Completeness and Ambiguity of Schema Cover

Avigdor Gal[1], Michael Katz[1], Tomer Sagi[1], Matthias Weidlich[1], Karl Aberer[2], Hung Quoc Viet Nguyen[2], Zoltán Miklós[3], Eliezer Levy[4], and Victor Shafran[4]

[1] Technion – Israel Institute of Technology, Technion City, Haifa, 32000 Israel
[2] École Polytechnique Fédérale de Lausanne EPFL, CH-1015 Lausanne, Switzerland
[3] University of Rennes 1, France
[4] SAP Research Israel, Ra'anana, 43665, Israel

Abstract. Given a schema and a set of concepts, representative of entities in the domain of discourse, schema cover defines correspondences between concepts and parts of the schema. Schema cover aims at *interpreting* the schema in terms of concepts and thus, vastly simplifying the task of schema integration. In this work we investigate two properties of schema cover, namely *completeness* and *ambiguity*. The former measures the part of a schema that can be covered by a set of concepts and the latter examines the amount of overlap between concepts in a cover. To study the tradeoffs between completeness and ambiguity we define a cover model to which previous frameworks are special cases. We analyze the theoretical complexity of variations of the cover problem, some aim at maximizing completeness while others aim at minimizing ambiguity. We show that variants of the schema cover problem are hard problems in general and formulate an exhaustive search solution using integer linear programming. We then provide a thorough empirical analysis, using both real-world and simulated data sets, showing empirically that the integer linear programming solution scales well for large schemata. We also show that some instantiations of the general schema cover problem are more effective than others.

1 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 becomes feasible. Such reuse can be based on a repository of information building blocks, referred to as *concepts*, [4] representative of entities in the domain of discourse (*e.g.*, a vendor concept in an eCommerce domain). Concepts establish a form of a conceptual middleware, providing a shared set of abstractions that facilitates interoperability.

Documents, modeled as schemata are mapped against a set of concepts in a process termed *schema cover*. The idea is to "cover" a schema 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

R. Meersman et al. (Eds.): OTM 2013, LNCS 8185, pp. 241–258, 2013.

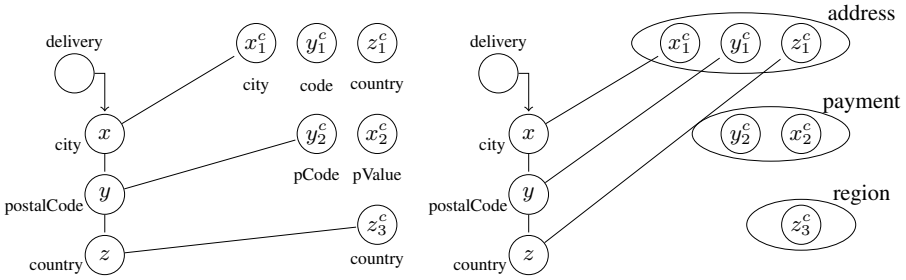


Fig. 1. Role of concepts illustrated

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 real-world entities, by using concepts.

Schema cover matches parts of schemata (called subschemata) with concepts, using schema matching techniques [5] and then adds cover-level constraints. One such constraint is called *ambiguity* [4], intuitively representing the number of times an attribute is matched to attributes in several concepts. Another constraint, introduced in this work, is *completeness*, representing the part of a schema that is "covered" by any concept. Ambiguity and completeness constraints, to be formally defined in this work along with subschemata and concepts, are embedded as (either hard or soft) constraints in an optimization cover problem.

To illustrate the role of concepts in schema cover consider Figure 1. On the left there is a schema with three attributes. On the right, we introduce six attributes that are available for matching in some repository (not necessarily from the same schema). In the absence of concepts (Figure 1(left)), the matching task can select attributes without any restriction. However, in the presence of concepts (Figure 1(right), marked with ovals), we are guided to choose closely related attributes. Our hypothesis is that well-designed concepts improve the quality of the integration for exactly this argument.

Schema cover was first introduced by Saha et. al. [4] as a solution for schema mapping. However, in recent years new applications were introduced, to which schema cover can bring benefit [6,7]. We illustrate next two applications that motivate our investigation of the trade-off between ambiguity and completeness in schema cover.

