reduce the complexity of individual rankings. However, these rankings then need to be combined into a "consensus" ranking of valid *schema* mappings. To the best of our knowledge, there is no evidence in the literature that such an approach can provide, at a low complexity cost, a semantically justified "consensus" ranking over the schema mappings while respecting schema-level matching constraints.

2 Formalism, Notation, and Problem Statement

We begin by introducing some formalism and notation essential for defining the schema metamatching problem.

Let *schema* S be a finite set of some *attributes*. We put no particular limitations on the notion of schema attributes; attributes can be both simple and compound, compound attributes need not necessarily be disjoint, and so forth. For any schemata pair S and S', let $\mathcal{S} = S \times S'$ be the set of all possible *attribute mappings* between S and S', and let the power-set $\Sigma = 2^{\mathcal{S}}$ be the set of all possible *schema mappings* between this pair of schemata. Let $\Gamma : \Sigma \rightarrow \{0, 1\}$ be a Boolean function that captures the application-specific constraints on schema mappings, for example, cardinality and interattribute mapping constraints.[4] Given such a constraint specification Γ, the set of all *valid* schema mappings in Σ is given by $\Sigma_\Gamma = \{\sigma \in \Sigma \,|; \Gamma(\sigma) = 1\}$. A *schema matcher* A takes as its input a schemata pair S, S', as well as a constraint specification Γ, and provides us with an ordering $\succeq_A$ over Σ_Γ. For schema mappings σ, $\sigma' \in \Sigma_\Gamma$, $\sigma \succeq_A \sigma'$ means that σ is estimated by A to be as good as σ'. It is worth noting that such an ordering may be given either implicitly or explicitly.

Although various schema-matching models have been proposed, many of them follow a similar two-step pattern [8] that we adopt here. In the first step, each attribute mapping in $\mathcal{S}$ is automatically assigned with a real-valued degree of similarity. If S and S' are of arity n and n', respectively, then this step results in an $n \times n'$ *similarity matrix* $M^{(A)}$, where $M^{(A)}_{i,j}$ represents the degree of similarity between the ith attribute of S and the jth attribute of S', as assigned by A. Various schema matchers differ mainly in the measures of similarity they employ and, thus, yield different similarity matrices. These similarity measures can be arbitrarily complex and may use various techniques for name matching, domain matching, structure matching (such as XML hierarchical representation), and semantic matching.

In the second step, the similarity information in $M^{(A)}$ is used to quantify the quality of different schema mappings σ in Σ_Γ using some real-valued *local aggregation function* (or *l-aggregator*, for short):

$$f^{(A)}\left(\sigma, M^{(A)}\right) = f^{(A)}\left(M^{(A)}_{1,\sigma(1)}, \ldots, M^{(A)}_{n,\sigma(n)}\right)$$

that is, a function that aggregates the degrees of similarity associated with the individual attribute mappings forming the schema mapping σ. The ordering $\succeq_A$ on Σ_Γ is then

$$\sigma \succeq_A \sigma' \quad \Leftrightarrow \quad f^{(A)}\left(\sigma, M^{(A)}\right) \geq f^{(A)}\left(\sigma', M^{(A)}\right)$$

for each σ, $\sigma' \in \Sigma_\Gamma$. A popular choice of l-aggregator is the sum (or average) of attribute mapping degrees of similarity (for example, see [8], [23], and [33]), but other l-aggregators have been found appealing as well (for example, the *Dice* l-aggregator suggested in [8], threshold-based aggregators [37], and so forth). Without loss of generality, in what follows, we assume that f is computable in time linear in n and n'. However, at least technically, nothing prevents us from using more sophisticated (and possibly more computation-intense) l-aggregators.

Having defined the ordering $\succeq_A$ over Σ_Γ, the schema matcher A can now provide answers to various queries. The most common query these days stands for retrieving a top-1 mapping:

$$\sigma^1 = \arg\max_\sigma \{f(\sigma, M) \mid \sigma \in \Sigma_\Gamma\}$$

(possibly) along with its quality estimation $f^{(A)}(\sigma^1, M^A)$. In the top-K approach, this query is generalized to retrieving a top-ith mapping

$$\sigma^i = \arg\max_\sigma \left\{f^{(A)}(\sigma, M^{(A)}) \mid \sigma \in \Sigma_\Gamma \setminus \{\sigma^1, \cdots, \sigma^{i-1}\}\right\} \quad (1)$$

annotated with $f^{(A)}(\sigma^i, M^A)$. In what follows, we refer to this query as $\mathbf{q\text{-}top}(i)$. In addition, the schema matcher can be queried for the estimate $f^{(A)}(\sigma, M^{(A)})$ for an *arbitrary* mapping $\sigma \in \Sigma_\Gamma$ and, here, we denote such a query by $\mathbf{q\text{-}estim}(\sigma)$. Clearly, the time and space complexity of answering these queries depend on both the structure of Γ and the l-aggregator $f^{(A)}$. On the positive side, however, in many natural settings, answering these queries can be efficient. For instance, when $f^{(A)}$ is equivalent to sum, and Γ is devoted to enforce one-to-one cardinality constraint, then the time complexity of retrieving σ^i is[5] $O(i\eta^4)$, where $\eta = \max\{n, n'\}$ [21], [24], [40], and providing the estimate $f^{(A)}(\sigma, M^{(A)})$ can be done in $O(\eta)$.

Now, let us consider an ensemble of m schema matchers $A_1, \ldots, A_m$ utilizing (possibly different) local aggregators $f^{(1)}, \ldots, f^{(m)}$, respectively. Given two schemata S and S' as before, these matchers produce an $m \times n \times n'$ similarity cube of $n \times n'$ similarity matrices $M^{(1)}, \ldots, M^{(m)}$. Such an ensemble of schema matchers $A_1, \ldots, A_m$ is used to generate a "consensus" ordering $\succeq$ over Σ_Γ from the individual orderings $\succeq_1, \ldots, \succeq_m$. This ordering aggregation is performed via aggregating the weights each A_i provides to the schema mappings in Σ_Γ. In turn, weight aggregation can always be modeled using a real-valued *global aggregation function* (or *g-aggregator*, for short) $F\ (f^{(1)}(\sigma, M^{(1)}), \cdots, f^{(m)}(\sigma, M^{(m)}))$ [8], [23]. In what follows, by $\langle\vec{f}, F\rangle$, we denote the set of l-aggregators and g-aggregator in use, respectively. Likewise, we use the notation

$$\langle\vec{f}, F\rangle(\sigma) \equiv F\left(f^{(1)}(\sigma, M^{(1)}), \cdots, f^{(m)}(\sigma, M^{(m)})\right)$$

for the aggregated weight provided by $A_1, \ldots, A_m$ with $\langle\vec{f}, F\rangle$ to the mapping σ. The aggregated ordering $\succeq$ on Σ_Γ is then

$$\sigma \succeq \sigma' \quad \Leftrightarrow \quad \langle\vec{f}, F\rangle(\sigma) \geq \langle\vec{f}, F\rangle(\sigma')$$

for each $\sigma, \sigma' \in \Sigma_\Gamma$. For instance, many g-aggregators proposed in the literature can be generalized as

4. We refrain from an in-depth analysis of cardinality and other interattribute mapping constraints. The interested reader is referred to [6], [10], [20], and [44].

5. Given $\sigma^1, \ldots, \sigma^{i-1}$, the time complexity of retrieving σ^i is $O(\eta^3)$ [21], [24], [40].

$$F\Big(f^{(1)}(\sigma, M^{(1)}), \cdots, f^{(m)}(\sigma, M^{(m)})\Big) = \frac{\lambda}{m}\sum_{l=1}^{m} k_l f^{(l)}(\sigma, M^{(l)}), \tag{2}$$

where (2) can be interpreted as a (weighted) sum (with $\lambda = m$) or a (weighted) average (with $\lambda = 1$) of the local rankings, where k_l is the arbitrary weighting parameters. It is important to note that the choice of g-aggregator is unavoidably ensemble dependent and, thus, here, we consider it as a *given* property of the ensemble.

Having formalized individual schema matchers and their ensembles as above, we define the *schema metamatching* problem to be that of generating the top-K valid mappings between S and S' with respect to an ensemble of schema matchers $A_1, \ldots, A_m$, their respective l-aggregators $f^{(1)}, \ldots, f^{(m)}$, and the ensemble's g-aggregator F. Formally, given S, S', Γ, and $K \geq 1$, our task is to generate $\{\sigma^1, \ldots, \sigma^K\} \subseteq \Sigma_\Gamma$, where the ith best mapping σ^i is inductively defined as

$$\sigma^i = \arg\max_{\sigma}\Big\{\langle\vec{f}, F\rangle(\sigma) \mid \sigma \in \Sigma_\Gamma \setminus \{\sigma^1, \cdots, \sigma^{i-1}\}\Big\} \tag{3}$$

similar to (1) for the basic case of $m = 1$.

3 Rank Aggregation for Schema Matching

Having formalized the problem of schema metamatching, we now proceed with exploring it from the computational standpoint. To stress some of the computational issues involved, consider a straightforward procedure for rank aggregation, where each judge (a schema matcher in our case) explicitly ranks the entire universe of alternatives, associating each alternative with a certain level of "goodness." These individual grades are then combined (this or another way) into a grading underlying the "consensus" ranking, and we are provided with the top-K elements of this aggregated ranked list. Unfortunately, in the case of schema matching, the size of the universe of alternatives makes this straightforward approach unrealistic: Given two schemata of n attributes each, there are already $n!$ alternative 1:1 mappings between them, and this number is even larger for less constrained settings. Therefore, any realistic method for schema metamatching has to either consider individual rankings represented implicitly in some compact form or carefully query the judges about the mappings while limiting the number and complexity of these queries to the extent possible.[6]

In the remainder of this paper, we focus on the algorithmic aspects of solving this problem. Before we begin discussing various algorithms, it is worth observing that a naïve approach of 1) generating m top-K lists of mappings with respect to $A_1, \ldots, A_m$ using the q-top queries and 2) subsequently aggregating these lists using F is not sound. To illustrate this, consider the top-1 mapping σ^1. First, as strange as it may seem, σ^1 may appear in none of the m individual top-K lists and, thus, will definitely not appear in an aggregated list of any length. Such a case may occur whenever σ^1 is not one of the top-K mappings of any of $A_1, \ldots, A_m$, yet, these experts are so in odds with each other that the common consensus becomes a convenient mediocre mapping. Second, even if σ^1 appears in some, or even most, individual top-K lists, it can be improperly ranked in step 2), and possibly even discarded from the aggregated top-K list. This can occur if the relative aggregated ranking of σ^1 is significantly affected by the scores it gets from the experts that individually ranked it lower than top-K.

6. Note that in contrast to the case of Web meta-search [12], our judges *are* ready to answer any query about mapping rankings.

```
1) Starting with $i = 1$, do incremental (on growing $i$) parallel querying of $A_1, \ldots, A_m$ with q-top($i$). This querying is unbounded, corresponding to a sorted access in parallel to each of the $m$ rankings of alternative valid mappings $\Sigma_\Gamma$.
   a) As a mapping $\sigma$ is introduced by one of the matchers, obtain the remaining $f^{(1)}(\sigma, M^{(1)}), \cdots, f^{(m)}(\sigma, M^{(m)})$ by querying the other matchers with q-estim($\sigma$), and compute the aggregated weight $\langle\vec{f}, F\rangle(\sigma)$. If this weight is one of the $K$ highest we have seen so far, then remember $\sigma$.
   b) For $1 \leq l \leq m$, let $\sigma_l$ be the mapping returned by the *last* q-top query to $A^{(l)}$. Define the threshold value $\tau_{TA} = F\Big(f^{(1)}(\sigma_1, M^{(1)}), \cdots, f^{(m)}(\sigma_m, M^{(m)})\Big)$. If at least $K$ mappings have been seen whose weight is at least $\tau_{TA}$, then halt.
2) Let $Y$ be a set containing $K$ mappings with the highest grades seen so far. The output is then the graded set $\Big\{\big[\sigma, \langle\vec{f}, F\rangle(\sigma)\big] \mid \sigma \in Y\Big\}$.
```

Fig. 1. The TA algorithm, adopted for schema metamatching.

