THE NETWORK IS THE BUSINESS

D2.1 – Micro-mapping based schema matching in the NisB network 1.0

Author:	Avigdor Gal – IIT
Contributors:	Karl Aberer – EPFL, Michael Katz – IIT, Eliezer Levy – SAP, Zoltán Miklós – EPFL, Nguyen Quoc Viet Hung – EPFL, Tomer Sagi – IIT, Victor Shafran – SAP
Dissemination:	Public
Contributing to:	WP 2
Date:	10/30/2011
Revision:	V1.1

NISB CONSORTIUM CONTACTS

Organization	Name	Phone	E-Mail
SAP	Victor Shafran	+972-52-3854883	victor.shafran@sap.com
IIT	Avigdor Gal	+972-54-5370811	avigal@ie.technion.ac.il
EPFL	Miklós Zoltán	+41 79 723 3682	zoltan.miklos@epfl.ch
HSG	Boris Otto	+41 79 219 0582	boris.otto@unisg.ch
TXT	Enrico Del Grosso	+39 02 25771230	enrico.delgrosso@txt.it
CRF	Giorgio Sobrito	+39 011 9080542	giorgio.sobrito@crf.it
Momentum	Ian Graham	+44-2890-450101	ian.Graham@momentumni.org

CONTENTS

List of Figures

PRELIMINARIES

This document is a report, due at M14, regarding the micro-mapping based schema matching in the NisB network. The scope of this deliverable is to report on a model for schema cover that was created and tests within the NisB context. Based on this report, the Consortium have already started implementing buidling constructs of the NisB application. This document will be updated in M20 with a more complete information and more thorough evaluation. The main contents of this deliverable include the following:

- Introduction to the generalized cover problem
- Preliminaries on schema matching and interoperability.
- A model for the generalized cover problem
- A formal description of the model
- A set of algorithms for solving cover problems
- An empirical evaluation
- Related work

The content of the deliverable is based on a scientific article that was submitted to the International Conference on Data Engineering 2012 and rejected. The term *micro-mapping* is in the process of being formally defined as part of the NisB project (see D1.3 for a first attempt). In this document, we refrain from using it before it will be fully formed. The relevant part of a micro-mapping in the cover problem is a valid matching between a set of subschemata and a set of concepts. Formally, it is marked as $\sigma(t, c)$.

I. Introduction

Semantic interoperability among heterogeneous information systems is a critical problem for modern business networks. Data integration is considered one of the main challenges in establishing such interoperability, due to the need to provide correct interpretation of data. [1], [2], [3] In today's world of connected businesses, the idea of approaching the data integration challenge with methods based on reuse and collaboration become feasible. Such reuse can be based on a repository of information building blocks, referred to as *concepts* [4]. Documents, modeled as schemas are mapped against a set of concepts, representative of entities in the domain of discourse (*e.g.*, a vendor concept in an eCommerce domain), in a process termed *schema cover*. The matched concepts provide interpretation through interoperability paths to other schemas. The idea is to cover a schema by concepts and thereby interpret the schema in terms of known concepts. This way, the schema is integrated into an existing body of information and knowledge. For example, consider a network of enterprises that exchange business documents and cooperate to establish interoperability in order to conduct business and generate value from the network. The business documents do not follow a common format or standard (although they come from the same domain, *e.g.*, sales and purchase orders). The aim of schema cover is to integrate different vocabulary and structural elements, representing similar or even identical real-world entities, by using concepts.

Schema cover builds upon the research work in schema matching [5], a basic data integration process that provides correspondences between heterogeneous schema elements. Schema matching starts with generating attribute correspondences by matching attributes from different schemas using various matching techniques and associating a degree of similarity to each pair. Then, the process of schema matching adds schema-level constraints such as $1:1$ matching to identify a set of attribute correspondences that both maximizes similarity and satisfies constraints. This process is extended to schema cover by matching parts of schemas (called subschemas) with concepts, and then adding cover-level constraints, called *ambiguity* [4] to identify a set of concepts that optimizes both similarity and ambiguity. Ambiguity constraints, to be formally defined in this work along with subschemas and concepts, represent the number of times an attribute is interpreted by various concepts.

To illustrate the role of concepts in improving the quality of schema matching and thus, interoperability, consider Figure 1. The figure illustrates a schema with three attributes (on the left) and three concepts (on the right). In the absence of concepts (Figure 1(top)), the matching task can select attributes without any restriction. However, in the presence of concepts (Figure 1(bottom)), a matcher is guided to choose closely related attributes. Our hypothesis is that well-designed concepts can improve the quality of the matching for exactly this argument.

To motivate this research and highlight the diverse tasks of reuse in resolving semantic heterogeneity, we illustrate the importance of schema cover in the context of NisB. NisB's goal is to promote agile business networks by leveraging and reusing past data integration experiences of the networked community of businesses. In one typical scenario in NisB, a new schema has to be decomposed and mapped against a concepts repository that represents previous data integration efforts in the community. The underlying NisB system attempts to interpret the new schema in terms of the concepts in the repository. That is, the system decomposes the new schema into subschemas and maps its subschemas to concepts that already have been mapped and therefore, are already understood. In this case, it is beneficial to cover the new schema by overlapping concepts, yielding high degree of ambiguity, yet leading to various opportunities of interoperating with other schemas of other partners in the community.

In a different scenario in NisB, the system attempts to create a specific interoperability path between two given schemas using the concept repository that represents the knowledge accumulated so far. In this case, there is a need to create a very specific cover of the source schema with minimum ambiguity.

In this document we propose a generalized schema cover model where the above scenarios can be handled, among other scenarios, and to which the framework of [4] is a special case. We analyze the theoretical complexity of various variations of the cover problem, showing it to be a hard problem in general and show a set of heuristics.

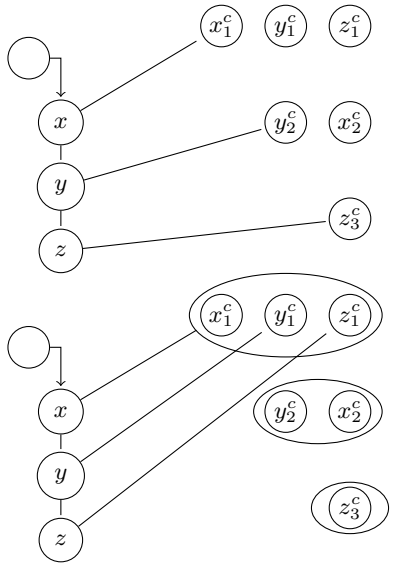

Fig. 1: Role of concepts illustrated

We provide a thorough empirical analysis, using both real-world and simulated data sets, to assess the effectiveness of various cover formulations, showing that some formulations yield more effective solutions than others. We also show empirically the limits of an integer linear programming (ILP) solution as an exhaustive search solution compared with the quality of outcome of our heuristics.

The novelty of this work can be summarized as follows:

- We provide a general framework for schema cover, offering a space of schema cover problems.
- We propose a new formulation of a cover problem and show it to be NP-Hard. The new formulation is shown empirically to be more effective than the one proposed in [4].
- We offer ILP formulation for solving certain special variations of the cover problem. These formulations are shown empirically to scale well to schemas with 1,000 attributes and cover repository with 8,000 concepts.
- We offer new heuristics to support efficient cover computation. The heuristics are shown to trade off performance with optimization.
- We provide a thorough empirical analysis of the proposed algorithmic solutions to cover problems.

The rest of the document is organized as follows. Preliminaries are given in Section II and Section III provides a model of the generalized cover. A set of cover problems are presented in Section IV, together with a theoretical analysis, followed by an ILP formulation and a set of heuristics in Section V. Section VI provides our empirical analysis. We conclude in a description of related work and directions for further research.