Partner Identification: An enterprise may be interested in investigating a new market, seeking partners with sufficient common interests to its own while providing sufficient added value to its capabilities. The use of a cover here can ensure just the right amount of commonality between prospective business partners. Here, the main interest is in the amount of completeness of a cover, judging what part of one schema can be covered by concepts of the other schema and vice versa. Ambiguity takes a secondary role of ensuring the cover clarity.

Concept Filtering: Effective reuse requires the accumulation of many concepts from which useful schema covers can be created. To ensure efficient cover processing, concept repositories require a filtering process that ensures that only concepts of good

quality that are also good candidates for a cover are retrieved. In this scenario, completeness is set as a constraint by a designer and ambiguity is a measure that needs to be optimized to avoid poor matching quality.

In this paper we investigate the trade-offs between completeness and ambiguity. The paper makes the following contributions:

- We introduce the trade-off of ambiguity and completeness through a general framework for schema cover, offering a spectrum of schema cover problems where the above applications can be supported, among other scenarios. In particular, we show that the framework of [4] is a special case of the generalized problem.
- We propose a new formulation of a cover problem and show it to be NP-Complete.
- We offer integer linear programming (ILP) formulation for optimally solving certain special variations of the cover problem in an exhaustive manner. These formulations are shown empirically to scale well to schemata with 1,000 attributes and cover repository with 8,000 concepts.
- We provide a thorough empirical analysis of the proposed algorithmic solutions to cover problems, using both real-world and simulated data sets. The empirical evaluation shows that the cover variation proposed in this work is more effective (in terms of completeness for similar levels of ambiguity) than the one presented by Saha et al. [4].

The rest of the paper is organized as follows. Preliminaries are given in Section 2 and Section 3 introduces the cover model. A set of cover problems are presented in Section 4, together with a theoretical analysis, followed by an ILP formulation in Section 5. Section 6 provides our empirical analysis. We conclude with a description of related work (Section 7) and directions for further research (Section 8).

2 Background

In this work, we focus on the task of covering a schema with a (possibly large) repository of concepts and analyze the trade-offs of completeness and ambiguity. Generally speaking, a preprocessing of the cover process involves decomposition and coupling. The former decomposes a schema into subschemata. Concepts and subschemata are then coupled using schema matching techniques, followed by a cover selection.

Schema decomposition is a process in which a schema is broken into subschemata. 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 An et al. [8]. Alternatively, decomposition by concept-first is guided by a set of concepts using schema matching techniques. Therefore, a schema matcher such as COMA++ [9] or OntoBuilder's Top-K algorithm [10] 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 first that the use of Integer Linear Programming with state-of-the-art solvers can do without filtering even for very large concept repositories. Secondly, we discuss a variant of the generalized cover description that assist in focusing 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, as a background, a model of schema matching based on the model presented by Gal [10], and put it in the context of the cover problem. 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.

2.1 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 schemata 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 schemata 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 1 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 [11,12]. Other matchers assume similar attributes share similar

Table 1. A Similarity Matrix Example

$\downarrow s'$	$s \longrightarrow$ 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

domains [13,14]. Others yet take instance similarity as an indication of attribute similarity [15,16]. Finally, some researchers use the experience of previous matchings as indicators of attribute similarity [17,12].

Given two schemata, s and s', let the power-set $\Sigma_{s,s'} = 2^{\mathcal{S}}$ be the set of all possible *schema matches* between the schema pair (s, s'), where a schema match $\sigma(s, s') \in \Sigma_{s,s'}$ 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 match simply as σ and to a power-set of matches 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 $a' \in s'$, $(a, a') \notin \sigma$. 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 match.

2.2 Similarity and Constraints

A schema match 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 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 match σ between two schemata s and s', we can define a schema match similarity measure $M_\sigma(s, s')$ to be:

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

For example, consider Table 1 and assume that the schema match σ involves matching

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

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

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 matches 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 match σ can be accepted by a designer. Γ has been modeled in the literature using matchers called *constraint enforcers* [18].

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 match* $\sigma \in \Sigma^\Gamma$.

3 Cover Model

We now introduce the model that serves in defining the different schema cover problems. The model includes subschemata, concepts (Section 3.1), and a cover (Section 3.2).