3.1 Adopting the Threshold Algorithm

The problem of optimal aggregation of several quantitatively ordered lists has been recently studied in the context of middleware for multimedia database systems [14], [15], [17], [16]. The most efficient general algorithm for this problem, called the Threshold (TA) algorithm, has been introduced in [17], and we begin by presenting this algorithm in terms of our problem in Fig. 1.

The intuition behind the TA algorithm is elegantly simple. For each schema matcher A_i, the algorithm utilizes q-top queries to generate as many mappings in a ranked order as needed. Assume that $K = 1$; that is, we are interested only in the best mapping. Assume further that we are at a stage in the algorithm where we have not seen any mapping σ whose aggregated weight $\langle\vec{f}, F\rangle(\sigma) \geq \tau_{TA}$, where τ_{TA} is determined in step 1b. If so, at this point we cannot be sure that the best mapping has already been seen, because the next mapping σ' generated by q-top could have aggregated weight $\langle\vec{f}, F\rangle(\sigma') \geq \tau_{TA}$. If this is the case, then clearly no mapping σ seen so far could be the best mapping, since $\langle \vec{f}, F\rangle(\sigma') > \langle \vec{f}, F\rangle(\sigma)$. Thus, it is safe to halt only when we see a mapping whose aggregated weight is at least τ_{TA}. Similarly, for $K > 1$, the stopping rule verifies a sufficient condition to ensure that the top-K mappings have been seen.

The only property required to ensure the completeness of the TA algorithm is monotonicity of the g-aggregator F in the following sense [17]: A function F is *monotonic* if, for every two mappings σ, σ', such that $f^{(l)}(\sigma, M^{(l)}) > f^{(l)}(\sigma', M^{(l)})$ holds for all $1 \leq l \leq m$, we have $\langle \vec{f}, F\rangle(\sigma) > \langle \vec{f}, F\rangle(\sigma')$. Since this requirement does not seem to induce any practical limitation, henceforth, we adopt this assumption of monotonicity for g-aggregators. Likewise, for ease of presentation and without loss of generality, we assume that F is computable in time linear in m.

Considering the time complexity of the TA algorithm while ignoring the specifics of the schema metamatching problem, it can be easily shown that this algorithm may have to access in a sorted manner as many as half of each sorted list (for example, see [17, Example 6.3]). Further, although we found no setting of schema-matching problem on which the TA algorithm performs that bad, the next theorem shows that in the context of schema metamatching, it may still have exponentially long runs.

Theorem 1. *The time complexity of schema metamatching using TA is* $\Omega((\frac{n}{2})!)$.

3.2 The MD Algorithm

Theorem 1 provides a strong motivation to seek more efficient alternatives to the TA algorithm. In [17], however, this algorithm is shown to be optimal in a strong sense of "instance optimality." For the formal definition of instance optimality, we refer the reader to [17]; roughly, for any set of data and any other rank aggregation algorithm A with the time complexity $Comp(A)$, instance optimality of the TA algorithm implies that its time complexity is of the order of that of A, that is, $Comp(\text{TA}) = O(Comp(A))$. Hence, at least at first view, it seems that using the TA algorithm for schema metamatching is the best we can do. However, below, we show that, for a certain class of aggregators $\langle \vec{f}, F\rangle$, an extremely simple technique exploiting specifics of the schema-matching problem provides a significantly better performance. Note that this does not contradict the instance optimality of the TA algorithm, as the latter is a generic algorithm independent of the actual grading mechanisms. In particular, the TA algorithm in our domain considers only the outputs of q-top queries and does not intervene in their processing. Hence, it is possible that one can devise algorithms outperforming TA by exploiting some properties of the specific problem domain at hand.

To begin with an example, let us consider l- and g-aggregators:

$$\forall l \in \{1, \ldots, m\} : f^{(l)}(\sigma, M^{(l)}) = \sum_{i=1}^{n} M^{(l)}_{i,\sigma(i)}$$
$$\langle \vec{f}, F\rangle(\sigma) = \sum_{l=1}^{m} k_l f^{(l)}(\sigma, M^{(l)}). \tag{4}$$

```
1) Given $A_1, \ldots, A_m$, (schematically) construct a new schema matcher $A^*$ with (a) similarity matrix $M^*$ such that, for $1 \leq i \leq n, 1 \leq j \leq n'$, $M^*_{i,j} = F(M^{(1)}_{i,j}, \cdots, M^{(m)}_{i,j})$, and (b) $l$-aggregator $f(\sigma, M^*)$.
2) Using queries $\mathsf{q\text{-}top}(1), \ldots, \mathsf{q\text{-}top}(K)$ to $A^*$, generate top-$K$ valid mappings with respect to $A^*$.
```

Fig. 2. The MD algorithm.

Observe that the summations in (4) can be distributed, resulting in

$$\langle f, F\rangle(\sigma) = \sum_{i=1}^{n}\sum_{l=1}^{m} k_l M^{(l)}_{i,\sigma(i)},$$

where the vector notation $\vec{f}$ is replaced with f to explicitly highlight the uniqueness of the l-aggregator in this case. That is, if the l-aggregator f and g-aggregator F happen to be as in (4), then using F for *local* weight aggregation and f for *global* weight aggregation will be *equivalent* to using f and F in their original roles. In other words, in case of (4), we have $\langle f, F\rangle(\sigma) = \langle F, f\rangle(\sigma)$ for any mapping σ between any pair of schemata S and S'. The special case of (4) can be generalized as follows:

Definition 1. *Given a set of similarity matrices* $M^{(1)}, \ldots, M^{(m)}$ *over a pair of schemata* S, S', *and a pair of l-aggregator* f *and g-aggregator* F, *we say that* f *and* F commute *on* $M^{(1)}, \ldots, M^{(m)}$ *if and only if, for every mapping* σ *between* S *and* S', *we have*

$$\langle f, F\rangle(\sigma) = \langle F, f\rangle(\sigma). \tag{5}$$

Likewise, if f *and* F *commute on all possible sets of similarity matrices, then we say that* f *and* F *are strongly commutative.*

For instance, the aggregators f and F as in (4) are strongly commutative. To illustrate commutativity in the absence of strong commutativity, consider a pair of aggregators corresponding to *min* and *product*, respectively. Although these two aggregators are clearly not strongly commutative, they do commute, for instance, on any set of Boolean similarity matrices.

The commutativity between the l- and g-aggregators leads to an extremely efficient algorithm for schema metamatching. Specifically, in Fig. 2, we present the MD algorithm, generalizing the applicability of the composite method of COMA [8] to any schema metamatching problem in which 1) all the judges use the same l-aggregator and 2) the l- and g-aggregators commute on the given set of similarity matrices. The correctness and time complexity of the MD algorithm are stated by Theorem 2.

Theorem 2. *Given a set of schema matchers* $A_1, \ldots, A_m$ *and a pair of local and global aggregators* $\langle f, F\rangle$, *let* M^* *be a matrix defined as* $M^*_{i,j} = F(M^{(1)}_{i,j}, \cdots, M^{(m)}_{i,j})$ *for all* $1 \leq i \leq n$, $1 \leq j \leq n'$. *If* f *and* F *commute on the similarity matrices* $M^{(1)}, \ldots, M^{(m)}$, *then the MD algorithm correctly finds the top-K valid mappings with respect to the aggregated ranking in time* $O(\eta^2 m + \Phi)$, *where* Φ *is the combined time complexity of iteratively executed queries* $\mathbf{q\text{-}top}(1), \ldots, \mathbf{q\text{-}top}(K)$ *over* M^*.

4 MD Algorithm with Bounding

Reading so far, it seems natural to conclude that the schema metamatching problems satisfying the conditions of Theorem 2 should be processed using MD, whereas all other problems should be processed using TA (that is, we are back to an instance optimal algorithm for general quantitative rank aggregation). However, below, we show that, although the former conclusion is sound, the latter is not necessarily so.

Definition 2. *Consider a set of similarity matrices $M^{(1)}, \ldots, M^{(m)}$ over a pair of schemata S, S' and two sets of l- and g-aggregators $\langle \vec{f}, F \rangle$ and $\langle \vec{f}', F' \rangle$. We say that $\langle \vec{f}', F' \rangle$* dominates *$\langle \vec{f}, F \rangle$ on $M^{(1)}, \ldots, M^{(m)}$ (denoted as $\langle \vec{f}', F' \rangle \succ \langle \vec{f}, F \rangle$) if, for every mapping σ from S to S', we have*

$$\langle \vec{f}', F' \rangle(\sigma) \geq \langle \vec{f}, F \rangle(\sigma). \tag{6}$$

Likewise, if (6) holds for all possible sets of similarity matrices, then we say that $\langle \vec{f}', F' \rangle$ strongly dominates *$\langle \vec{f}, F \rangle$.*

Consider a schema metamatching problem defined by a set of similarity matrices $M^{(1)}, \ldots, M^{(m)}$ and a set of l- and g-aggregators $\langle \vec{f}, F \rangle$ that *do not* commute on $M^{(1)}, \ldots, M^{(m)}$. Suppose that there exists a pair of functions $\langle h, H \rangle$ that 1) *do* commute on $M^{(1)}, \ldots, M^{(m)}$ and 2) dominate $\langle \vec{f}, F \rangle$ on these matrices. Corollary 3, which follows immediately from the definition of the MD algorithm, gives us a simple property of this algorithm that provides some intuition for the subsequent steps of construction.

Corollary 3. *Given a set of schema matchers $A_1, \ldots, A_m$ and a pair of local and global aggregators $\langle h, H \rangle$ commuting on $M^{(1)}, \ldots, M^{(m)}$, the top-K result of the MD algorithm with respect to $\langle h, H \rangle$ is a correct top-K aggregation with respect to any set of l- and g-aggregators $\langle \vec{f}, F \rangle$, such that both $\langle h, H \rangle \succ \langle \vec{f}, F \rangle$ and $\langle \vec{f}, F \rangle \succ \langle h, H \rangle$ hold on $M^{(1)}, \ldots, M^{(m)}$.*

In general, nothing prevents Corollary 3 to be realized. To illustrate that, consider the following set of four real-valued functions: $f(x) = x^2$, $F(x) = x/2$, $h(x) = x^2/2$, and $H(x) = x$. Although f and F do not commute on reals ($F(f(x)) = x^2/2$ and $f(F(x)) = x^2/4$), the functions h and H *are* strongly commutative ($H(h(x)) = h(H(x)) = x^2/2$), and we have $H(h(x)) = F(f(x))$. However, the practical realizability of Corollary 3 with respect to schema metamatching is less clear, as it is not clear whether there exists a set of four functions that will be interesting in practice for schema metamatching.