II. Interoperability and Schema Matching

In this work, we assume that a new schema is matched with a (possibly large) repository of concepts. We focus on the scenario where the new schema is matched against the concept repository as a first step for establishing interoperability between schemas. The schema is decomposed into subschemas and at the same time the concept repository is filtered in order to select a subset of concepts that potentially match. Concepts and subschemas are then coupled using schema matching techniques, followed by a cover selection. Figure 2 provides an illustration of this process, which is part of the design of the NisB system. While the cover selection process is the main focus

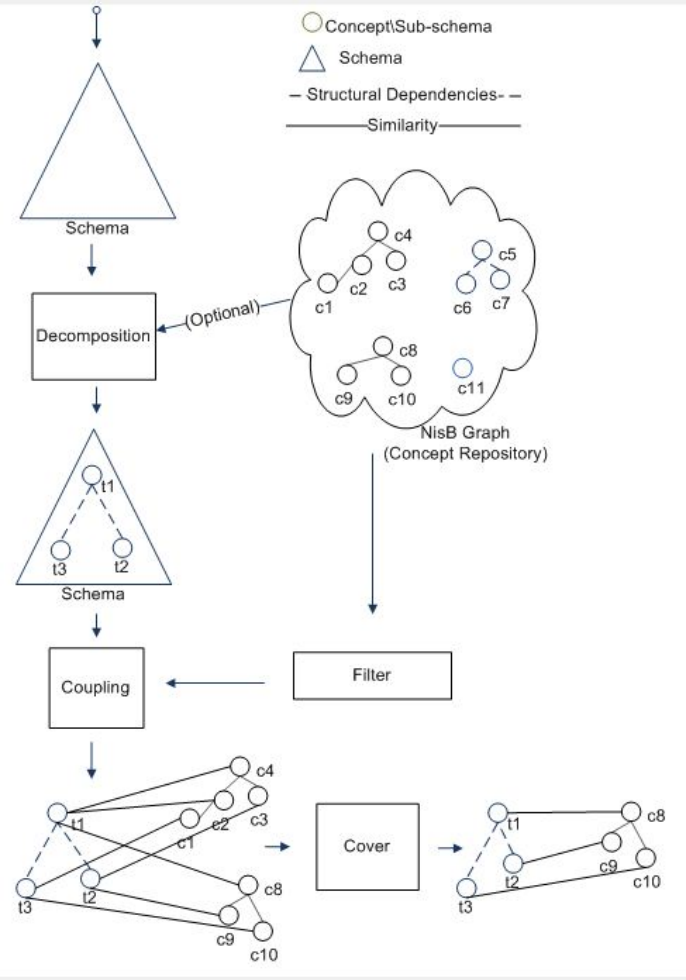

Fig. 2: The NisB Process

of this work, we first give an informal discussion of the overall encompassing process that includes decomposition, coupling, filtering and cover.

Schema decomposition is a process in which a schema is broken into subschemas. We set no particular restrictions on this process, and therefore attributes can belong to more than one schema, a subschema (modeled often as a graph) does not have to be a connected component, *etc.* Decomposition can be performed in one of two ways, namely *native* and by *concept-first*. Native decomposition is performed independently of the concept repository. For example, a method for decomposing an underlying ER model was proposed by [6]. Alternatively, decomposition by concept-first is guided by a set of concepts using schema matching techniques. Therefore, a schema matcher such as COMA++ [7] or OntoBuilder's Top-K algorithm [8] can determine the best matchings between a concept and the schema. All the schema attributes that participate in such a matching are considered part of a subschema.

To make the subsequent pairing and covering process more efficient the concept repository can be filtered by selecting representatives of concept clusters. Filtering was advocated in [4] as a heuristic to confront the high complexity of the cover problem. In this work, we show that the use of Integer Linear Programming with state-of-

the-art solvers can do without filtering even for very large concept repositories. Nevertheless, filtering can also serve to focus the cover solutions on relevant concepts. Therefore, we can filter-in clusters for which a representative concept is identified as a good candidate for covering.

Coupling is a schema matching process in which the similarity between a concept and a subschema is assessed. If decomposition is performed by concept-first, then coupling trivially becomes the outcome of the matching discussed above. Otherwise, the same schema matching techniques are applied to construct a set of concept-subschema pairs for the cover process.

In the remainder of this section we provide a model of schema matching based on [8]. We shall use a car rental domain example to demonstrate our model, where a set of concepts may include details about cars, customers, drivers, payments, pickup, return, and stations. The example attempts to match a schema of a car rental portal to these concepts.

A. Schema Matching

Let *schema* $s = \{a_1, a_2, ..., a_n\}$ be a finite set of *attributes*. Attributes can be both simple and compound and compound attributes should not necessarily be disjoint. An attribute in a schema of a car rental portal might be carType, firstName, *etc*. A compound attribute might be pickUpDate, combining two other attributes – eDay, and eMonth. In what follows, given a schema s with n attributes, we shall use the attribute indices $(1, 2, ..., n)$ to identify the attributes of s.

Let s and s' be schemas with n and n' attributes, respectively. Let $\mathcal{S} = s \times s'$ be the set of all possible *attribute correspondences* between s and s'. $\mathcal{S}$ is a set of attribute pairs (*e.g.*, (puDate, PickUpDate)). Let $m(s, s')$ be an $n \times n'$ *similarity matrix* over $\mathcal{S}$, where $m_{i,j}(s, s')$ represents a degree of similarity between the i-th attribute of s and the j-th attribute of s'. Whenever the schemas are known from the context we use $m_{i,j}$ instead of $m_{i,j}(s, s')$. The majority of works in the schema matching literature define $m_{i,j}$ to be a real number in $[0, 1]$. For example, Table I represents a simplified similarity matrix of the running case study. Schema s has the attributes group (representing car group), seat (referring to child's safety seat), xDriver (representing an extra driver), puDate (for pickup date) and rDate (for return date). For schema s' the attributes carType, pickUpDate and returnDate are self explanatory. The attribute options is a compound attribute with two sub-attributes chkBaby and chkExtraDriver, each a binary attribute. It is worth noting that the matcher, in this case, has identified chkBaby as a perfect match to seat and xDriver as a perfect match to chkExtraDriver, propagating these scores to the matching of seat and xDriver with options.

Similarity matrices are generated by *schema matchers*, instantiations of the schema matching process. They differ mainly in the measures of similarity they employ, which yield different similarity matrices. These measures can be arbitrarily complex, and may use various techniques. Some matchers assume similar attributes are more likely to have similar names [9], [10]. Other matchers assume similar attributes share similar domains [11], [12]. Others yet take instance similarity as an indication of attribute similarity [13], [14]. Finally, some researchers use the experience of previous matchings as indicators of attribute similarity [15], [10].

TABLE I: A Similarity Matrix Example

$s \longrightarrow$ / $\downarrow s'$	1 group	2 seat	3 xDriver	4 puDate	5 rDate
1 carType	0.843	0.323	0.323	0.317	0.302
2 options	0.290	1.000	1.000	0.326	0.303
3 pickUpDate	0.344	0.328	0.328	0.351	0.352
4 returnDate	0.312	0.310	0.310	0.359	0.356

Given two schemas, s and s', let the power-set $\Sigma_{s,s'} = 2^{\mathcal{S}}$ be the set of all possible *schema matchings* between the schema pair (s, s'), where a schema matching $\sigma(s, s') \in \Sigma_{s,s'}$ is a set of attribute correspondences (and thus $\sigma(s, s') \subseteq \mathcal{S}$). Whenever the schema pair is known from the context, we refer to a matching simply as σ and to power-set of matchings as Σ. It is worth noting that σ does not necessarily contain all attributes in s or s'. Therefore, there may exist an attribute $a \in s$, such that for all $a' \in s'$, $(a, a') \notin \sigma$. For convenience, we denote by $\bar{\sigma} = \{a \in s \,|\forall a' \in s', (a, a') \notin \sigma\} \cup \{a' \in s' \,|\forall a \in s, (a, a') \notin \sigma\}$ the set of all attributes that do not participate in a schema matching.

B. Similarity and constraints

A schema matching is assigned a similarity measure that is aggregated from the similarity measures of its attribute correspondences. In the literature, such an aggregation took various forms, including the use of summation and the use of fuzzy aggregate operators [16] that includes, among others, the aggregate functions of scaled summation (such as $average$), $\max$, and $\min$.

For the remainder of this work, f represents any of the common linear aggregate operators for schema matching. Given a non-empty schema matching σ between two schemas s and s', we can define a schema matching similarity measure $M_\sigma(s, s')$ to be:

$$M_\sigma(s, s') = f\left(\left\{m_{i,j}(s, s') \mid (a_i, a_j) \in \sigma\right\}\right). \tag{1}$$

For example, consider Table I and assume that the schema matching σ involves matching

$$\{(1,1), (2,2), (3,3), (4,3), (5,4)\}.$$

Also, let $f = average$, then $M_\sigma(s, s') = 0.71$.