3.1 Subschemata and Concepts

Let $T_s = \{t_1, t_2, ..., t_m\}$ be a set of *subschemata* 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 schemata, subschemata, and concepts are all defined to consist of a set of attributes, however with different semantics. Concepts are schemata whose meaning is assumed to be known and well-defined 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.

3.2 Cover

We are now ready to introduce the notion of a cover, defined in this paper as any set of concepts that match parts of 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, demonstrate it using our case study example, and explain the roles of completeness and ambiguity as quality measures for a cover.

Given a set of subschemata 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 subschemata 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 presence vector $\bar{\lambda}_\sigma = (\lambda_{\sigma,1}, \lambda_{\sigma,2}, ...\lambda_{\sigma,n})$ as follows:

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

Table 2. 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...	

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 that are **not** in σ).

To illustrate, a partial sample cover is given in Table 2. 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 present twice in the cover, matched 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.

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

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

Using the values in a presence vector, two quality measures can be derived from it. First, *ambiguity* was introduced in [4] as a phenomenon where several concepts may give a different semantic interpretation to an attribute in a schema. We define the ambiguity of a cover to be the sum of duplicate appearances of an attribute in a cover:

$$A_v(s) = \sum_i \max(0, \lambda_{v,i} - 1) \tag{4}$$

As another quality measure we offer *completeness*, checking to what extent schema attributes are present in a cover:

$$C_v(s) = \frac{\sum_i \min(1, \lambda_{v,i})}{|s|} \tag{5}$$

$\lambda_{v,i} = 0$ means that attribute a_i is not matched by any of the concepts that participate in v, hence reducing completeness (C_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 present in more than one pair in the cover, hence increasing ambiguity (A_v). For example, the cover in Table 2 is represented by a presence 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 the attribute ambiguity in the cover.

[1] This has been the case for many years when bidding for car rentals in `priceline.com`

To generalize schema matching similarity to a cover we observe that Eq. 1 can be adopted to reflect the similarity of a subschema-concept pair. Let $M_\sigma(t, c)$ denote the similarity measure of a matching σ between a subschema t and a concept c. Then,

$$M_v(s, C) = f(\{M_\sigma(t, c) \mid \sigma \in v\}). \tag{6}$$

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

$$M_v(s, C) = \sum_{\sigma \in v} M_\sigma(t, c). \tag{7}$$

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

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

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

4 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 subschemata 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 2. The similarity values of the elements of $\mathcal{E}$ are given as follows:

σ	$M_\sigma(t, c)$	σ	$M_\sigma(t, c)$	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

□

4.1 Singly-Bounded Maximization Cover

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.

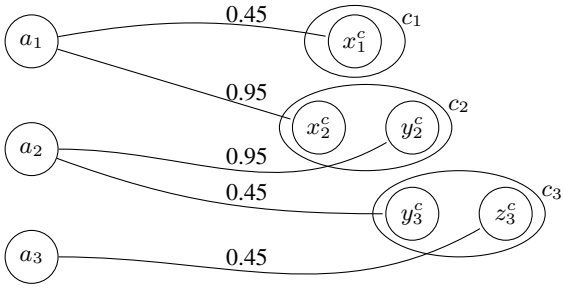


Fig. 2. Illustration of a Cover

For example, let $\bar{H} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$. Such a vector ensures no ambiguity ($A_v(s) = 0$) but does not guarantee completeness ($C_v(s) \leq 1$). Two possible covers that satisfy the presence 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, note that a solution to the optimization problem guarantees ambiguity to be within a certain limit but lacks control over completeness. As a result, 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 cover, simply because the presence "budget" was not fully spent yet. This feature negatively affects ambiguity uneccessarily, without any added value to completeness. 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 achieve maximal completeness.

4.2 Doubly-Bounded Minimization Cover

We are now ready to introduce an alternative cover problem, the *doubly-bounded minimization* cover problem. This problem seeks covers with different properties than Problem 1. As our empirical analysis shows later, the doubly-bounded minimization cover problem offers a more attractive alternative in terms of matching effectiveness.