Corollary 3, however, does provide us with some useful intuition. Consider a schema metamatching problem defined by a set of similarity matrices $M^{(1)}, \ldots, M^{(m)}$ and l- and g-aggregators $\langle \vec{f}, F \rangle$ that *do not* commute on $M^{(1)}, \ldots, M^{(m)}$. Suppose that there exists a pair of functions $\langle h, H \rangle$ that *do* commute on $M^{(1)}, \ldots, M^{(m)}$ and dominate $\langle \vec{f}, F \rangle$ on these matrices, yet is not dominated by $\langle \vec{f}, F \rangle$. For instance, let F be a weighted sum as in (4), and f be defined as

$$f^{(i)}(\sigma, M) = \begin{cases} \sum_{j=1}^{n} M_{j,\sigma(j)}, & \sum_{j=1}^{n} M_{j,\sigma(j)} > t_i \\ 0, & \text{otherwise,} \end{cases} \tag{7}$$

where $t_i > 0$ is some predefined constant threshold. The intuition behind (7) is that judges that can no longer provide mappings with sufficient similarity measure (set as the threshold t_i) "quit" by nullifying all further mappings. Another example, reflecting one of the currently used settings in schema matching (for example, [4] and [37]), is

$$f^{(i)}(\sigma, M) = \sum_{j=1}^{n} \big(M_{j,\sigma(j)} \cdot \delta \big(M_{j,\sigma(j)} > t_j \big) \big), \tag{8}$$

where δ is the Kronecker discrete delta function. According to (8), individual pairwise attribute mappings that do not pass a predefined matcher-specific threshold are nullified. In both cases, it is not hard to verify that $\vec{f}$ and F do not commute (in all but trivial cases of effectively redundant thresholds.) On the other hand, functions h and H standing for simple sum and weighted sum (as in (4)) are (strongly) commutative, and we have $\langle \vec{f}, F \rangle \prec \langle h, H \rangle$ for both (7) and (8). For such cases, we now present the MDB algorithm. This algorithm draws upon both the TA and MD algorithms, addressing problems with noncommutative pairs of local and global aggregation functions, while being more efficient than the TA algorithm in at least some such problem instances.

The MDB algorithm is shown in Fig. 3. Consider a schema metamatching problem with schema matchers $A_1, \ldots, A_m$ and aggregators $\langle \vec{f}, F \rangle$ that do not commute on $M^{(1)}, \ldots, M^{(m)}$. As we already mentioned, the basic idea behind the MDB algorithm is to use a pair of functions $\langle h, H \rangle$ (that both dominate $\langle \vec{f}, F \rangle$ and commute

```
1) Given $A_1, \ldots, A_m$, (schematically) construct a new schema matcher $A^*$ with (a) similarity matrix $M^*$ such that, for $1 \leq i \leq n, 1 \leq j \leq n'$, $M^*_{i,j} = H(M^{(1)}_{i,j}, \cdots, M^{(m)}_{i,j})$, and (b) $l$-aggregator $h(\sigma, M^*)$.
2) Starting with $i = 1$, do incremental (on growing $i$) querying of $A^*$ with q-top($i$).
    a) As a mapping $\sigma$ is introduced, obtain the actual weights $f^{(1)}(\sigma, M^{(1)}), \cdots, f^{(m)}(\sigma, M^{(m)})$ by querying $A_1, \ldots, A_m$ with q-estim($\sigma$), and compute the aggregated weight $\langle \vec{f}, F \rangle(\sigma)$. If this weight is one of the $K$ highest we have seen so far, then remember $\sigma$.
    b) Define the threshold value $\tau_{MDB}$ to be $h(\sigma, M^*)$.[a] If $K$ mappings have been seen whose weight is at least $\tau_{MDB}$, then halt.
3) Let $Y$ be a set containing $K$ mappings with the highest grades seen so far. The output is then the graded set $\left\{ \left[ \sigma, \langle \vec{f}, F \rangle(\sigma) \right] \mid \sigma \in Y \right\}$.
```

[a]It is worth noting that, due to commutativity of h and H (either strong or just with respect to $M^{(1)}, \ldots, M^{(m)}$), we have $\tau_{MDB} = \langle H, h \rangle(\sigma) = \langle h, H \rangle(\sigma)$.

Fig. 3. The MDB algorithm.

on $M^{(1)}, \ldots, M^{(m)}$) as an upper bound for the "inconvenient" $\langle \vec{f}, F \rangle$ of our actual interest. Informally, the MDB algorithm behaves similarly to the MD algorithm if the latter is given with the aggregators $\langle h, H \rangle$. However, instead of reporting immediately on the generated mappings σ, the MDB algorithm uses the decreasing aggregated weights $\langle h, H \rangle(\sigma)$ to update the value of a threshold τ_{MDB}. In turn, as much as the way the threshold τ_{TA} is used in the TA algorithm, the threshold τ_{MDB} is used to judge our progress with respect to the weights $\langle \vec{f}, F \rangle$ that really matter. Theorem 4 shows that the MDB algorithm is correct for any such upper bound $\langle h, H \rangle$.

Theorem 4. *Consider a set of schema matchers $A_1, \ldots, A_m$, with $\langle \vec{f}, F \rangle$ being their l- and g-aggregators. Given a function pair $\langle h, H \rangle$ that both commute and dominate $\langle \vec{f}, F \rangle$ on $M^{(1)}, \ldots, M^{(m)}$, the MDB algorithm correctly finds the top-K valid mappings with respect to $\langle \vec{f}, F \rangle$.*

Returning to the question of performance, recall that our intention in developing the MDB algorithm was to provide an alternative to the TA algorithm for ensemble-aggregation settings where the standard MD is not applicable. Have we achieved our goal, or will the TA algorithm always be more efficient anyway? We now show that, for schema metamatching, MDB can significantly outperform TA.

Theorem 5. *Given a schema metamatching problem instance, the time complexity of the TA algorithm on this instance can be exponentially worse than that of the MDB algorithm.*

Theorem 5 shows that the TA algorithm does not dominate the MDB algorithm, but it says nothing about the opposite direction: Does the MDB algorithm dominate the TA algorithm, or maybe the relative attractiveness of these two algorithms (complexity-wise) depends on the actual metamatching instance at hand? The problem with answering this question in a general manner is that the running time of the MDB algorithm depends on the choice of bounding functions. Therefore, dominance of the MDB algorithm over the TA algorithm would mean that, for *each* metamatching problem instance and *each* K, the *optimal* choice of bounding functions $\langle h, H \rangle$ will make the MDB algorithm at least as efficient as TA. At this stage, we have no evidence that this is actually the case. In fact, so far it is not even clear that the above notion of optimality has a clear mathematical semantics. It is worth noting here that the actual tightness of $\langle h, H \rangle$ with respect to $\langle \vec{f}, F \rangle$ is only one factor in determining the efficiency of the MDB algorithm, and the optimality as above should also relate to this or another notion of order preserving: Intuitively, the MDB algorithm is most efficient if the order induced by $\langle h, H \rangle$ over alternative schema matchings coincides with the order induced by $\langle \vec{f}, F \rangle$, and $\langle h, H \rangle$ is sufficiently tight to allow the discovered mappings to cross τ_{MDB} quickly enough. On the other extreme, the MDB algorithm is least efficient if the order induced by $\langle h, H \rangle$ is the inversion of the order induced by $\langle \vec{f}, F \rangle$. Later, we provide an algorithm that makes use of the "good" mappings that were discovered by the MDB algorithm even when $\langle h, H \rangle$ fails to provide a sufficiently tight threshold. As for order preserving, the superiority of the algorithm should hold for all problem instances and all choices of K, and it is not clear how (if at all) this notion can be defined in a problem-instance-independent manner.

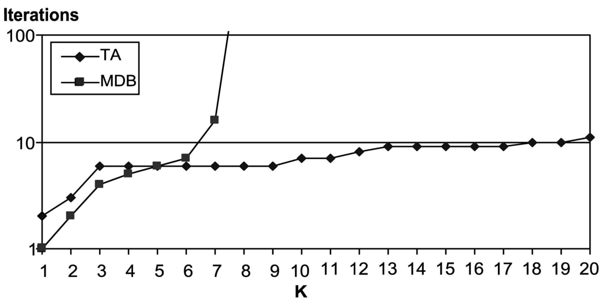


Fig. 4. Crossing performance of the TA and MDB algorithms on a certain problem instance from our experiments.

In the absence of a general relation as above, the question now is whether or not we can say something about the attractiveness of the TA algorithm with respect to the MDB algorithm that is equipped with a "reasonable" pair of bounding aggregators. Theorem 6 provides an affirmative answer to this question and shows that TA can significantly outperform MDB.

Theorem 6. *Given a schema metamatching problem instance, the time complexity of the MDB algorithm on this instance can be exponentially worse than that of the TA algorithm.*

5 The CrossThreshold Algorithm

The main conclusion to be drawn from Theorem 6 is that the MDB algorithm should not replace but rather complement the TA algorithm. Thus, it would be natural to adopt a parallel execution of TA and MDB, that is, performing $m + 1$ parallel q-top querying of schema matchers. This way, we involve both algorithms in computing the top-K mappings, halting as soon as one of these algorithms reaches the desired goal.

The question that suggests itself immediately is whether we can improve the performance of this parallel execution of the TA and MDB algorithms by either monitoring their intermediate behavior or letting TA and MDB share some information gathered from their own individual computations. Our discussion of this possibility leads to specification and analysis of a mixed version of the TA and MDB algorithms, in which TA and MDB are executed in parallel; yet, these parallel executions are not independent but rather communicating and mutually enhancing.

5.1 Is an Early Winner a True Winner?

A naïve approach to accelerate parallel execution of the TA and MDB algorithms corresponds to the hypothesis that by observing the performance of both TA and MDB in identifying the top-i mappings (where $i < K$), a decision can be taken to continue with only one of these algorithms in identifying the remaining $K - i$ mappings. Observe that such an "early winner detection" will be especially helpful in problems where de facto MDB outperforms TA, since TA's execution of m parallel q-top querying is more costly than executing MDB.

Unfortunately, such a selection strategy provides us with no guarantee that the performance will not worsen after abandoning the "so-far looser" algorithm. More interestingly, our experiments show that this absence of guarantee is not of theoretical interest only. For instance, Fig. 4 compares the performance of the TA and MDB algorithms on two schemata from the hotel reservation domain. The x- and y-axes in Fig. 4 correspond to the requested number of the top mappings K and the (plotted on a logarithmic scale) number of iterations performed by the algorithms, respectively. On this problem instance, the MDB algorithm manages to get the top-4 mappings faster than the TA algorithm. However, from $K = 6$ on, TA outperforms MDB. This example demonstrates that in practice as well, the relative performance of the TA and MDB algorithms for any $i < K$ cannot serve as a perfect indicator for their future behavior. (We discuss our experiments in more details in Section 6.)

5.2 Can the TA and MDB Algorithms Help Each Other?

Although we have shown that neither the TA algorithm nor the MDB algorithm can be safely abandoned, a natural next step would be to allow these two algorithms to share a pool of the top-K candidates. This way, both algorithms will contribute to each other new candidates as they come along, possibly replacing other less attractive candidates (discovered by either of the two algorithms.) Moreover, such a pool sharing suggests aggregating the thresholds used in the TA and MDB algorithms, achieving a new threshold that is more effective than the original two. This way, the schema mappings selected by the TA algorithm as candidates for the top-K set can be "approved" by means of the information obtained by the MDB algorithm, and vice versa.

Fig. 5 formalizes the resulting algorithm, which we refer to as CrossThreshold. The joint threshold τ used in this algorithm is set to $\min\{\tau_{TA}, \tau_{MDB}\}$, and Theorem 7 shows the correctness of the CrossThreshold algorithm with such a threshold. Note that this choice of τ for the CrossThreshold algorithm is optimal, because any other $\tau' > \tau$ cannot be more effective than τ, whereas setting τ to any value lower than $\min\{\tau_{TA}, \tau_{MDB}\}$ cannot guarantee the soundness of the procedure.