In the special case, where both s and s' are singletons, we are back to the attribute correspondence similarity measure.

Let $\Gamma : \Sigma \to \{0, 1\}$ be a boolean constraint function that captures the application-specific constraints on schema matchings, *e.g.*, cardinality constraints and inter-attribute correspondence constraints. Γ partitions Σ into two sets, where the set of all *valid* schema matchings in Σ is given by $\Sigma^\Gamma = \{\sigma \in \Sigma \mid \Gamma(\sigma) = 1\}$. Γ is a general constraint model, where $\Gamma(\sigma) = 1$ means that the matching σ can be accepted by a designer. Γ has been modeled in the literature using special types of matchers called *constraint enforcers* [17].

The input to the process of schema matching is given by two schemata s and s' and Γ. The output of the schema matching process is a *schema matching* $\sigma \in \Sigma^\Gamma$.

III. Model

We now present our model of schema cover, including subschemas, concepts (Section III-A), and covers (Section III-B).

A. Subschemas and concepts

Let $T_s = \{t_1, t_2, ..., t_m\}$ be a set of *subschemas* of s, $t_i \subseteq s$ for all $i = 1, 2, ..., m$. A subschema contains a subset of the attributes of s. For example, a subschema of a car rental portal may include the attributes carType, options and insurance, the first two already discussed above and the third referring to the type of insurance needed for this car group. In what follows, we keep the original indexes of s to represent the attributes from s that are present in t. For example, $t = \{a_1, a_5, a_8\}$ is a subschema of s that contains three of the attributes of s, namely a_1, a_5, and a_8.

Let $C = \{c_1, c_2, ..., c_p\}$ be a set of concepts, where a concept is a schema by itself. For example, in the car rental domain, a concept repository contains the following concepts: CarDetails, CustomerDetails, DriverDetails, PaymentDetails, PickUpDetails, ReturnDetails, and Station.

We note that schemas, subschemas, and concepts are all defined to consist of a set of attributes, however with different semantics. Concepts are schemas whose meaning is assumed to be known in a given business context. Such concepts can be prepared by an enterprise for internal standardization purposes. Alternatively, it can be generated and maintained by organizations such as schema.org. A schema represents a new, unknown set of attributes, that is a candidate for a matching task. For clarity sake, we differentiate schema attributes from concept attributes and denote by a_j^c the j-th attribute of concept c.

B. Covers

We are now ready to introduce covers. A cover is defined in this document as any set of concepts that match a given schema. Therefore, if a concept interprets part of a schema, a cover interprets a schema as a whole. We provide a formal definition of a cover, demonstrating it with our case study example, and explain the role of ambiguity as a quality measure for a cover.

TABLE II: A Cover Example

Concept	Matching subschema
CarDetails: group, seat, insuranceTypeRequested...	carType, options, insurance...
CustomerDetails: address, addressCity, addressCountry...	address,city,country...
DriverDetails: ageGroup, email, license...	
PaymentDetails: CCV, card, cardExpiryMonth...	
PickUpDetails: date, time, stationID...	pickUpDate, pickUpTime, location...
ReturnDetails: date, time, stationID...	returnDate, returnTime, location...
Station: ID, name, location...	

Given a set of subschemas T_s of s, a set C of concepts, and a constraint function Γ for each $t \in T_s, c \in C$, we define a set of valid matchings between subschemas and concepts

$$\mathcal{E}(T_s, C) = \left\{ \sigma(t,c) \,\middle|\, t \in T_s, c \in C, \sigma(t,c) \in \Sigma_{t,c}^{\Gamma} \right\}.$$

Definition 1 *A* cover *of s by C, $v_{s,C} \subseteq \mathcal{E}(T_s, C)$ is a subset of valid matchings between T_s and C.*

A cover is a set of pairs, where each pair in the set is a matching between a subschema and a concept. Let $\sigma = \sigma(t,c) \in v_{s,C}$ be an element of a cover. We define the ambiguity vector $\bar{\lambda}_\sigma = (\lambda_{\sigma,1}, \lambda_{\sigma,2}, ...\lambda_{\sigma,n})$ as follows:

$$\lambda_{\sigma,i} = \begin{cases} 0 & a_i \in \bar{\sigma} \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

Each element of the vector $\bar{\lambda}_\sigma$ is an indicator, set to 1 if attribute a_i is part of the matching σ and 0 otherwise (recall that $\bar{\sigma}$ represents all the attributes not in σ).

To illustrate, a partial sample cover is given in Table II. Four concepts are used to cover a schema of a car rental portal where pickup and return location are always the same.[1] Among the schema attributes in the table, the attribute location is "covered" twice, by the concepts PickUpDetails and ReturnDetalis, and therefore the relevant entry of the attribute location in $\bar{\lambda}_\sigma$ for each of the two concepts is assigned with the value of 1.

[1]This has been the case for many years with the bidding site priceline.com

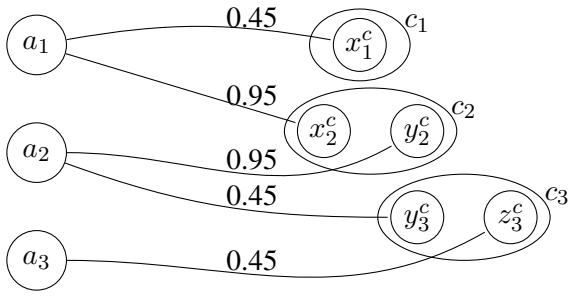

Fig. 3: Illustration of a Cover

The vector $\bar{\lambda}_\sigma$ serves as a quality measure for a matching $\sigma_{t,c}$. It offers a specific angle of the matching quality, observing the impact of the matching on each of the attributes in the original schema.

Given a cover $v = v_{s,C}$ between a schema s and a set of concepts C, we define the ambiguity vector of v, $\bar{\lambda}_v = (\lambda_{v,1}, \lambda_{v,2}, ...\lambda_{v,n})$, to be the vector summation of its matchings ambiguity vectors. Therefore,

$$\bar{\lambda}_v = \sum_{\sigma \in v} \bar{\lambda}_\sigma. \tag{3}$$

Ambiguity was introduced in [4] as a phenomenon where several concepts may give a different semantic interpretation to an attribute in a schema. $\lambda_{v,i} = 0$ means that attribute a_i is not matched by any of the concepts that participate in v. $\lambda_{v,i} = 1$ means that attribute a_i is matched by exactly one pair and $\lambda_{v,i} > 1$ means that attribute a_i is covered by more than one pair in the cover. For example, the cover in Table II is represented by an ambiguity vector where the relevant entry of the attribute carType is assigned the value of 1 and the relevant entry of the attribute location is assigned with 2, reflecting its presence in two out of the four pairs in the cover.

We generalize the notion of schema matching similarity to a cover. We first note that Eq. 1 can be adopted to reflect the similarity of a subschema-concept pair. We denote by $M_\sigma(t, c)$ the similarity measure of a matching σ between a subschema t and a concept c. Therefore,

$$M_v(s, C) = f(\{M_\sigma(t, c) \mid \sigma \in v\}). \tag{4}$$

For example, assume that $f = sum$, then the schema matching similarity of a cover v is

$$M_v(s, C) = \sum_{\sigma \in v} M_\sigma(t, c). \tag{5}$$

In what follows, we shall also use a measure of dissimilarity. Therefore, given a cover v, $dM_v(s, C)$ is defined similarly

$$dM_v(s, C) = f(\{dM_\sigma(t, c) \mid \sigma \in v\}) \tag{6}$$

where $dM_\sigma(t, c)$ is defined to be $1 - M_\sigma(t, c)$.

IV. Problem Definitions

Equipped with the formal definition of a cover, we can devise an array of optimization problems, all aiming at optimizing some quality aspect of a cover. To illustrate the various optimization problems, we now provide a simplified example.