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 presence. If $\bar{H}_l = \bar{0}$ then we are back to the presence constraint that was set in Problem 1. Setting $\bar{H}_l = \bar{1}$ ensures full completeness. 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 change mitigates the "greediness" of Problem 1. Concepts are added to maintain the lower presence bound and the overall presence 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, presence 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. Intuitively, such a cover problem specification allows the designer to specify the importance she assigns with various attributes, putting more emphasis on the similarity of attributes that are of greater importance. Therefore, such specification becomes handy when filtering the concept base.

The need for introducing the generalized cover problem and several of its instantiations goes beyond intellectual curiosity. If there are various ways to use concepts to cover schemata, 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. Indeed, in Section 6 we answer this question empirically, showing that solutions to the new formulation offered in this work (Problem 2) outperform solutions to Problem 1 (as introduced by Saha et al.), reaching higher completeness for the same ambiguity level.

4.3 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. Theorem 4 (which proof is omitted due to space considerations) asserts the same complexity result for the *Doubly-Bounded Minimization Cover* problem.

Theorem 1. *The decision of the DBMC problem is NP-complete.*

5 ILP Formulation for Cover Problems

We present now an Integer Linear Programming formulation to the DBMC problem. ILP problems are known to be NP-complete [19], 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 6 we present an extensive empirical evaluation using MOSEK solver [20], showing its ability to solve the DBMC problem efficiently, even for large concept bases. We have also implemented the ILP formulation for SBMC, the cover problem presented in [4], showing the efficiency (albeit with lower effectiveness then DBMC) of the solution.

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.

$$\bar{H}_l \leq \sum_{\sigma \in \mathcal{E}(T_s, C)} X_\sigma \cdot \bar{\lambda}_\sigma \leq \bar{H}_u. \tag{9}$$

Using Eq. 8, dM_v is defined as a linear aggregation function over the elements of v, *i.e.*

$$dM_v = f(\{dM_\sigma \mid \sigma \in v\}), \tag{10}$$

where each dM_σ depends only on σ. Then, the (linear) objective function of our ILP can be defined as in Eq. 10. For example, the summation aggregation function yields

$$\min \sum_{\sigma \in \mathcal{E}(T_s, C)} dM_\sigma \cdot X_\sigma. \tag{11}$$

6 Empirical Evaluation

In this section we describe and discuss our empirical evaluation of the cover problem. We first detail the datasets and the experiment setup (Section 6.1) followed by a description and analysis of experiment results.

6.1 Datasets and Experiment Setup

Datasets. We used two types of datasets, namely real-world and synthetic. Features of both datasets are summarized in Table 3.

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 and Business Partner datasets consists of 3 schemata each, containing information about the vendor and business partner business objects, derived from three 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. It is worth noting that we have reimplemented the algorithm of [4] using our general ILP formulation and experimented with it, comparing it to our proposed algorithm.

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

Table 3. 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

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 three schemata in the Vendor dataset. The UBL concept set was formed from schemata 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].

The synthetic dataset is used for run-time evaluation. Therefore, we generate additional schemata and concepts using multiple copies of the real-world schemata with minor variations of schema element names. The number of copies depend on the specific scale that is required per experiment.

Experimental Setup. We implemented an experimental environment to test the different solutions. In each experiment a single schema (from the pool of real-world or synthetic datasets) 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 subschemata also vary among experiments. Jointly, the number of concepts and the number of subschemata determine a range for the number of pairs in each experiment.

We have varied the low and high presence constraints. For the high presence constraint we use a k-reduction value that ranges from 0 to 4. A k-reduction value of 0 sets the high presence 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 presence. Table 4 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:

– Ambiguity: For a cover v we measure $A_v(s)$ normalized by schema size, for better comparison among schemata.

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

Table 4. Controlled Parameters

Parameter	Parameter Name	Range	Baseline
$\|C\|$	# of concepts	4-3000	
$\|T\|$	# of subschemata	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

- Completeness: For a cover v we measure $C_v(s)$.
- Runtime: the execution time of each algorithm.

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 $8.00GB$.

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

6.2 Ambiguity-Completeness Tradeoff

In the first set of experiments we solve the DBMC problem starting with $\bar{H}_l = \bar{1}$ and then relaxing completeness by assigning a 0 value to some elements of the $\bar{H}_l$ vector.