Theorem 7. *Let $A_1, \ldots, A_m$ be a set of schema matchers with $\langle \vec{f}, F\rangle$ being their l- and g-aggregators. Given a function pair $\langle h, H\rangle$ that both commute and dominate $\langle \vec{f}, F\rangle$ on $M^{(1)}, \ldots, M^{(m)}$, the CrossThreshold algorithm correctly finds the top-K valid mappings with respect to $\langle \vec{f}, F\rangle$.*

Although Theorem 7 shows the correctness of the CrossThreshold algorithm, the reader may rightfully wonder whether it can provide any computational speedup compared to the basic independent parallel execution of the TA and MDB algorithms. Below, we provide an affirmative answer to this question.

Considering the generation of the top-K mappings for a general schema metamatching problem, let I be the minimal number of iterations required for this purpose by the CrossThreshold algorithm and I_{TA} and I_{MDB} be the corresponding minimal number of iterations required by independently running the TA and MDB algorithms, respectively. If using CrossThreshold provides any computational speedup on this problem instance, then we should have

$$I < \min\{I_{TA}, I_{MDB}\}. \qquad (9)$$

```
1) Given $A_1, \ldots, A_m$, (schematically) construct a new schema matcher $A^*$ with (a) similarity matrix $M^*$ such that, for $1 \le i \le n, 1 \le j \le n'$, $M^*_{i,j} = H(M^{(1)}_{i,j}, \cdots, M^{(m)}_{i,j})$, and (b) $l$-aggregator $h(\sigma, M^*)$.
2) Starting with $i = 1$, do incremental (on growing $i$) parallel querying of $A_1, \ldots, A_m, A^*$ with q-top($i$).
    a) As a mapping $\sigma$ is introduced by one of these $m + 1$ matchers, obtain the remaining $f^{(1)}(\sigma, M^{(1)}), \cdots, f^{(m)}(\sigma, M^{(m)})$ by querying the other matchers (excluding $A^*$) with q-estim($\sigma$), and compute the aggregated weight $\langle \vec{f}, F\rangle(\sigma)$. If this weight is one of the $K$ highest we have seen so far, then remember $\sigma$.
    b) Let $\sigma_1, \ldots, \sigma_m, \sigma_*$ be the mappings returned by the *last* q-top queries by $A_1, \ldots, A_m, A^*$, respectively. Define the threshold value $\tau = \min\{\tau_{TA}, \tau_{MDB}\}$, where $\tau_{TA} = F\Big(f^{(1)}(\sigma_1, M^{(1)}), \cdots, f^{(m)}(\sigma_m, M^{(m)})\Big)$ and $\tau_{MDB} = h(\sigma_*, M^*)$. If $K$ mappings have been seen whose weight is at least $\tau$, then halt.
3) Let $Y$ be a set containing $K$ mappings with the highest grades seen so far. The output is then the graded set $\Big\{\Big[\sigma, \langle \vec{f}, F\rangle(\sigma)\Big] \mid \sigma \in Y\Big\}$.
```

Fig. 5. The CrossThreshold algorithm.

To obtain some intuition on when (if at all) (9) may hold, let $\tau[x]$, $\tau_{TA}[x]$, and $\tau_{MDB}[x]$ be the values obtained after x iterations by τ, τ_{TA}, and τ_{MDB}, respectively. By definition of the CrossThreshold algorithm and its reported top-K list Y, we have

$$\forall \sigma \in Y: \quad \langle \vec{f}, F\rangle(\sigma) \ge \tau[I] = \min\{\tau_{TA}[I], \tau_{MDB}[I]\} \qquad (10)$$

and without loss of generality, assume that $\tau_{TA}[I] \neq \tau_{MDB}[I]$.

First, suppose that the value of $\tau[I]$ is contributed by the TA algorithm, that is, $\tau[I] = \tau_{TA}[I]$; thus, (10) can be reformulated as

$$\forall \sigma \in Y: \quad \langle \vec{f}, F\rangle(\sigma) \ge \tau_{TA}[I]. \qquad (11)$$

On the other hand, (9), in particular, implies that there exists at least one mapping $\sigma \in Y$ that has not been seen by TA. Thus, after I iterations, such a mapping σ is *exclusively* provided to the shared pool of candidates by the MDB algorithm, yet, the TA algorithm can successfully verify membership of σ in the top-K list. The situation with $\tau[I] = \tau_{MDB}[I]$ is symmetric; in this case, there exists at least one mapping $\sigma \in Y$ that is exclusively discovered by the TA algorithm, and yet its membership in the top-K list can be successfully verified by the MDB algorithm.

We now formalize this intuition to characterize schema metamatching problem instances on which the CrossThreshold algorithm can provide a computational speedup over its basic "asynchronous" counterpart. Starting with (9) and (10), we provide two lemmas that significantly reduce the spectrum of scenarios in which such a speedup is theoretically possible. Specifically, Lemmas 8 and 9 restrict the global aggregator value of mappings that are identified by one algorithm and verified with the appropriate threshold of the other algorithm to be *equal* to the joint threshold τ.

Extending the notation introduced in Section 5.2, let $Y = Y_{TA} \cup Y_{MDB}$ be a (possibly not disjoint) cover of Y, where Y_{TA} and Y_{MDB} contain the top-K mappings provided to the CrossThreshold algorithm by the q-top queries to $A_1, \ldots, A_m$ and A^*, respectively.

Lemma 8. *If $I < I_{TA}$ and $\tau[I] = \tau_{TA}[I]$, then $Y \setminus Y_{TA} \neq \emptyset$, and for each $\sigma \in Y \setminus Y_{TA}$, we have*

$$\langle \vec{f}, F \rangle(\sigma) = \tau_{TA}[I].$$

Lemma 9. *If $I < I_{MDB}$ and $\tau[I] = \tau_{MDB}[I]$, then $Y \setminus Y_{MDB} \neq \emptyset$, and for each $\sigma \in Y \setminus Y_{MDB}$, we have*

$$\langle \vec{f}, F \rangle(\sigma) = \tau_{MDB}[I].$$

At first view, it seems that the restrictions posed by Lemmas 8 and 9 are too strict for the CrossThreshold algorithm to provide a significant computational speedup (if any). However, below, we show that even in such boundary situations, the speedup is not only possible, but also potentially significant.

Theorem 10. *There exist schema metamatching problem instances for which the time complexity of both the TA and MDB algorithms is exponentially worse than that of the CrossThreshold algorithm.*

6 Empirical Evaluation

We have implemented the generic versions of the four algorithms TA, MD, MDB, and CrossThreshold.[7] In this implementation, each algorithm can be plugged in with a concrete schema model (for example, relational), a set of (standard or user defined) schema matchers, and a pair of l- and g-aggregators.

As a testbed, we have gathered 24 Web forms from six different domains, namely, dating and matchmaking, job hunting, Web mail, hotel reservation, news, and cosmetics. We first extracted the schemata of these Web forms using the OntoBuilder ontology extractor. Then, we generated the similarity matrices for all pairs of domain-compatible Web forms using four different schema matchers called `Term`, `Value`, `Composition`, and `Precedence` [23]. The valid schema mappings have been defined to be all the mappings obeying one-to-one cardinality constraint.

In our experiments, we have evaluated the TA, MDB, and CrossThreshold algorithms on five pairs of these matchers, namely, {`Term`, `Value`}, {`Term`, `Precedence`}, {`Term`, `Composition`}, {`Value`, `Precedence`}, and {`Value`, `Composition`}. Likewise, all these 15 schema metamatching settings have been evaluated on nine pairs of l- and g-aggregators $\langle f, F \rangle$, namely, $\langle \text{avg}, \min \rangle$ and $\langle \text{avg}(t), \text{avg} \rangle$, where $\text{avg}(t)$ stands for the average version of (7), and $t \in \{0.025, 0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 0.75\}$. To eliminate possible influence of having different l-aggregators for different schema matchers on the conclusiveness of the evaluation, in all these 135 experiment configurations, the matchers have been set to use the same l-aggregator. Likewise, in all these configurations, we have used $\langle \text{avg}, \text{avg} \rangle$ as the dominating pair of bounding aggregators $\langle h, H \rangle$ and generated up to $K = 20$ top mappings.

To summarize, we have experimented with 12 pairs of schemata, five groups of schema matcher pairs, and nine pairs of l- and g-aggregators, for a total of 540 comparative experiments between the TA, MDB, and CrossThreshold algorithms. Below, we discuss the results of our empirical evaluation of TA, MDB, and CrossThreshold. Note that empirical evaluation of the MD algorithm is redundant, as the running time of MD on a given problem instance can be derived analytically from Theorem 2.

6.1 Evaluating the TA and MDB Algorithms

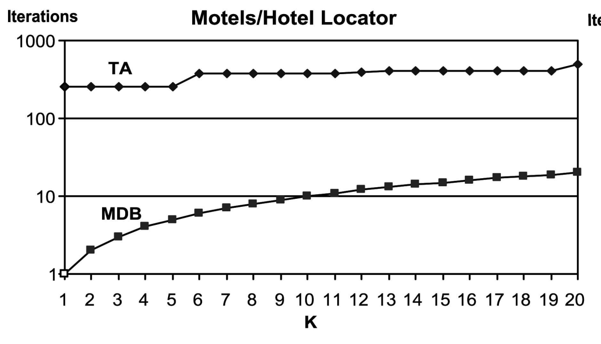
Recall that Theorems 5 and 6 show that the TA and MDB algorithms do not dominate each other. These formal results, however, say little about the practical relationship between the two algorithms. Interestingly, our experiments on real-world schemata support the formal conclusion of Theorems 5 and 6 that there is no clear winner between the TA and MDB algorithms.

To start with a concrete example, in Fig. 6, we present the performance of the TA and MDB algorithms on two different pairs of schemata, while employing the *same* pair of matchers {`Term`, `Precedence`} and the *same* pair of l- and g-aggregators $\langle f, F \rangle = \langle \text{avg}(0.25), \text{avg} \rangle$ (bounded by $\langle h, H \rangle = \langle \text{avg}, \text{avg} \rangle$.) The x- and y-axes in these graphs correspond respectively to the requested number of the top mappings K and the number of iterations performed by the algorithms (plotted on a logarithmic scale). It is easy to see that the MDB algorithm significantly outperforms the TA algorithm on the problem instance depicted in Fig. 6a, whereas the TA algorithm significantly outperforms the MDB algorithm on the problem instance depicted in Fig. 6b. Thus, Fig. 6 clearly shows that performance incomparability between the TA and MDB algorithms is not restricted to some extreme pathological problem instances. Moreover, this example illustrates that these two algorithms are incomparable even if there is no difference between their settings neither in the choice of schema matchers nor in the choice of l- and g-aggregators.