Example 1 *Let* $s = \{a_1, a_2, a_3\}$ *be a schema and* $T_s = \{\{a_1\}, \{a_1, a_2\}, \{a_2, a_3\}\}$ *be a set of subschemas of* s*. Also, let* $C = \{c_1, c_2, c_3\}$ *be a set of concepts so that* $\mathcal{E}(T_s, C) = \{(\{a_1\}, c_1), (\{a_1, a_2\}, c_2), (\{a_2, a_3\}, c_3)\}$ *is the set of valid matchings between* T_s *and* C*. The attribute correspondences are illustrated in Figure 3. The similarity values of the elements of* $\mathcal{E}$ *are given as follows:*

σ	$M_\sigma(t, c)$
$(\{a_1\}, c_1)$	0.45
$(\{a_1, a_2\}, c_2)$	0.95
$(\{a_2, a_3\}, c_3)$	0.45

□

The following problem was specified in [4] as the schema covering problem.

Problem 1 (Singly-Bounded Maximization Cover) *Given a set of valid matchings* $\mathcal{E}(T_s, C)$*, the* singly-bounded maximization cover *problem (SBMC) is defined to be:*

$$\max_{v \subseteq \mathcal{E}(T_s, C)} M_v \ s.t. \ \bar{\lambda}_v \leq \bar{H},$$

where $\bar{H}$ *is a vector of integers.*

For example, let $\bar{H} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$. Two possible covers that satisfy the ambiguity constraint are $\{(\{a_1\}, c_1), (\{a_2, a_3\}, c_3)\}$ and $\{(\{a_1, a_2\}, c_2)\}$. Using *sum* for M_v, as in [4], we get $M_v = 0.9$ for the first cover and $M_v = 0.95$ for the second cover. Therefore, the cover $\{(\{a_1, a_2\}, c_2)\}$ is the solution to Problem 1.

Problem 1 carries two characteristics we would like to tune. First, we note that a solution to this optimization problem may not necessarily yield a cover in the intuitive sense. That is, there may be attributes in the schema that are not covered by any concept. This is evident in the example above, where covering a_1 and a_2 is preferred over covering all attributes. Our second observation is that Problem 1 is "greedy" in the sense that concepts may be added although they offer no true contribution to the matching, simply because the ambiguity "budget" was not fully spent yet. To demonstrate this phenomenon, we change $\bar{H}$ to be $\bar{H} = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$. The cover $\{(\{a_1\}, c_1), (\{a_1, a_2\}, c_2), (\{a_2, a_3\}, c_3)\}$ becomes the solution to Problem 1 with $M_v = 1.85$, although both $\{(\{a_1\}, c_1), (\{a_2, a_3\}, c_3)\}$ (with $M_v = 0.9$) and $\{(\{a_1, a_2\}, c_2), (\{a_2, a_3\}, c_3)\}$ (with $M_v = 1.4$) suffice to "cover" (in the intuitive sense) s.

We are now ready to introduce the doubly-bounded minimization cover problem, seeking covers with different properties than Problem 1.

Problem 2 (Doubly-Bounded Minimization Cover) *Given a set of valid matchings* $\mathcal{E}(T_s, C)$*, the doubly-bounded minimization cover problem (DBMC) is defined to be:*

$$\min_{v \subseteq \mathcal{E}(T_s, C)} dM_v \ s.t. \ \bar{H}_l \leq \bar{\lambda}_v \leq \bar{H}_u,$$

where $\bar{H}_l$ *and* $\bar{H}_u$ *are vectors of integers.*

In Problem 2 we set a minimal bound on ambiguity. If $\bar{H}_l = \bar{0}$ then we are back to the ambiguity constraint that was set in Problem 1. Setting $\bar{H}_l = \bar{1}$ ensures that an intuitive notion of a schema cover is achieved and each attribute is "covered" by at least one concept. Higher values of elements of $\bar{H}_l$ generalize the problem. We also note that Problem 2 aims at minimizing dissimilarity. Therefore, any additional concept added to the cover may either leave its dissimilarity measure unchanged or increase it. This feature changes the "greediness" feature of Problem 1. Concepts are added to maintain the lower bound of ambiguity and the overall ambiguity is also bounded from above as before.

dM_v is a representative of a quality measure assigned with a specific cover and $\bar{H}_l$ and $\bar{H}_u$ are constraints associated with each attribute in the original schema. As such, Problem 2 (as well as Problem 1) are instantiations of a general formulation for a cover problem in which a quality measure of a cover is optimized subject to a set of constraints on individual attributes.

It is worth noting that problems 1 and 2 both treat $\bar{\lambda}$ as a hard constraint while optimizing a measure of similarity (or dissimilarity). Optimizing two quality measures of a cover, ambiguity and similarity, may be alternatively viewed as a bi-objective optimization problem. Therefore, one can also envision an ambiguity minimization problem, where the role of ambiguity and similarity is exchanged. We refrain from providing a formal presentation of such a problem due to space consideration and defer it to an extended version of this work.

The need for introducing the generalized cover problem and several of its instantiations goes beyond intellectual interest. If there are various ways to use concepts to cover schemas, can we evaluate whether one way is better than the other? One way of doing so can evaluate which of the instantiations yields a more effective matching, in terms of precision and recall. Indeed, in Section VI we answer this question empirically, showing that solutions to Problem 2 outperform solutions to Problem 1 in terms of both precision and recall.

A. *Complexity Analysis*

Having introduced two variations of the generalized cover problem we now turn our attention to analyzing the complexity of the problem. it was shown in [4] that the *Singly-Bounded Maximization Cover* problem is NP-complete. In what follows, we show the same complexity result for the *Doubly-Bounded Minimization Cover* problem.

Theorem 1 *The decision of the Doubly-Bounded Minimization Cover problem is NP-complete.*

Proof: The decision of the Doubly-Bounded Minimization Cover problem is as follows. Given a set of valid matchings $\mathcal{E}(T_s, C)$, two vectors of integers $\bar{H}_l$ and $\bar{H}_u$, and a constant B, is there a cover $v \subseteq \mathcal{E}(T_s, C)$ such that $dM_v \leq B$ and $\bar{H}_l \leq \bar{\lambda}_v \leq \bar{H}_u$? Naturally, given $v \subseteq \mathcal{E}(T_s, C)$, one can validate in polynomial time whether $dM_v \leq B$ and $\bar{H}_l \leq \bar{\lambda}_v \leq \bar{H}_u$. Thus, the problem is in NP. To show NP-completeness, we present a polynomial reduction from Exact Set Cover as follows:
Given a set of elements $\mathcal{U}$, a set of subsets $U_1 \dots U_m$ of $\mathcal{U}$, and positive integer b, let $\sigma_i = \{(a, a^c) \mid a \in U_i\}$ be a schema matching for $1 \leq i \leq m$, and $\mathcal{E}(T_s, C) = \{\sigma_1, \dots, \sigma_m\}$. a^c represents some attribute in a concep that matches a. Let $\bar{H}_l = \bar{1}$ and $\bar{H}_u = \bar{b}$.

We need to show now that (A) there exists $v \subseteq \mathcal{E}(T_s, C)$ such that $\bar{1} = \bar{\lambda}_v$ iff (B) there exists an exact set cover of $\mathcal{U}$ of the same size. First we show that if (A) then (B). Let $v = \{\sigma_i \mid i \in I \subseteq [1, m]\} \subseteq \mathcal{E}(T_s, C)$ be such that $\bar{1} = \bar{\lambda}_v$. We need to show that

(i) for each $u \in \mathcal{U}$, there exists $i \in I$ such that $u \in U_i$, and
(ii) for each $i, j \in I$, we have $U_i \cap U_j = \emptyset$.

First, for each $u \in \mathcal{U}$, $\lambda_{v,u} = 1$, and thus there exists $i \in I$ such that $(u, u^c) \in \sigma_i$ and therefore $u \in U_i$. Now, if $u \in U_i \cap U_j$, then $\lambda_{v,u} > 1$.

Now we show that if (B) then (A). If there exists an exact set cover $\{U_i \mid i \in I \subseteq [1,m]\}$ of $\mathcal{U}$, let $v \subseteq \mathcal{E}(T_s, C)$ be $v = \{\sigma_i \mid i \in I \subseteq [1,m]\}$. We show that $\bar{1} = \bar{\lambda}_v$. For each element $u \in \mathcal{U}$, there exists $i \in I$ such that $u \in U_i$ and $\forall j \neq i, u \notin U_j$. Thus $(u, u^c) \in \sigma_i$ and $\forall j \neq i, u \notin \sigma_j$, and therefore $\lambda_{v,u} = 1$. Thus we have (A) iff (B), and the decision of the DBMC problem is NP-complete. ■

V. Algorithms

In this section we present three algorithms for solving the Doubly-Bounded Minimization Cover Problem. We start with an Integer Linear Programming formulation of the problem (Section V-A) followed by two heuristics in Section V-B.