Figure 3 provides the tradeoff between the two parameters for one schema pair. Other pairs show similar behavior. As expected, there is a tradeoff between ambiguity and completeness, where an increased completeness is necessarily accompanied with an increased ambiguity. Specifying both $\bar{H}_l$ and $\bar{H}_u$ allows a designer the flexibility to choose where on the curve to seek a solution.

In the next set of experiments we have taken real-world schemata of varying sizes (ranging from 30 to 40 attributes), and tested the impact of various presence constraints. As a baseline, we have applied a $k = 0$ reduction of presence, allowing maximum ambiguity per attribute and setting $\bar{H}_u$ accordingly. Even at this level some attributes may not find a correspondence, due to a threshold constraint that is applied by AMC, in which case their lower bound is set to 0. Then, we started reducing the values of $\bar{H}_u$ by increasing k and checking for completeness.

Table 5 illustrates the impact of the presence constraint on SBMC and DBMC and the tradeoff between ambiguity and completeness. Each table's row represents the average of completeness for schemata of a different size. Each column represents the k-reduction in ambiguity for SBMC and DBMC. Each entry is the average completeness of schemata of a given size with a different k-reduction value and one of SBMC and DBMC.

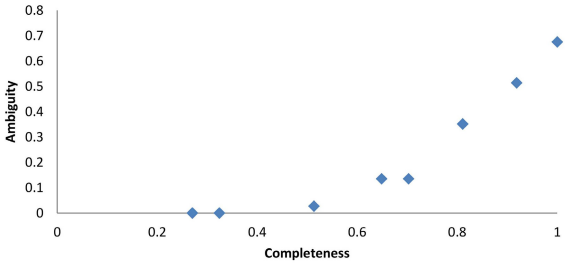


Fig. 3. Ambiguity vs. Completeness

Table 5. Impact of Presence Constraint

k-reduce →	0		1		2		3		4	
↓ Schema size	SBMC	DBMC	SBMC	DBMC	SBMC	DBMC	SBMC	DBMC	SBMC	DBMC
30	0.63	0.63	0.6		0.57		0.57		0.57	
31	0.58	0.58	0.48		0.48		0.48		0.48	
32	0.72	0.72	0.72	0.72	0.69		0.69		0.63	
33	0.61	0.61	0.55		0.52		0.52		0.52	
34	0.56	0.56	0.56	0.56	0.56	0.56	0.56	0.56	0.56	0.56
35	0.43	0.43	0.43	0.43	0.43	0.43	0.43	0.43	0.43	0.43
36	0.36	0.36	0.25	0.36	0.28	0.36	0.28	0.36	0.28	0.36
37	0.76	0.76	0.68		0.49		0.49		0.46	
38	0.53	0.53	0.37		0.37		0.32		0.29	
39	0.51	0.51	0.41		0.41		0.41		0.36	
40	0.7	0.7	0.68	0.7	0.58	0.7	0.58	0.7	0.53	

For DBMC, completeness remains fixed, since $\bar{H}_l$ does not change with increased k. Therefore, whenever the upper limit of presence does not allow certain attributes to be mapped, DBMC returns no solution, marked with a blank cell in the table.

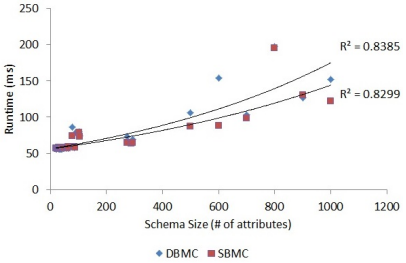
We observe that whenever a full-cover solution (baseline completeness) is possible, DBMC by definition covers all possible attributes while SBMC does not necessarily do so. For example, for a schema of size 40, completeness reduces with an increased k in solutions for SBMC, while up to $k = 3$, a solution that cover all possible attributes is still found with DBMC.

An interesting anomaly can be seen for a schema of size 36. Here, when k increases from 1 to 2, meaning that a tighter presence constraint is applied, the completeness for SBMC actually increases (from 9 to 10). This may indicate some instability in the performance of SBMC.