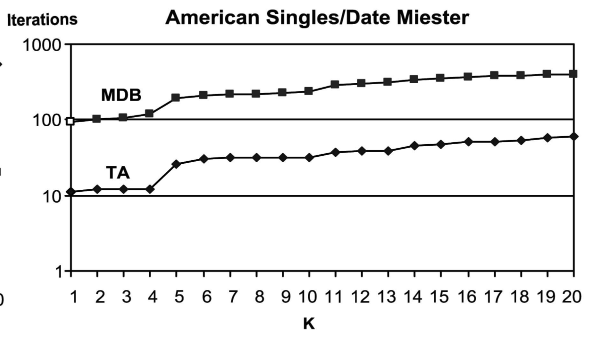
Table 1 summarizes the relative performance of the MDB and TA algorithms on various pairs of schemata and various choices of schema matcher groups. The rows in Table 1 correspond to the different choices of the l- and g-aggregators, whereas its four columns capture the percentage of experiments in which:

1. TA performed at least as well as MDB for all $1 \leq K \leq 20$ and outperformed MDB for at least one such K;
2. MDB performed at least as well as TA for all $1 \leq K \leq 20$ and outperformed TA for at least one such K;
3. TA and MDB performed exactly the same; and
4. none of these two algorithms dominated the other for *all* $1 \leq K \leq 20$.

7. This requires Java 2 JDK 1.4 (or higher) and is available as part of the OntoBuilder distribution from http://ie.technion.ac.il/OntoBuilder.



(a)



(b)

Fig. 6. The TA and MDB algorithms with schema matchers {`Term`, `Precedence`} and $\langle f, F\rangle = \langle \text{avg}(0.25), \text{avg}\rangle$ evaluated on two different pairs of schemata.

This table further illustrates that the performance of the MDB and TA algorithms is generally incomparable. Likewise, as it was expected, Table 1 shows that the performance of the MDB algorithm correlates with the relative informativeness of our bounding aggregators $\langle \text{avg}, \text{avg}\rangle$ with respect to the actual l- and g-aggregators in use. Specifically, the lower the threshold is, the better the MDB algorithm performs.

6.2 Evaluating the CrossThreshold Algorithm

Next, we compare the empirical performance of the CrossThreshold algorithm with the independent in-parallel execution of the TA and MDB algorithms. Recalling that the CrossThreshold algorithm is always at least as effective as its basic counterpart, and that Theorem 10 implies the theoretical feasibility of (9), our intention here is to check whether the computational gain from using the CrossThreshold algorithm can also be observed in practice.

Let $I_{\min} = \min\{I_{TA}, I_{MDB}\}$ denote the number of iterations required to solve a given schema metamatching problem using the basic in-parallel execution of the TA and MDB algorithms. Fig. 7 illustrates the relative performance of the CrossThreshold algorithm with respect to in-parallel execution by plotting the ratio $I_{\min}/I$ averaged over all 12 tested pairs of schemata and five tested pairs of schema matchers. Since we always have $I \leq I_{\min}$, the (averaged) ratio $I_{\min}/I$ is always bounded from below by 1. Each vertex on this surface corresponds to an average ratio for a certain number of required mappings K (x-axis) and a certain choice of l- and g-aggregator functions (y-axis).

Fig. 7 clearly shows that using the CrossThreshold algorithm is beneficial not only in theory, but also in practice. Averaging over all 540 experimental sessions, the CrossThreshold algorithm was $\approx$ 16 percent faster than its basic counterpart. For the aggregator pairs $\langle \text{avg}(t), \text{avg}\rangle$, the relative benefit of using the CrossThreshold communication between the TA and MDB algorithms was roughly proportional to the cutoff value t. The intuition behind this relationship is that, as t gets closer to 0, the values that the bounding functions $\langle \text{avg}, \text{avg}\rangle$ provide to the mappings are closer to those provided by the actual aggregators $\langle \text{avg}(t), \text{avg}\rangle$, and thus, the MDB algorithm is getting closer to the "perfect" algorithm MD.

Now, consider the pair of local and global aggregators $\langle \text{avg}, \min\rangle$. Recall that in our experiments with this pair of aggregators, the TA algorithm outperformed the MDB algorithm in 95 percent of the experiments for any $1 \leq K \leq 20$ (see the last row in Table 1). If so, then one would expect the performance of the CrossThreshold algorithm on these problems to be similar to that of the TA algorithm, as it seems that the MDB algorithm will have nothing to contribute to the process. However, Fig. 7 shows exactly the opposite; not only does the CrossThreshold algorithm outperform the TA algorithm on this problem set, but the marginal contribution of using it was also the largest among all the pairs of l- and g-aggregators considered.

TABLE 1
Relative Performance of the MDB and TA Algorithms

	TA ≥ MDB	MDB ≥ TA	TA=MDB	no winner
$\langle \text{avg}(0.025), \text{avg}\rangle$	8%	57%	27%	8%
$\langle \text{avg}(0.05), \text{avg}\rangle$	17%	53%	22%	8%
$\langle \text{avg}(0.10), \text{avg}\rangle$	28%	40%	18%	13%
$\langle \text{avg}(0.15), \text{avg}\rangle$	42%	33%	15%	10%
$\langle \text{avg}(0.20), \text{avg}\rangle$	47%	23%	12%	18%
$\langle \text{avg}(0.25), \text{avg}\rangle$	65%	20%	3%	12%
$\langle \text{avg}(0.50), \text{avg}\rangle$	78%	15%	0%	7%
$\langle \text{avg}(0.75), \text{avg}\rangle$	97%	2%	0%	2%
$\langle \text{avg}, \min\rangle$	95%	2%	0%	3%

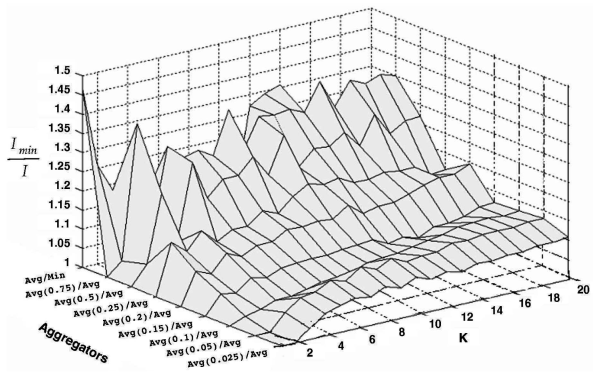


Fig. 7. The CrossThreshold algorithm versus independent in-parallel execution of the TA and MDB algorithms.

This phenomenon corresponds to a certain interesting form of "mutual assistance" between the TA and MDB algorithms in CrossThreshold, the possibility of which we have exploited in our proof of Theorem 10. Recall our discussion that the efficiency of the MDB algorithm is affected by *two* separate factors. First, top mappings with respect to $\langle \vec{f}, F\rangle$ might be pushed down when using $\langle h, H\rangle$. However, even if this is not the case and the MDB algorithm immediately finds the true top mappings, the algorithm may not be able to verify them due to the threshold τ_{MDB}, which is too high. Now, consider MDB embedded in the CrossThreshold algorithm and a schema-matching problem instance corresponding to the latter situation. Despite the fact that the MDB algorithm fails to report the top-K mappings, it can still successfully provide the right *candidates*. In turn, these candidates can be approved by the (lower) threshold τ_{TA}, whereas the TA algorithm may fail to generate good candidates by itself. In such situations, the marginal contribution of using the CrossThreshold algorithm is expected to be the highest, and this was exactly the typical situation in our experiments on problem instances with $\langle f, F\rangle = \langle \text{avg}, \text{min}\rangle$.

To summarize, our experiments demonstrate the practical advantages of using the CrossThreshold algorithm over the basic in-parallel execution of the TA and MDB algorithms. Hence, the CrossThreshold algorithm provides a more appealing solution for situations in which one is uncertain about the relative attractiveness of the TA and MDB algorithms in a domain of discourse.

7 Conclusions and Future Work

We introduced schema metamatching, a novel computational framework for robust automatic schema matching that generalizes and extends previous proposals for exploiting an ensemble of schema matchers. We presented several algorithms for schema metamatching, varying from adaptation of a standard technique for quantitative rank aggregation in the area of database middleware, to novel techniques developed especially for the problem of schema matching. We provided a formal computational analysis of all algorithms and characterized their relative applicability. In particular, our formal analysis allowed us to devise a pair of strictly superior algorithms MD and CrossThreshold, where the choice between the two depends on whether the l- and g-aggregators commute on the schema-matching problem at hand, which is an easy-to-check property. Likewise, we evaluated all the algorithms empirically on a set of real-life schemata gathered from Web forms and a set of state-of-the-art schema matchers. Our experiments demonstrate the benefit of using the CrossThreshold algorithm over using the TA or MDB algorithms independently or in parallel.

Our work opens several venues for future research, two of which are discussed below. First, observe that in the TA algorithm (and thus in the CrossThreshold algorithm), the parallel querying of different matchers with the q-top queries is kept uniform, that is, each iteration of the TA algorithm progresses on *all* the matchers. In general-purpose aggregation of quantitative rankings [17], this strategy is indeed expected to be as good as any other strategy. However, having additional knowledge about the data can provide us with (at least heuristically) better strategies and, currently, we are exploring this direction to further improve the performance of the CrossThreshold algorithm. Second, as discussed in Section 4, it is clear that the complexity of our MDB algorithm depends crucially on the quality of the chosen pair of dominating aggregators. Therefore, we are looking into refining our notion of dominance by incorporating topological measures of ordering and tightness.

Appendix

Theorem 1. *The time complexity of schema metamatching using TA is* $\Omega((\frac{\eta}{2})!)$.

Proof. The proof of this lower bound is by construction of a certain set of similarity matrices for which the TA algorithm finds the best mapping only after $O((\frac{\eta}{2})!)$.

Consider two algorithms, A_1 and A_2, and a pair of schemata S and S', each consisting of n attributes, where $n = 2k,\ k \in \mathbb{N}$ (and thus $\eta = n$). Likewise, let the l-aggregators $f^{(1)} = f^{(2)}$ both be the regular *product* (denoted by f), Σ_Γ be the set of all one-to-one mappings from Σ, and the g-aggregator F be the utilitarian aggregator *min*. Given S and S', the similarity matrices $M^{(1)}$ and $M^{(2)}$, induced by A_1 and A_2, respectively, are given as follows:

$$M^{(1)}_{i,j} = \begin{cases} x, & (i \le n/2) \wedge (j \le n/2) \wedge (i \ne j) \\ x - \epsilon, & i = j \\ 0, & \text{otherwise} \end{cases}$$

$$M^{(2)}_{i,j} = \begin{cases} x, & i > n/2 \wedge j > n/2 \wedge i \ne j \\ x - \epsilon, & i = j \\ 0, & \text{otherwise} \end{cases}$$

for arbitrary positive values of x and ϵ, where $\epsilon \ll x$, and $x - \epsilon > 0$. We illustrate these matrices for $n = 4$ as follows:

$$M^{(1)} = \begin{pmatrix} x-\epsilon & x & 0 & 0 \\ x & x-\epsilon & 0 & 0 \\ 0 & 0 & x-\epsilon & 0 \\ 0 & 0 & 0 & x-\epsilon \end{pmatrix}$$
$$M^{(2)} = \begin{pmatrix} x-\epsilon & 0 & 0 & 0 \\ 0 & x-\epsilon & 0 & 0 \\ 0 & 0 & x-\epsilon & x \\ 0 & 0 & x & x-\epsilon \end{pmatrix}.$$