A. ILP Formulation

We present an Integer Linear Programming formulation to the DBMC problem. ILP problems are known to be NP-complete [18], and therefore no polynomial time algorithm exists (unless P=NP). However, contemporary efficient solvers solve many instances of ILP within a reasonable time frame. In Section VI we present an extensive empirical evaluation using MOSEK solver [19], showing its ability to solve both the SBMC and the DBMC problems efficiently, even for large concept bases.

We start by noting that in the DBMC problem, the optimization is performed over subsets of the set $\mathcal{E}(T_s, C)$, and thus we associate a binary variable X_σ with each $\sigma \in \mathcal{E}(T_s, C)$. There is a natural $1:1$ correspondence between the assignments to these variables and the subsets of $\mathcal{E}(T_s, C)$. Given that, the linear constraints are defined according to Eq. 3 as follows.

$$\bar{H}_l \leq \sum_{\sigma \in \mathcal{E}(T_s, C)} X_\sigma \cdot \bar{\lambda}_\sigma \leq \bar{H}_u. \tag{7}$$

Using Eq. 6, dM_v is defined as a linear aggregation function over the elements of v, that is

$$dM_v = f(\{dM_\sigma \mid \sigma \in v\}), \tag{8}$$

where each dM_σ depends only on σ. Then, the (linear) objective function of our ILP can be defined as in Eq. 8. For example, if the aggregation function is summation, we get

$$\min \sum_{\sigma \in \mathcal{E}(T_s, C)} dM_\sigma \cdot X_\sigma. \tag{9}$$

B. Heuristic methods

In this section we describe a simple heuristic algorithm (with two instantiations) for finding an optimal cover, according to Problem 2. Our heuristic algorithms follow a simple scheme: we first order the set $\mathcal{E}(T_s, C)$ using a simple scoring function and then we process this ordered list, one by one. In each step, we maintain a set of candidate subschema-concept pairs: if at least one attribute of the actual pair brings us closer to satisfying the lower bound and the pair also satisfies the upper bound, we add this element to the set of candidates. We also maintain temporary ambiguity values for the remaining problem that we update each time we add a new pair to the candidate set. We continue with this process until either we find a valid cover or we processed the entire list. Algorithm 1 provides the pseudocode for our heuristic method.

It is worth noting that any new element of $\mathcal{E}(T_s, C)$ that is added to the cover satisfies two conditions. First, it is useful in the sense that it helps satisfying the lower bound of ambiguity for at least one attribute. Then, it does not violate the upper bound constraint. For the case in which $\bar{H}_l = \bar{1}$, we expect each and every attribute of the schema to be covered.

Algorithm 1: Heuristic algorithm for schema covering

$\bar{\lambda}_v \leftarrow \bar{0}$ /* initialize the ambiguity vector */
$v \leftarrow \emptyset$ /* initialize the cover variable */
$\mathcal{E}(T_s, C)$ ordered according to a scoring function
for all $\sigma \in \mathcal{E}(T_s, C)$ **do**
  if $\bar{\lambda}_v + \bar{\lambda}_\sigma \leq H_u$ and $\exists i$ s.t. $\bar{\lambda}_{v,i} < \bar{H}_{l,i}$ and $\bar{\lambda}_{\sigma,i} > 0$ **then**
    $\bar{\lambda}_v \leftarrow \bar{\lambda}_v + \bar{\lambda}_\sigma$
    $v \leftarrow v \cup \{\sigma\}$
    if $\bar{\lambda}_v \geq \bar{H}_l$ **then**
      return v
    end if
  end if
end for
return Fail

TABLE III: Data and Concept Sets

Dataset	# of schemata	# of domains	schema size (# of attributes)	Relevant concept sets
OntoBuilder	27	2	21-88	WF
Vendor	3	1	304	UBL, NativeVendor
Business Partners	3	1	~100	UBL, NativeBP
IBM	5	1	144-456	IBM
Synthetic			700-1000	Synthetic
Concept set	**# of concepts**	**# of domains**	**concept size (# of attributes)**	**Relevant datasets**
WF	15	2	3-12	OntoBuilder
NativeVendor	10-12	1	10	Vendor, Business Partner
UBL	62	1	500-700	Vendor, Business Partner
IBM	292	4	~15	IBM
Synthetic	350-8000		25-40	Synthetic
NativeBP	10	1	3-10	Vendor, Business Partner

We consider two heuristic scoring functions. The first, dubbed SDF (Smallest Dissimilarity First), ranks the elements of $\mathcal{E}(T_s, C)$ in an increasing dissimilarity value $1-\mu_\sigma$ order. The second, dubbed LSF (Largest Subschema First), ranks elements of $\mathcal{E}(T_s, C)$ according to the size of the subgraphs.

VI. Empirical Evaluation

In this section we describe and discuss our empirical evaluation of the cover problem. We first describe the datasets and the experiment setup (Section VI-A) followed by a description and analysis of experiment results.

A. Datasets and Experiment Setup

1) Datasets: We used two types of datasets, namely real-world and synthetic. Features of both datasets are summarized in Table III.

The OntoBuilder dataset consists of 27 Web forms from two domains, car reservations and aviation. We extracted a schema from each Web form using OntoBuilder.[2] The schemata vary in size, from 21 to 88 attributes. The Vendor

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

and Business Partner datasets consists of 3 schemas each, containing information about the vendor and business partner business objects, derived from 3 different SAP systems. The IBM dataset describes the features of the dataset that was used in [4]. We were unable to experiment with this dataset but we have experimented with datasets (both real-world and synthetic) with similar features.

The Web form (WF) concept set is constructed by human effort for each domain. The NativeVendor concept set is constructed by performing schema decomposition to each of the 3 schemas in the Vendor dataset. The UBL concept set was formed from schemas of the Universal Business Language (UBL) version 2.1. UBL is a library of standard electronic XML business documents such as purchase orders and invoices that is developed by OASIS.[3] the IBM concept set represents the features of the concept set that was used in [4].

2) Experimental Setup: We implemented an experimental environment to test the different solutions. We now explain our methodology, summarized in Table IV. In each experiment a single schema (from the pool of real-world or synthetic dataset) is introduced to a concept repository. We vary the size and type of concepts in the concept repository. The decomposition phase is performed using concepts from a relevant domain and the number of subschemas also vary among experiments. Jointly, the number of concepts and the number of subschemas determine a range for the number of pairs in each experiment.

We have varied the low and high ambiguity constraints. For the high ambiguity constraint we use a k-reduction value that ranges from 0 to 4. A k-reduction value of 0 sets the high ambiguity constraint for each attribute to be the number of concepts that are matched with this attribute. Higher k-reduction values put an increasing constraint on the allowed ambiguity. Table IV provides the six control parameters, for each stating the range of values we use in our experiments and a baseline value (whenever there is one), kept fixed unless otherwise described.

We report on the following metrics:

- Effectiveness: For a cover v we measure M_v and dM_v.
- Runtime: here we report on the actual execution time of each algorithm.
- Correctness: This metric aims at justifying the use of a cover from a typical, schema matching, point of view. The evaluation of schema matchings is performed with respect to an *exact matching*, based on expert opinions. *Precision* and *recall* are used for the empirical evaluation of performance. Assume that out of the $n \times n'$ attribute matchings, there are $c \leq n \times n'$ correct attribute matchings, with respect to the exact matching. Also, let $t \leq c$ be the number of matchings, out of the correct matchings, that were chosen by the matching algorithm**,** and let $f \leq n \times n' - c$ be the number of incorrect attribute matchings. Then, precision is computed to be

$$P = \frac{t}{t+f}$$

and recall is computed as

$$R = \frac{t}{c}$$

Clearly, higher values of both precision and recall are desired.

A common derivative of precision and recall is *F-Measure*, the harmonic mean of precision and recall:

$$FM = 2 \cdot \frac{PR}{P+R}$$

We extend the definition of precision and recall to the evaluation of a cover result in three ways. In the first version ($P1$ and $R1$) we evaluate the precision and recall of each subschema-concept pair and then average the performance over all pairs in the cover. In the second version ($P2$ and $R2$) we assemble all the concepts

[3]http://www.oasis-open.org/committees/ubl/

TABLE IV: Controlled Parameters

Parameter	Parameter Name	Range	Baseline
$\|C\|$	# of concepts	4-3000	
$\|T\|$	# of subschemas	2-500	
$\|a\|$	# of attributes in a subschema	2-100	
$\|a^c\|$	# of attributes in a concept	2-100	
H_l	low ambiguity constraint	0-2	1
k	reduction of high ambiguity constraint	0-4	0

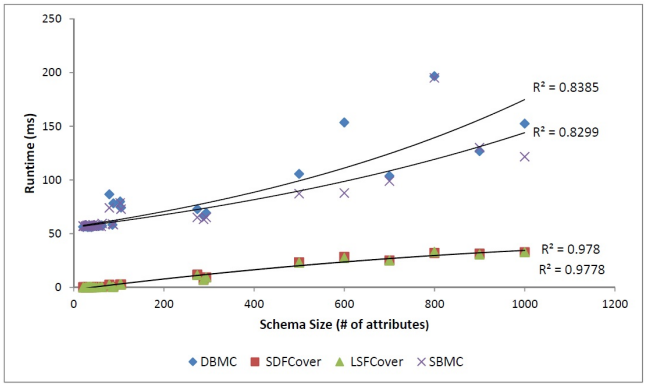

Fig. 4: Runtime as a function of schema size

that participate in a cover and compute precision and recall over the matching of the schema to the union of the concepts. In the third and final version ($P3$ and $R3$), we take ALL concepts in the concept repository, turn them into one big schema and then evaluate the result of matching the schema to the new concept-based schema. We note that the first two versions are different methods for computing the quality of a cover algorithm while the third version performs a different schema matching task altogether. When comparing $P1/R1$ with $P3/R3$ one gets a sense of the quality of the chosen concepts. When comparing $P2/R2$ with $P3/R3$ we can assess the quality of the concept selection process.

The experiments were conducted on a Intel(R) Core(TM)2 Quad CPU Q8200 @ 2.33GHz. The algorithms were implemented in Java, using JDK version 1.6.0. The JVM was initiated with a heap-memory size of $8.00GB$.

We use Auto-Mapping Core (AMC) to perform decomposition. AMC is a tool developed by SAP Research that provides an infrastructure and a set of algorithms to establish correspondences between two business schemas. The algorithms are designed to explore various features and dependencies that exist in business schemas. Based on the extracted features the algorithm may suggest correspondences between nodes in two different schemas. Different algorithms may suggest different correspondences and the overall result is integrated based on quality measures as reported by the various algorithms.

B. Runtime

Figure 4 demonstrates the increase of runtime with schema size, using both real-world and synthetic datasets. The runtime of the ILP-based solutions is higher than the two heuristics. While the ILP-based solutions demonstrate an exponential runtime trend (top two trendlines) the heuristics show a close to quadratic runtime trend (two bottom trendlines).

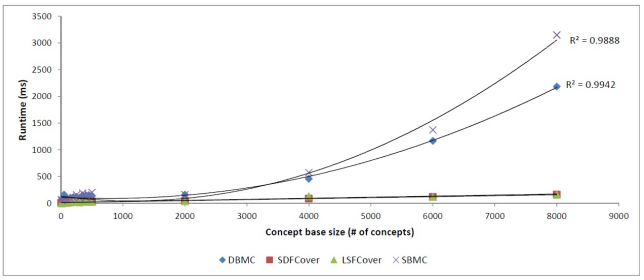

Fig. 5: Runtime as a function of concept base size

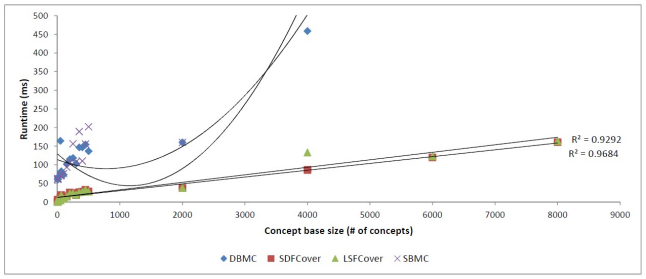

Fig. 6: Runtime as a function of concept base size, magnified

Figures 5 and 6 illustrate the impact of concept base size on runtime. Runtime increases with concept base size, albeit with a weaker trend. The ILP-based solutions show a quadratic runtime trend (Figure 5), and the heuristics show a linear trend in the concept base size (Figure 6).

Figure 7 illustrates the impact of the number of concepts in a cover on the execution time. The correlations is inconclusive, with low R^2 values, for all but DBMC that exhibits an exponential correlation. It is worth noting that SBMC finds covers with more concepts than DBMC due to the way it is defined. However, it maintains a lower execution time.

C. Heuristic Performance

In the following set of experiments we have compared the performance of the optimal DBMC with the performance of the two heuristics, in terms of achieved dissimilarity (which should be minimized) and ambiguity (which we also aim at keeping low). Figure 8 shows the results of 18 experiments with real-world datasets, for each

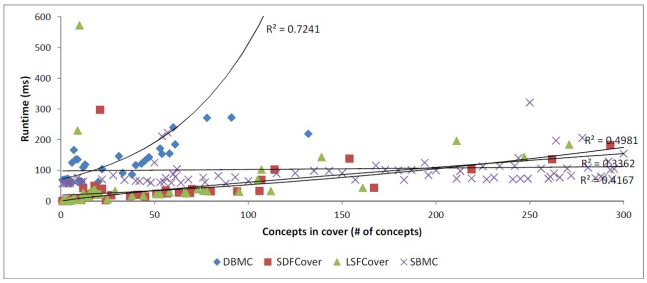

Fig. 7: Runtime as a function of number of concepts in a cover

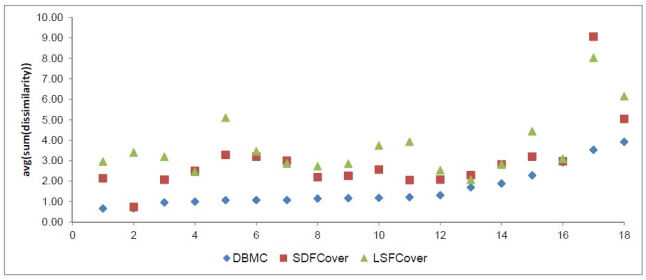

Fig. 8: Dissimilarity comparison of DBMC vs. heuristics

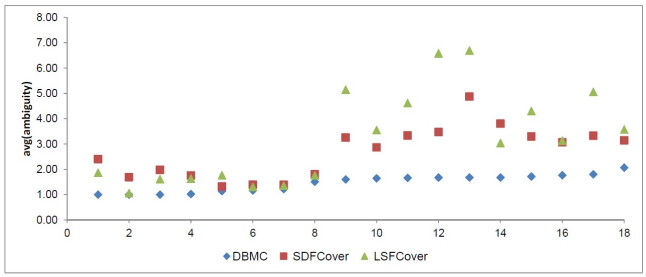

Fig. 9: Ambiguity comparison of DBMC vs. heuristics

recording the average of the sum of dissimilarity over all participating concepts. DBMC is optimal and therefore the heuristics cannot achieve lower dissimilarity. SDFCover performs better than LSFCover. SDFCover achieves a dissimilarity between 1.6% and 221% higher than DBMC with an average of about 100% higher dissimilarity. LSFCover achieves a dissimilarity between 6% and 409% higher than DBMC with an average of about 170% higher dissimilarity. Such a difference in performance is expected, given that SDFCover chooses first concepts that match with lower dissimilarity while LSDCover aims at concepts that cover more attributes.

Average ambiguity is compared and presented in Figure 9. Although DBMC does not aim at minimizing ambiguity, the heuristics posses a greedy behavior and hence they create higher ambiguity. SDFCover achieves an ambiguity between 14% and 190% higher than DBMC with an average of about 80% higher ambiguity. LSFCover achieves an ambiguity between 6% and 199% higher than DBMC with an average of about 110% higher dissimilarity. The difference in performance can be justified by a higher chance of overlap between concepts that cover many attributes, which is the selection criteria for LSFCover.