First, consider $M^{(1)}$. Each valid mapping between the first $n/2$ attributes of S and the first $n/2$ attributes of S' (see the top left quadrant of $M^{(1)}$) results in a nonzero value of f restricted to these attributes. There are $(\frac{n}{2})!$ such mappings. Any other mapping of any of these attributes will nullify the value of f. On the other hand, the last $n/2$ attributes of S have to be mapped to the $n/2$ last attributes of S', and there is only one such mapping leading to a nonzero value of f, namely, the main diagonal of the bottom right quadrant of $M^{(1)}$. Therefore, we have constructively shown that $M^{(1)}$ induces exactly $(\frac{n}{2})!$ mappings σ, such that $f(\sigma, M^{(1)}) > 0$. Denote this set of mappings by $\Sigma_1^+ \subset \Sigma_\Gamma$. By a similar construction, the same holds for $M^{(2)}$, that is, $|\Sigma_2^+| = (\frac{n}{2})!$. Now, consider the sets Σ_1^+ and Σ_2^+, and let σ_I denote the indentity mapping, that is, the mapping captured by the main diagonals of $M^{(1)}$ and $M^{(2)}$. Evidently, for $l \in \{1,2\}$, we have $\sigma_I \in \Sigma_l^+$, and for each $\sigma_I \neq \sigma \in \Sigma_l^+$, we have $f(\sigma, M^{(l)}) > f(\sigma_I, M^{(l)})$. Therefore, σ_I will be discovered by the TA algorithm after exactly $(\frac{n}{2})!$ q-top queries to each of $A^{(1)}$ and $A^{(2)}$. On the other hand, we have $\Sigma_1^+ \cap \Sigma_2^+ = \{\sigma_I\}$, and thus, for each mapping $\sigma \in \Sigma$, we have

$$\langle f, F\rangle(\sigma) = \min\left\{\prod_{i=1}^{n} M^{(1)}_{i,\sigma(i)}, \prod_{i=1}^{n} M^{(2)}_{i,\sigma(i)}\right\} = \begin{cases} n(x-\epsilon), & \sigma = \sigma_I \\ 0, & \text{otherwise.}\end{cases}$$

This means that, under the considered aggregators f and F, the top-1 mapping between S and S' has to be σ_I. However, it will take the TA algorithm $(\frac{n}{2})!$ iterations to discover σ_I. □

Theorem 2. *Given a set of schema matchers $A_1, \ldots, A_m$ and a pair of local and global aggregators $\langle f, F\rangle$, let M^* be a matrix defined as $M^*_{i,j} = F(M^{(1)}_{i,j}, \cdots, M^{(m)}_{i,j})$ for all $1 \le i \le n, 1 \le j \le n'$. If f and F commute on the similarity matrices $M^{(1)}, \ldots, M^{(m)}$, then the MD algorithm correctly finds the top-K valid mappings with respect to the aggregated ranking in time $O(\eta^2 m + \Phi)$, where Φ is the combined time complexity of iteratively executed queries $\mathbf{q\text{-}top}(1), \ldots, \mathbf{q\text{-}top}(K)$ over M^*.*

Proof. The correctness is immediate from the construction of the MD algorithm and Definition 1. As F is assumed to be computable in time linear in the number of F's parameters, generating M^* takes time $O(\eta^2 m)$. Thus, the overall complexity of the MD algorithm is $O(\eta^2 m + \Phi)$. For instance, for aggregators as in (4) and Γ enforcing one-to-one cardinality constraint, the time complexity of the MD algorithm is $O(\eta^4 K + \eta^2 m)$. □

Theorem 4. *Consider a set of schema matchers $A_1, \ldots, A_m$, with $\langle \vec{f}, F\rangle$ being their l- and g-aggregators. Given a function pair $\langle h, H\rangle$ that both commute and dominate $\langle \vec{f}, F\rangle$ on $M^{(1)}, \ldots, M^{(m)}$, the MDB algorithm correctly finds the top-K valid mappings with respect to $\langle \vec{f}, F\rangle$.*

Proof. Let Y be as in step 3 of the MDB algorithm. We need only show that every mapping $\sigma \in Y$ has at least as high weight according to $\langle \vec{f}, F\rangle$ as every mapping $\sigma' \notin Y$. By definition of Y, this is the case for each mapping $\sigma' \notin Y$ that has been seen by MDB. Thus, assume that σ' was not seen. By the definition of τ_{MDB} as in step 2b of the MDB algorithm and the incrementality of querying A^* with q-top, for each such unseen σ' and for each $\sigma \in Y$, we have

$$\langle \vec{f}, F\rangle(\sigma) \ge \tau \ge \langle h, H\rangle(\sigma') \ge \langle \vec{f}, F\rangle(\sigma')$$

where τ is the value of τ_{MDB} at termination of MDB. The second inequality holds since σ' has not been seen and therefore $\langle h, H\rangle(\sigma')$ cannot receive a value higher than τ. Thus, we have proven that Y contains the top-K mappings with respect to $\langle \vec{f}, F\rangle$. □

Theorem 5. *Given a schema metamatching problem instance, the time complexity of the TA algorithm on this instance can be exponentially worse than that of the MDB algorithm.*

Proof. The proof is by example of the corresponding problem instance. Specifically, we consider the class of schema metamatching problems used in the proof of Theorem 1 and show that, for a certain subclass of these problems, the MDB algorithm can identify the best mapping after only two iterations.

Consider the schema metamatching problem exactly as in the proof of Theorem 1 and assume further that $x \in (0, 1]$. We already showed that on this problem instance the TA algorithm performs $\Omega((\frac{n}{2})!)$ iterations for $K = 1$. Recall that the aggregators f and F in this example stand for *product* and *min*, respectively. Hence, f and F do not commute on this problem similarity matrices and, thus, the MD algorithm cannot be used for this problem instance. Now, consider a pair of functions $\langle h, H\rangle$, where both h and H stand for a simple *average*. Observe that, since the entries of both matrices $M^{(1)}$ and $M^{(2)}$ lie in the interval [0, 1], we have $\langle f, F\rangle \prec \langle h, H\rangle$. Likewise, since h and H are (trivially) strongly commutative, we can solve this problem instance using the MDB algorithm with $\langle h, H\rangle$. The matrix M^*, constructed by the MDB algorithm from the matrices $M^{(1)}$ and $M^{(2)}$ using H, is defined as follows by the first equation and illustrated for $n = 4$ by the second equation:

$$M^*_{i,j} = \begin{cases} x/2, & (i \le n/2) \wedge (j \le n/2) \wedge (i \neq j) \\ x/2, & (i > n/2) \wedge (j > n/2) \wedge (i \neq j) \\ x - \epsilon, & i = j \\ 0, & \text{otherwise} \end{cases}$$

$$M^* = \begin{pmatrix} x-\epsilon & x/2 & 0 & 0 \\ x/2 & x-\epsilon & 0 & 0 \\ 0 & 0 & x-\epsilon & x/2 \\ 0 & 0 & x/2 & x-\epsilon \end{pmatrix}.$$

Since $x - \epsilon > x/2$ for any $\epsilon < x/2$, the mapping processed in the *first* iteration of the MDB algorithm will be the mapping σ_I, corresponding to the main diagonal of M^*, with $\langle f, F\rangle(\sigma_I) = (x-\epsilon)^n$. Also, at the *second* iteration of the MDB algorithm, we have $\tau_{MD} = (x-\epsilon)^{n-1} \cdot (x/2)$. The MDB algorithm would halt at the second iteration if $\tau_{MD} \le \langle f, F\rangle(\sigma_I)$, which holds, for example, for $x = 0.98$ and $\epsilon = 0.01$. Likewise, in the proof of Theorem 1, we have already shown that σ_I is the best mapping with respect to $\langle f, F\rangle$. Hence, the time complexity of the TA algorithm on this problem instance with $K = 1$ is exponentially worse than that of the MDB algorithm (with properly chosen upper bound $\langle h, H\rangle$). □

Theorem 6. *Given a schema metamatching problem instance, the time complexity of the MDB algorithm on this instance can be exponentially worse than this of the TA algorithm.*

Proof. This proof is by example of a corresponding class of schema metamatching problems: On any instance of this problem class, the TA algorithm identifies the best mapping on the first iteration, yet, it will be the *last* mapping discovered by the MDB algorithm.

Consider two algorithms, A_1 and A_2, and a pair of schemata S and S', each consisting of n attributes. Likewise, let the l-aggregator f be the *product* operator, g-aggregator F be the *min* operator, and Σ_Γ be the set of all one-to-one mappings from Σ. Given S and S', the similarity matrices $M^{(1)}$ and $M^{(2)}$, induced by A_1 and A_2, respectively, are defined as follows:

$$M^{(1)}_{i,j} = \begin{cases} \epsilon, & i = j \\ 0, & \text{otherwise} \end{cases}$$
$$M^{(2)}_{i,j} = \begin{cases} 1-3\epsilon, & i = j \\ 1, & \text{otherwise} \end{cases}$$

for an arbitrary $1/3 > \epsilon > 0$. We illustrate such matrices for $n = 4$ as follows:

$$M^{(1)} = \begin{pmatrix} \epsilon & 0 & 0 & 0 \\ 0 & \epsilon & 0 & 0 \\ 0 & 0 & \epsilon & 0 \\ 0 & 0 & 0 & \epsilon \end{pmatrix}$$
$$M^{(2)} = \begin{pmatrix} 1-3\epsilon & 1 & 1 & 1 \\ 1 & 1-3\epsilon & 1 & 1 \\ 1 & 1 & 1-3\epsilon & 1 \\ 1 & 1 & 1 & 1-3\epsilon \end{pmatrix}.$$

Considering the execution of the TA algorithm on $M^{(1)}$ and $M^{(2)}$ as defined above, first, notice that the only mapping σ, for which we have $f(\sigma, M^{(1)}) > 0$, is the mapping σ_I (that is, the identity permutation). Therefore, σ_I will be discovered by the TA algorithm on the *first* iteration, with $\tau_{TA} = \langle f, F\rangle(\sigma_I)$. Second, notice that all the entries of $M^{(1)}$ and $M^{(2)}$ lie in the interval [0, 1]. Thus, for all $\sigma_I \neq \sigma \in \Sigma_\Gamma$, we have $\langle f, F\rangle(\sigma) = 0$. Finally, since $f(\sigma_I, M^{(2)}) > 0$, we have $\langle f, F\rangle(\sigma) > 0$, and thus, σ_I is the best mapping with respect to $\langle f, F\rangle$.