D. Impact of Ambiguity Constraint

Table V illustrates the impact of the ambiguity constraint on SBMC and DBMC. For schemas of different sizes (ranging here from 30 to 40 attributes), we have applied a k reduction of ambiguity, starting with allowing maximum ambiguity ($k = 0$). We report for each schema size and for each k, the number of covered attributes when applying SBMC and DBMC.

We have set DBMC with a minimum ambiguity of 1. Therefore, whenever the upper limit of ambiguity does not allow certain attributes to be mapped, DBMC returns no solution, marked with a blank cell in the table.

Whenever a full-cover solution is possible for DBMC it covers all of the attributes in the schema while SBMC does not necessarily do so. For example, for a schema of size 40, number of covered attributes reduces with an increased k in solutions for SBMC, while up to $k = 3$, a solution that cover all attributes can still be found.

TABLE V: Impact of Ambiguity Constraint

k-reduce → ↓ Schema size	0		1		2		3		4	
	SBMC	DBMC	SBMC	DBMC	SBMC	DBMC	SBMC	DBMC	SBMC	DBMC
30	19	19	18		17		17		17	
31	18	18	15		15		15		15	
32	23	23	23	23	22		22		20	
33	20	20	18		17		17		17	
34	19	19	19	19	19	19	19	19	19	19
35	15	15	15	15	15	15	15	15	15	15
36	13	13	9	13	10	13	10	13	10	13
37	28	28	25		18		18		17	
38	20	20	14		13		12		11	
39	20	20	16		16		16		14	
40	28	28	27	28	23	28	23	28	21	

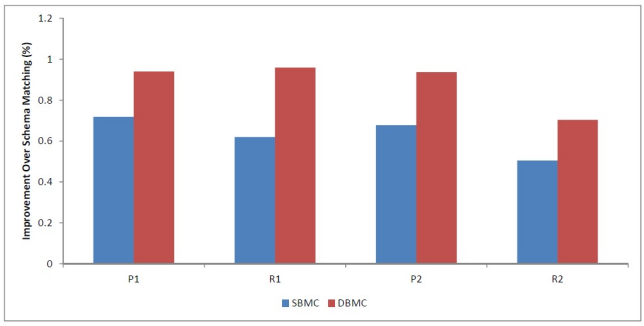

Fig. 10: Improvement of Performance over Schema Matching

An interesting anomaly can be seen for a schema of size 36. Here, when k increases from 1 to 2, meaning that a tighter ambiguity constraint is applied, the number of covered attributes for SBMC actually increases (from 9 to 10).

E. Correctness

We now turn to compare the performance of the cover algorithms with respect to a regular schema matching. Recall that in Section VI-A we defined three variations of precision and recall. We first demonstrate it using the cover task of a vendor schema (called vendor2) to a concept base that was constructed by another vendor schema (called vendor1). Performing a schema matching of vendor2 with vendor1 yields $P3 = 0.69$ and $R3 = 0.9$. Finding a cover using DBMC yields $P1 = 0.7$, $R1 = 0.94$, $P2 = 0.66$, and $R2 = 0.9$. A cover using SBMC yields $P1 = R1 = R2 = 0.41$ and $P2 = 0.31$. In this example, when comparing $P1/R1$ with $P3/R3$, we see that DBMC manages to choose concepts well and even improves slightly over the basic schema matching task (both in terms of precision and in terms of recall). Comparing $P2/R2$ to $P3/R3$ suggests that the concept base is of reasonable quality yet sets limits to the ability of this concept set to improve over the basic schema matching problem. Using SBMC in this case harms the ability to match schemas well.

In half of the experiments, the solution to DBMC satisfies $P1 > P3$. As for recall, in 61% of the experiments, DBMC satisfies $R1 > R3$. This means that the use of concepts improves the correctness of the outcome. Figure 10 compares the average performance of the two problems with respect to the schema matching with no concepts, performed over all real-world data sets. All in all, the performance of solutions to DBMC are comparable with that

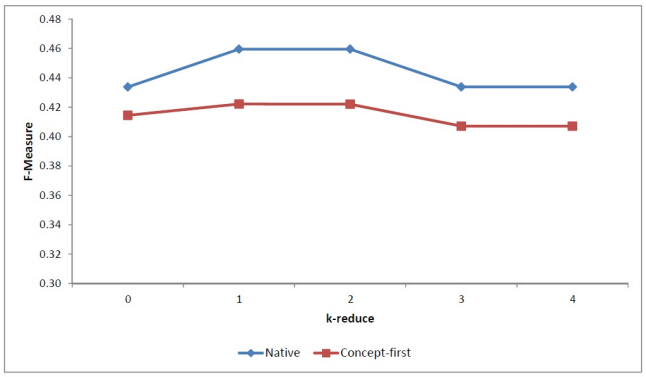

Fig. 11: F-Measure as a function of k-reduce for native and concept-first decomposition

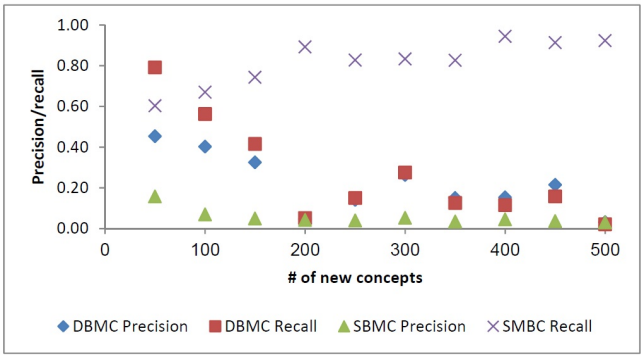

Fig. 12: Precision and recall as a function of concept base size

of regular schema matching, showing worse performance when computing $P2$ and $R2$. The added value of using concepts is the interpretation of the schema in terms of concepts. It is also clear that cover solutions using SBMC performs worse.

F. Impact of Decomposition Type

We next compare the effectiveness of the two decomposition types, introduced in Section II, *native* and by *concept-first*.

Figure 11 compares the effectiveness of the two decomposition types, in terms of F-measure, for different ambiguity upper-bound constraints. The experiment was performed using the DBMC ILP cover algorithm. For both decomposition methods, F-Measure first improves (for $k = 1, 2$) and then drops. This can be explained by the fact that tighter ambiguity bounds force the algorithm to focus on concepts that provide overall better performance. However, at some point the ambiguity constraints become an obstacle by themselves to achieving good performance.

Native decomposition outperforms concept-first decomposition for all k-reduced values, which is somewhat surprising. A native decomposition may restrict concepts in their ability to choose better attributes to be matched. However, a native decomposition has similar effect to concepts, in that it requires locality in the matching (see Figure 1 for a similar reasoning in favor of using concepts in the first place).

G. Robustness

To check the impact of the quality of concepts on the cover process we performed the following experiment. We start by comparing the schema Vendor1 with the concept set that was created from it. Such a matching yields a 100% precision and recall. Then, we start mutating the concept set as follows. We randomly choose an attribute from Vendor1 and a concept in the concept set. We then create a new concept by adding the chosen attribute to the concept. By doing so, we contaminate the concept by adding an unrelated attribute. Finally, we run the cover algorithms with k-reduced of 0 and an ambiguity lower bound of $\bar{1}$ for DBMC. Then, we measure the precision ($P2$) and recall ($R2$) of the outcome.

Figure 12 presents the results. The x-axis counts the number of mutated concepts and the y-axis shows the resulting precision and recall. Precision suffers largely from reducing the quality of concepts results. With 50 new concepts, precision drops below 0.5 for DBMC and to about 15% for SBMC. Recall increases for SBMC, as expected, since we set no constraint on ambiguity. The recall of DBMC is reduced, just like the precision, albeit in a slower rate up to 150 new concepts. We conclude that the success of cover depends, to a large degree, on the quality of the concept base. We also observe that DBMC is more resilient, in terms of precision, to such loss of quality.

H. Discussion

Our empirical analysis covers run-time, correctness, and quality of cover solutions. When it comes to run-time analysis we show that while the general cover problem and its two instantiations are NP-Complete, encoding cover problems using ILP generates an efficient solution that can handle large schemas and big concept bases. Regarding correctness, we have compared the performance of cover algorithms with that of traditional schema matching solutions. We show that different cover solutions provide different levels of correctness. DBMC shows better performance for the datasets tested than SBMC. Ambiguity constraints, when given in moderation, help in identifying better interpretation to a schema. At the extremes (no constraint or harsh constraints), ambiguity harms the ability to generate useful matchings. Finally, native concept decomposition shows better results that by concept-first.

VII. Related Work

Our work on schema covering borrows heavily from the area of schema matching. We provide a brief overview of major achievements in schema matching modeling. The body of research on the topic of schema matching is vast and we do not attempt to cover all of it here. Schema matching research has been going on for more than 25 years now (see surveys [1], [20], [21], [22] and online lists, *e.g.*, OntologyMatching[4] and Ziegler[5]) first as part of schema integration and then as a standalone research. Due to its cognitive complexity, schema matching has been traditionally considered to be AI-complete, performed by human experts [23], [24]. Semi-automatic schema matching has been justified in the literature using arguments of scalability (especially for matching between large schemata [25]) and by the need to speed-up the matching process. Fully-automatic (that is, unsupervised) schema matching was suggested in settings where a human expert is absent from the decision process, *e.g.*, machine-understandable Web resources [26].

Over the years, a significant body of work was devoted to the identification of *schema matchers*, heuristics for schema matching. Examples include COMA [7], Cupid [11], OntoBuilder [12], Autoplex [13], Similarity Flooding [27], Clio [28], Glue [29], to name just a few. The main objective of schema matchers is to provide schema matchings

[4]http://www.ontologymatching.org/

[5]http://www.ifi.unizh.ch/~pziegler/IntegrationProjects.html

that will be effective from the user point of view, yet computationally efficient (or at least not disastrously expensive). Such research has evolved in different research communities, including databases, information retrieval, information sciences, data semantics and the semantic Web, and others.

The closest work to our own is that of [4], where the problem of schema covering was first introduced. In this work we extend the cover definition of [4] to a general linear constraint optimization, and offer a new algorithm and two heuristics to solve cover problems. We show that the ILP formulation of the problem is efficient even for large schemas and large concept repositories. We also show that the proposed cover algorithm performs better, in terms of precision and recall.

VIII. Conclusions

We have presented a general framework for schema cover, where micro-mappings are used to cover a newly arived schema. In this formulation, a cover problem aims at optimizing a quality measure subject to a set of constraints on individual attributes. We focus on minimizing a dissimilarity measure subject to bounds on ambiguity and also show that this general framework extends previous works in which similarity was maximized subject to an upper bound on ambiguity.

The empirical analysis shows the effectiveness of the proposed cover formulation (DBMC) over previous proposal (SBMC) in terms of precision and recall. It also shows the efficiency of ILP encoding of cover problems, by running experiments on schemas with up to 1,000 attributes and a concept repository with up to 8,000 concepts. Due to the NP-Complete nature of the cover problems, we also provide two simple heuristics and compare their performance with that of DBMC. We conclude that a heuristic that prefers low dissimilarity is in general preferred, both in terms of the optimization value and in terms of the achieved ambiguity, over a heuristic that aims at achieving ambiguity lower bound faster.

We believe that cover formulation is an essential component in the NisB toolkit of data integration, serving as a discovery as well as a matching tool. In terms of future work, we intend to investigate formulations of cover problems that aim at minimizing ambiguity subject to individual constrains on dissimilarity. Improving the quality of concept bases is also part of the future work agenda, possibly through techniques of clustering and natural evolution.

Acknowledgement

We thank the NisB team for useful observations and comments.

REFERENCES

[1] C. Batini, M. Lenzerini, and S. Navathe, "A comparative analysis of methodologies for database schema integration," *ACM Computing Surveys*, vol. 18, no. 4, pp. 323–364, Dec. 1986.

[2] M. Lenzerini, "Data integration: A theoretical perspective." in *Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, 2002, pp. 233–246.

[3] P. Bernstein and S. Melnik, "Meta data management," in *Proc. 20th Int. Conf. on Data Engineering*, 2004, tutorial Presentation.

[4] B. Saha, I. Stanoi, and K. Clarkson, "Schema covering: a step towards enabling reuse in information integration," in *Proc. 26th Int. Conf. on Data Engineering*, 2010, pp. 285–296.

[5] S. Melnik, *Generic Model Management: Concepts and Algorithms*. Springer-Verlag, 2004.

[6] Y. An, A. Borgida, R. Miller, and J. Mylopoulos, "A semantic approach to discovering schema mapping expressions," in *Proceedings of the IEEE CS International Conference on Data Engineering*, 2007, pp. 206–215.

[7] H. Do and E. Rahm, "COMA - a system for flexible combination of schema matching approaches," in *Proc. 28th Int. Conf. on Very Large Data Bases*, 2002, pp. 610–621.

[8] A. Gal, *Uncertain Schema Matching*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[9] B. He and K. C.-C. Chang, "Statistical schema matching across Web query interfaces," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2003, pp. 217–228.

[10] W. Su, J. Wang, and F. Lochovsky, "A holistic schema matching for Web query interfaces," in *Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology*, 2006, pp. 77–94.

[11] J. Madhavan, P. Bernstein, and E. Rahm, "Generic schema matching with Cupid," in *Proc. 27th Int. Conf. on Very Large Data Bases*, 2001, pp. 49–58.

[12] A. Gal, G. Modica, H. Jamil, and A. Eyal, "Automatic ontology matching using application semantics," *AI Magazine*, vol. 26, no. 1, pp. 21–32, 2005.

[13] J. Berlin and A. Motro, "Autoplex: Automated discovery of content for virtual databases," in *Proc. Int. Conf. on Cooperative Information Systems*. Springer, 2001, pp. 108–122.

[14] A. Doan, P. Domingos, and A. Halevy, "Reconciling schemas of disparate data sources: A machine-learning approach," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2001, pp. 509–520.

[15] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy, "Corpus-based schema matching," in *Proc. 21st Int. Conf. on Data Engineering*, 2005, pp. 57–68.

[16] G. Klir and B. Yuan, Eds., *Fuzzy Sets and Fuzzy Logic*. Prentice Hall, 1995.

[17] Y. Lee, M. Sayyadian, A. Doan, and A. Rosenthal, "eTuner: tuning schema matching software using synthetic scenarios." *VLDB J.*, vol. 16, no. 1, pp. 97–122, 2007.

[18] R. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. Miller and J. Thatcher, Eds. Plenum Press, 1972, pp. 85–103.

[19] MOSEK, "The MOSEK Optimization Tools Version 6.0 (revision 61)," [Online], 2009, `http://www.mosek.com`.

[20] A. Sheth and J. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Comput. Surv.*, vol. 22, no. 3, pp. 183–236, 1990.

[21] E. Rahm and P. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001.

[22] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," *Journal of Data Semantics*, vol. 4, pp. 146 – 171, Dec. 2005.

[23] B. Convent, "Unsolvable problems related to the view integration approach," in *Proceedings of the International Conference on Database Theory (ICDT)*, Rome, Italy, Sept. 1986, in *Computer Science*, Vol. 243, G. Goos and J. Hartmanis, Eds. Springer-Verlag, New York, pp. 141-156.

[24] R. Hull, "Managing semantic heterogeneity in databases: A theoretical perspective," in *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*. ACM Press, 1997, pp. 51–61.

[25] B. He and K.-C. Chang, "Making holistic schema matching robust: an ensemble approach." in *Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2005, pp. 429–438.

[26] B. Srivastava and J. Koehler, "Web service composition - Current solutions and open problems," in *Workshop on Planning for Web Services (ICAPS-03)*, Trento, Italy, 2003.

[27] S. Melnik, E. Rahm, and P. Bernstein, "Rondo: A programming platform for generic model management," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2003, pp. 193–204.

[28] R. Miller, M. Hernàndez, L. Haas, L.-L. Yan, C. Ho, R. Fagin, and L. Popa, "The Clio project: Managing heterogeneity," *SIGMOD Record*, vol. 30, no. 1, pp. 78–83, 2001.

[29] A. Doan, J. Madhavan, P. Domingos, and A. Halevy, "Learning to map between ontologies on the semantic web," in *Proc. 11th Int. World Wide Web Conf.*, 2002, pp. 662–673.