It is not hard to see that the aggregators f and F do not commute on our $M^{(1)}$ and $M^{(2)}$. Consider a pair of functions $\langle h, H\rangle$, where both h and H stand for a simple *average*. Since the entries of both matrices $M^{(1)}$ and $M^{(2)}$ lie in the interval [0, 1], we have $\langle f, F\rangle \prec \langle h, H\rangle$. Likewise, since h and H are (trivially) strongly commutative, we can solve this problem instance using the MDB algorithm with $\langle h, H\rangle$. The matrix M^*, constructed by MDB from the matrices $M^{(1)}$ and $M^{(2)}$ using H, is defined as follows by the first equation and illustrated for $n = 4$ by the second equation:

$$M^*_{i,j} = \begin{cases} \frac{1}{2} - \epsilon, & i = j \\ \frac{1}{2}, & \text{otherwise} \end{cases}$$
$$M^* = \begin{pmatrix} \frac{1}{2}-\epsilon & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2}-\epsilon & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2}-\epsilon & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2}-\epsilon \end{pmatrix}.$$

For each mapping $\sigma \in \Sigma_\Gamma$, let k_σ be the number of attributes $i \in S$, such that $\sigma(i) = i$ (that is, the number of the attribute mappings in σ that lie on the main diagonal of M^*). For each $\sigma \in \Sigma_\Gamma$, we have $k_\sigma \in \{1, 2, \ldots, n-3, n-2, n\}$, and

$$h(\sigma, M^*) = \begin{cases} \frac{1}{2}, & k_\sigma = 0 \\ \frac{1}{2} - \frac{k\epsilon}{n}, & 0 < k_\sigma \leq n-2 \\ \frac{1}{2} - \epsilon, & k_\sigma = n. \end{cases}$$

Therefore, for each $\sigma_I \neq \sigma \in \Sigma$, we have

$$\langle h, H\rangle(\sigma) > \langle h, H\rangle(\sigma_I)$$

and, thus, (the best mapping!) σ_I will be the *last* mapping discovered by the MDB algorithm. □

Theorem 7. *Let $A_1, \ldots, A_m$ be a set of schema matchers with $\langle \vec{f}, F\rangle$ being their l- and g-aggregators. Given a function pair $\langle h, H\rangle$ that both commute and dominate $\langle \vec{f}, F\rangle$ on $M^{(1)}, \ldots, M^{(m)}$, the CrossThreshold algorithm correctly finds the top-K valid mappings with respect to $\langle \vec{f}, F\rangle$.*

Proof. Let Y be the set of mappings as in step 3 of the CrossThreshold algorithm. We only need to show that every mapping $\sigma \in Y$ has a weight at least as high, according to $\langle \vec{f}, F\rangle$, as every mapping $\sigma' \notin Y$. By definition of Y, this is the case for each mapping $\sigma' \notin Y$ that has been seen by the CrossThreshold algorithm. Assume that σ' was not seen, and let τ', τ'_{TA}, and τ'_{MDB} be the value of τ, τ_{TA}, and τ_{MDB}, respectively, at the termination of CrossThreshold. If $\tau'_{MDB} > \tau'_{TA}$, by monotonicity of F, we have $\tau' = \tau'_{TA} \geq \langle \vec{f}, F\rangle(\sigma')$ for every $\sigma' \notin Y$. Otherwise, if $\tau'_{MDB} \leq \tau'_{TA}$, from the incrementality of querying A_i with q-top, we have $\tau' \geq \langle h, H\rangle(\sigma') \geq \langle \vec{f}, F\rangle(\sigma')$ for every $\sigma' \notin Y$. However, by definition of Y, for every $\sigma \in Y$, we have $\langle \vec{f}, F\rangle(\sigma) \geq \tau'$. Therefore, for every $\sigma' \notin Y$, we have $\langle \vec{f}, F\rangle(\sigma) \geq \tau' \geq \langle \vec{f}, F\rangle(\sigma')$, as desired. □

Lemma 8. *If $I < I_{TA}$ and $\tau[I] = \tau_{TA}[I]$, then $Y \setminus Y_{TA} \neq \emptyset$, and for each $\sigma \in Y \setminus Y_{TA}$, we have*

$$\langle \vec{f}, F\rangle(\sigma) = \tau_{TA}[I]. \tag{12}$$

Proof. The assumption of the lemma that $I < I_{TA}$ implies that there exists at least one mapping $\sigma \in Y$ that would have been discovered by (independently running) TA only at some iteration $I' > I$, and thus, we have $Y \setminus Y_{TA} \neq \emptyset$. Now, considering mappings $\sigma \in Y \setminus Y_{TA}$, recall that $\tau_{TA}[I] = F(f^{(1)}(\sigma_1, M^{(1)}), \ldots, f^{(m)}(\sigma_m, M^{(m)}))$, where $\sigma_1, \ldots, \sigma_m$ are mappings provided by $A_1, \ldots, A_m$ at the iteration I, respectively. From the lemma assumption ($\tau[I] = \tau_{TA}[I]$) and (10), we have

$$\begin{aligned} \tau_{TA}[I] &= F(f^{(1)}(\sigma_1, M^{(1)}), \ldots, f^{(m)}(\sigma_m, M^{(m)})) \\ &\leq \langle \vec{f}, F\rangle(\sigma) \\ &= F(f^{(1)}(\sigma, M^{(1)}), \ldots, f^{(m)}(\sigma, M^{(m)})) \end{aligned} \tag{13}$$

for all $\sigma \in Y$. On the other hand, by the definition of the q-top queries, we have $f^{(i)}(\sigma_i, M^{(i)}) \geq f^{(i)}(\sigma, M^{(i)})$ for each mapping σ_i as in (13). Thus, by the monotonicity of F, we have

$$F(f^{(1)}(\sigma_1, M^{(1)}), \ldots, f^{(m)}(\sigma_m, M^{(m)})) \geq F(f^{(1)}(\sigma, M^{(1)}), \ldots, f^{(m)}(\sigma, M^{(m)})), \tag{14}$$

and together, (13) and (14) imply (12). □

Lemma 9. *If $I < I_{MDB}$ and $\tau[I] = \tau_{MDB}[I]$, then $Y \setminus Y_{MDB} \neq \emptyset$, and for each $\sigma \in Y \setminus Y_{MDB}$, we have*

$$\langle \vec{f}, F\rangle(\sigma) = \tau_{MDB}[I]. \tag{15}$$

Proof. Similar to the proof of Lemma 8, the assumption that $I < I_{MDB}$ implies that there exists at least one mapping $\sigma \in Y$ that would have been discovered by (independently running) MDB only at some iteration $I' > I$, and thus, we have $Y \setminus Y_{MDB} \neq \emptyset$. Considering such mappings $\sigma \in Y \setminus Y_{MDB}$, from the assumption that $\tau[I] = \tau_{MDB}[I]$, we have

$$\tau_{MDB}[I] = \langle h, H\rangle(\sigma') \leq \langle \vec{f}, F\rangle(\sigma), \tag{16}$$

where σ' is the mapping discovered by the MDB algorithm at the iteration I. Likewise, since $\sigma \in Y \setminus Y_{MDB}$, we have

$$\langle h, H\rangle(\sigma) \leq \langle h, H\rangle(\sigma'). \tag{17}$$

Finally, since $\langle h, H\rangle$ dominates $\langle \vec{f}, F\rangle$, we have

$$\langle \vec{f}, F\rangle(\sigma) \leq \langle h, H\rangle(\sigma). \tag{18}$$

Together, (16), (17), and (18) provide us with the lemma claim that $\langle \vec{f}, F\rangle(\sigma) = \tau_{MDB}[I]$. □

Theorem 10. *There exist schema metamatching problem instances for which the time complexity of both the TA and MDB algorithms is exponentially worse than that of the CrossThreshold algorithm.*

Proof. The proof is by example of the corresponding problem instance. Consider two algorithms, A_1 and A_2, and a pair of schemata S and S', each consisting of n attributes. Likewise, let the l-aggregator f be the regular *product*, and the g-aggregator F be the utilitarian aggregator *min*. Given S and S', the similarity matrices $M^{(1)}$ and $M^{(2)}$, induced by A_1 and A_2, respectively, are given as follows:

$$M^{(1)}_{i,j} = \begin{cases} x + 2\epsilon, & (i = j) \vee (i + j < n + 1) \\ 0, & \text{otherwise} \end{cases}$$
$$M^{(2)}_{i,j} = \begin{cases} x, & (i = j) \vee (i + j > n + 1) \\ 0, & \text{otherwise} \end{cases}$$

such that $x > 0$, $\epsilon > 0$, and $x + 2\epsilon < 1$. We illustrate the matrices for $n = 5$ as follows:

$$M^{(1)} = \begin{pmatrix} x+2\epsilon & x+2\epsilon & x+2\epsilon & x+2\epsilon & 0 \\ x+2\epsilon & x+2\epsilon & x+2\epsilon & 0 & 0 \\ x+2\epsilon & x+2\epsilon & x+2\epsilon & 0 & 0 \\ x+2\epsilon & 0 & 0 & x+2\epsilon & 0 \\ 0 & 0 & 0 & 0 & x+2\epsilon \end{pmatrix}$$
$$M^{(2)} = \begin{pmatrix} x & 0 & 0 & 0 & 0 \\ 0 & x & 0 & 0 & x \\ 0 & 0 & x & x & x \\ 0 & 0 & x & x & x \\ 0 & x & x & x & x \end{pmatrix}.$$

Likewise, consider a pair of bounding functions $\langle h, H\rangle$, where both h and H stand for *average*. Since the entries of both matrices $M^{(1)}$ and $M^{(2)}$ lie in the interval [0, 1], we have $\langle f, F\rangle \prec \langle h, H\rangle$. The matrix M^*, constructed by the MDB algorithm from the matrices $M^{(1)}$ and $M^{(2)}$ using H, is defined as follows by the first equation and illustrated for $n = 5$ by the second equation:

$$M^*_{i,j} = \begin{cases} x + \epsilon & i = j \\ \frac{x}{2} + \epsilon, & (i \neq j) \wedge (i + j < n + 1) \\ 0, & i + j = n + 1 \\ \frac{x}{2}, & \text{otherwise} \end{cases}$$
$$M^* = \begin{pmatrix} \mathbf{x}+\epsilon & \frac{x}{2}+\epsilon & \frac{x}{2}+\epsilon & \frac{x}{2}+\epsilon & \mathbf{0} \\ \frac{x}{2}+\epsilon & \mathbf{x}+\epsilon & \frac{x}{2}+\epsilon & \mathbf{0} & \frac{x}{2} \\ \frac{x}{2}+\epsilon & \frac{x}{2}+\epsilon & \mathbf{x}+\epsilon & \frac{x}{2} & \frac{x}{2} \\ \frac{x}{2}+\epsilon & \mathbf{0} & \frac{x}{2} & \mathbf{x}+\epsilon & \frac{x}{2} \\ \mathbf{0} & \frac{x}{2} & \frac{x}{2} & \frac{x}{2} & \mathbf{x}+\epsilon \end{pmatrix}.$$

First, consider the execution of the TA algorithm on this problem instance. Let σ_I stand for the mappings captured by the primary diagonal. That is, for $1 \leq i \leq n$, $\sigma_I(i) = i$. It is not hard to see that

$$\langle f, F\rangle(\sigma_I) = \min\{(x + 2\epsilon)^n, x^n\} = x^n$$

whereas, for each mapping $\sigma \neq \sigma_I$, we have either $f(\sigma, M^{(1)}) = 0$ or $f(\sigma, M^{(2)}) = 0$, and thus, $\langle f, F\rangle(\sigma) = 0$. Hence, the top-1 mapping for this problem instance cannot be anything but $\{\sigma_I\}$.

On the other hand, $M^{(1)}$ induces $\Theta((\frac{n}{2} - 1)!)$ mappings σ having

$$f\left(\sigma, M^{(1)}\right) = (x + 2\epsilon)^n = f\left(\sigma_I, M^{(1)}\right)$$

and $M^{(2)}$ induces $\Theta((\frac{n}{2} - 1)!)$ mappings σ having

$$f\left(\sigma, M^{(2)}\right) = x^n = f\left(\sigma_I, M^{(2)}\right).$$

Therefore, the best mappings σ_I might be discovered by the TA algorithm only after $\Theta((\frac{n}{2} - 1)!)$ iterations.

In turn, consider the performance of the MDB algorithm on this problem instance and further assume that $x^n < (x + \epsilon)/n$. From the description of M^*, it is not hard to see that the best mapping σ_I will be discovered by the MDB algorithm on the first iteration. However, observe that the lowest value obtained by τ_{MDB} on M^* will be higher than $(x + \epsilon)/n$. Since $\langle f, F\rangle(\sigma_I) = x^n$, we conclude that the MDB algorithm could *verify* that the candidate σ_I is indeed the best mappings only after $\Theta(n!)$ iterations.

Now, consider the "cooperative" execution of TA and MDB in the scope of the CrossThreshold algorithm. Following our discussion above, assume that TA would fail to discover σ_I for the first $\Theta((\frac{n}{2}-1)!)$ iterations. However, immediately after the first iteration, we have $\tau_{TA} = x^n$. Recall that σ_I is discovered by the MDB algorithm at the first iteration. It is easy to see that after the first iteration of the CrossThreshold algorithm, we have $\tau = \tau_{TA}$ and, thus, we immediately conclude that

$$\langle f, F\rangle(\sigma) = x^n = \tau.$$

Hence, the best mapping σ_I is discovered by the CrossThreshold algorithm immediately after the first iteration, while both I_{TA} and I_{MDB} for this top-1 problem are exponential in n. □

Acknowledgments

The authors would like to thank Ilan Shimshoni for the useful discussions. They also would like to thank the anonymous reviewers for their useful comments. The work of C. Domshlak was partially supported by the Intelligent Information Systems Institute, Cornell University (AFOSR Grant F49620-01-1-0076). The work of A. Gal was partially supported by Technion V.P.R. Fund—E. and J. Bishop Research Fund, the Fund for the Promotion of Research at Technion, and the IBM Faculty Award for 2003/2004 on "Self-Configuration in Autonomic Computing Using Knowledge Management." The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the US Government.

References

[1] J. Berlin and A. Motro, "Autoplex: Automated Discovery of Content for Virtual Databases," *Proc. Ninth Int'l Conf. Cooperative Information Systems (CoopIS '01),* C. Batini, Giunchiglia, P. Giorgini, and M. Mecella, eds., pp. 108-122, 2001.
[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific Am.,* May 2001.
[3] P.A. Bernstein and S. Melnik, "Meta Data Management," *Proc. IEEE CS Int'l Conf. Data Eng.,* 2004.
[4] A. Bilke and F. Naumann, "Schema Matching Using Duplicates," *Proc. IEEE CS Int'l Conf. Data Eng.,* pp. 69-80, 2005.
[5] B. Convent, "Unsolvable Problems Related to the View Integration Approach," *Proc. Int'l Conf. Database Theory (ICDT '86),* G. Goos and J. Hartmanis, eds., pp. 141-156, 1986.
[6] R. Dhamankar, Y. Lee, A. Doan, A.Y. Halevy, and P. Domingos, "iMAP: Discovering Complex Mappings between Database Schemas," *Proc. ACM Special Interest Group of Management of Data (SIGMOD) Conf. Management of Data,* pp. 383-394, 2004.
[7] H. Do, S. Melnik, and E. Rahm, "Comparison of Schema Matching Evaluations," *Proc. Second Int'l Workshop Web Databases (German Informatics Soc. '02),* 2002.
[8] H.H. Do and E. Rahm, "COMA—A System for Flexible Combination of Schema Matching Approaches," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* pp. 610-621, 2002.
[9] A. Doan, P. Domingos, and A.Y. Halevy, "Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach," *Proc. ACM Special Interest Group of Management of Data (SIGMOD) Conf. Management of Data,* W.G. Aref, ed., 2001.
[10] A. Doan, J. Madhavan, P. Domingos, and A. Halevy, "Learning to Map between Ontologies on the Semantic Web," *Proc. 11th Int'l Conf. World Wide Web,* pp. 662-673, 2002.
[11] A. Doan, N.F. Noy, and A.Y. Halevy, "Introduction to the Special Issue on Semantic Integration," *SIGMOD Record,* vol. 33, no. 4, pp. 11-13, 2004.
[12] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank Aggregation Methods for the Web," *Proc. 10th Int'l World Wide Web Conf. (WWW '01),* pp. 613-622, 2001.
[13] D.W. Embley, D. Jackman, and L. Xu, "Attribute Match Discovery in Information Integration: Exploiting Multiple Facets of Metadata," *J. Brazilian Computing Soc.,* vol. 8, no. 2, pp. 32-43, 2002.
[14] R. Fagin, R. Kumar, and D. Sivakumar, "Comparing Top-K Lists," *Soc. Industrial and Applied Math. (SIAM) J. Discrete Math.,* vol. 17, no. 1, pp. 134-160, 2003.
[15] R. Fagin, R. Kumar, and D. Sivakumar, "Efficient Similarity Search and Classification via Rank Aggregation," *Proc. ACM Special Interest Group on Modification of Data (SIGMOD) Conf. Management of Data,* pp. 301-312, 2003.
[16] R. Fagin, A. Lotem, and M. Naor, "Optimal Aggregation Algorithms for Middleware," *Proc. ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems (PODS),* 2001.
[17] R. Fagin, A. Lotem, and M. Naor, "Optimal Aggregation Algorithms for Middleware," *J. Computer and System Sciences,* vol. 66, pp. 614-656, 2003.
[18] G.H.L. Fletcher and C.M. Wyss, "Data Mapping as Search," *Advances in Database Technology—Proc. 10th Int'l Conf. Extending Database Technology (EDBT '06),* pp. 95-111, 2006.
[19] N. Fridman Noy and M.A. Musen, "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment," *Proc. 17th Nat'l Conf. Artificial Intelligence (AAAI '00),* pp. 450-455, 2000.
[20] A. Gal, "On the Cardinality of Schema Matching," *Proc. IFIP WG 2.12 and WG 12.4 Int'l Workshop Web Semantics (SWWS '05),* pp. 947-956, 2005.
[21] A. Gal, "Managing Uncertainty in Schema Matching with Top-K Schema Mappings," *J. Data Semantics,* vol. 6, pp. 90-114, 2006.
[22] A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi, "A Framework for Modeling and Evaluating Automatic Semantic Reconciliation," *Very Large Databases (VLDB) J.,* vol. 14, no. 1, pp. 50-67, 2005.
[23] A. Gal, G. Modica, H.M. Jamil, and A. Eyal, "Automatic Ontology Matching Using Application Semantics," *AI Magazine,* vol. 26, no. 1, 2005.
[24] H.W. Hamacher and M. Queyranne, "K-Best Solutions to Combinatorial Optimization Problems," *Annals of Operations Research,* vol. 4, pp. 123-143, 1985/6.
[25] B. He and K. Chen-Chuan Chang, "Statistical Schema Matching Across Web Query Interfaces," *Proc. ACM Special Interest Group on Management of Data (SIGMOD) Conf. Management of Data,* pp. 217-228, 2003.
[26] B. He and K.C.-C. Chang, "Making Holistic Schema Matching Robust: An Ensemble Approach," *Proc. 11th ACM Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) Int'l Conf. Knowledge Discovery and Data Mining,* pp. 429-438, 2005.
[27] A. Heß and N. Kushmerick, "Learning to Attach Semantic Metadata to Web Services," *Proc. Second Semantic Web Conf.,* 2003.
[28] R. Hull, "Managing Semantic Heterogeneity in Databases: A Theoretical Perspective," *Proc. ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems (PODS),* pp. 51-61, 1997.
[29] G. Koifman, A. Gal, and O. Shehory, "Schema Mapping Verification," *Proc. Very Large Databases (VLDB '04) Workshop Information Integration on the Web,* H. Davulcu and N. Kushmerick, eds., pp. 52-57, 2004.
[30] J. Madhavan, P.A. Bernstein, P. Domingos, and A.Y. Halevy, "Representing and Reasoning about Mappings between Domain Models," *Proc. 18th Nat'l Conf. Artificial Intelligence and 14th Conf. Innovative Applications of Artificial Intelligence (AAAI/IAAI '02),* pp. 80-86, 2002.
[31] J. Madhavan, P.A. Bernstein, and E. Rahm, "Generic Schema Matching with Cupid," *Proc. Int'l Conf. Very Large Data Bases (VLDB '01),* pp. 49-58, Sept. 2001.
[32] S. Melnik, *Generic Model Management: Concepts and Algorithms.* Springer, 2004.
[33] S. Melnik, H. Garcia-Molina, and E. Rahm, "Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching," *Proc. IEEE CS Int'l Conf. Data Eng.,* pp. 117-140, 2002.

[34] S. Melnik, E. Rahm, and P.A. Bernstein, "Rondo: A Programming Platform for Generic Model Management," *Proc. ACM Special Interest Group on Management of Data (SIGMOD) Conf. Management of Data,* pp. 193-204, 2003.
[35] R.J. Miller, L.M. Haas, and M.A. Hernández, "Schema Mapping as Query Discovery," *Proc. Int'l Conf. Very Large Data Bases (VLDB '00),* A. El Abbadi, M.L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, eds., pp. 77-88, 2000.
[36] R.J. Miller, M.A. Hernández, L.M. Haas, L.-L. Yan, C.T.H. Ho, R. Fagin, and L. Popa, "The Clio Project: Managing Heterogeneity," *SIGMOD Record,* vol. 30, no. 1, pp. 78-83, 2001.
[37] G. Modica, A. Gal, and H. Jamil, "The Use of Machine-Generated Ontologies in Dynamic Information Seeking," *Proc. Ninth Int'l Conf. Cooperative Information Systems (CoopIS '01),* C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, eds., pp. 433-448, 2001.
[38] P. Mork, A. Rosenthal, L.J. Seligman, J. Korb, and K. Samuel, "Integration Workbench: Integrating Schema Integration Tools," *Proc. 22nd Int'l Conf. Data Eng. Workshops (ICDE '06),* p. 3, 2006.
[39] N. Friedman Noy, A. Doan, and A.Y. Halevy, "Semantic Integration," *AI Magazine,* vol. 26, no. 1, pp. 7-10, 2005.
[40] M. Pascoal, M.E. Captivo, and J. Cl'imaco, "A Note on a New Variant of Murty's Ranking Assignments Algorithm," *4OR: Quarterly J. the Belgian, French and Italian Operations Research Soc.,* vol. 1, no. 3, pp. 243-255, 2003.
[41] E. Rahm and P.A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," *Very Large Databases (VLDB) J.,* vol. 10, no. 4, pp. 334-350, 2001.
[42] P. Rodriguez-Gianolli and J. Mylopoulos, "A Semantic Approach to XML-Based Data Integration," *Proc. Int'l Conf. Conceptual Modelling (ER '01),* pp. 117-132, 2001.
[43] B. Srivastava and J. Koehler, "Web Service Composition—Current Solutions and Open Problems," *Proc. Int'l Conf. Automated Planning and Scheduling (ICAPS '03) Workshop Planning for Web Services,* 2003.
[44] L. Xu and D.W. Embley, "A Composite Approach to Automating Direct and Indirect Schema Mappings," *Information Systems,* vol. 39, no. 8, pp. 657-886, Dec. 2006.

Carmel Domshlak received the PhD degree in computer science from Ben-Gurion University of the Negev, Israel, in 2002 for his work on preference representation models. He is a member of the Faculty of Industrial Engineering and Management, Technion—Israel Institute of Technology, Israel. His research interests include synthesis of knowledge representation and computational techniques across the field of AI and, in particular, in modeling and reasoning about preferences, planning, and probabilistic reasoning. He is a member of the editorial board of the *Journal of Artificial Intelligence Research*.

Avigdor Gal received the DSc degree in temporal active databases from the Technion—Israel Institute of Technology in 1995. He is a member of the Faculty of Industrial Engineering and Management, Technion. He has published more than 50 papers in journals, books, and conferences on data integration, temporal databases, information systems architectures, and active databases. He is a member of the steering committee of IFCIS, a member of IFIP WG 2.6, and a recipient of the IBM Faculty Award for 2002–2004. He is a member of the ACM and a senior member of the IEEE and the IEEE Computer Society.

Haggai Roitman received the BSc degree in information systems engineering (*cum laude*) from Technion in 2004. He is currently a PhD candidate at the Faculty of Industrial Engineering and Management, Technion. His primary interests are in the areas of Web resource monitoring and online data delivery.