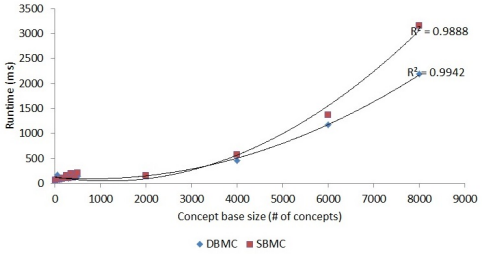
6.3 Runtime

Figure 4a demonstrates the increase of runtime with schema size, using both real-world and synthetic datasets. As expected, the runtime of the ILP-based solutions demonstrate an exponential runtime trend,[4] yet even for large schemata (of size up to 1,000 attributes) cover is performed in less than 200ms.

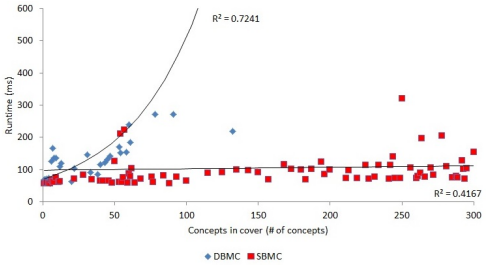
[4] Trendlines were generated using MS-Excel.



(a) A function of schema size



(b) A function of concept base size



(c) A function of number of concepts in a cover

Fig. 4. Runtime analysis

Figures 4b illustrates the impact of concept base size on runtime. Runtime increases with concept base size, albeit with a quadratic runtime trend, handling concept bases of 8,000 concepts in a few seconds.

Figure 4c illustrates the impact of the number of concepts in a cover on the execution time. The correlations is less conclusive, with low R^2 values. DBMC exhibits an exponential correlation. SBMC finds covers with more concepts than DBMC due to its greedy approach, while maintaining a lower execution time.

6.4 Discussion

Our empirical analysis covers tradeoff and run-time analysis 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 schemata and big concept bases. Using presence constraints with DBMC, when given in moderation, help in identifying better interpretation with higher completeness.

7 Related Work

A definition of a schema cover was first introduced by Saha et al. [4]. In this work we extend the cover definition to a general linear constraint optimization, and offer a new algorithm 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 outperforms the one proposed in [4], in its ability to provide higher completeness for a given level of ambiguity.

Our work on schema covering builds on schema matching techniques. We provide next a brief overview of major achievements in schema matching modeling, although it is not the focus of this work. 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 [1,21,22,23], books [24,10], and online lists, *e.g.*, OntologyMatching[5] and Ziegler[6]) 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 [25,26]. Semi-automatic schema matching has been justified in the literature using arguments of scalability (especially for matching large schemata [27], where schema covering can be specifically useful) 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 [28].

Over the years, a significant body of work was devoted to the identification of *schema matchers*, heuristics for schema matching. Examples include COMA [9], Cupid [13], OntoBuilder [14], Autoplex [15], Similarity Flooding [29], Clio [30], Glue [31], to name just a few. The main objective of schema matchers is to provide schema matchings that will be effective from the user point of view, yet computationally efficient (or at least not disastrously expensive). Such research has evolved in different research communities, including databases, information retrieval, information sciences, data semantics and the semantic Web, and others.

8 Conclusions

We have presented a general framework for schema cover. 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 completeness for a given ambiguity. It also shows the efficiency of ILP encoding of cover problems, by running experiments on schemata with up to 1,000 attributes and a concept repository with up to 8,000 concepts.

As part of an ongoing work, due to the complexity of the ILP solution we also created a heuristic framework that follows a simple scheme: we first order the set $\mathcal{E}(T_s, C)$ using a simple scoring function and then process this ordered list, one by one. At 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

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

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

the upper bound, we add this element to the set of candidates. We also maintain temporary presence 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. We intend to report on this framework and on our empirical evaluation with it in an extended version of this work.

We believe that cover formulation is an essential component in the toolkit of data integration. It is implemented as part of the NisB platform,[7] 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. This research has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 256955. This research is partially supported by JAPAN TECHNION SOCIETY RESEARCH FUND.

References

1. Batini, C., Lenzerini, M., Navathe, S.: A comparative analysis of methodologies for database schema integration. ACM Computing Surveys 18(4), 323–364 (1986)
2. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, pp. 233–246 (2002)
3. Bernstein, P., Melnik, S.: Meta data management. In: Proc. 20th Int. Conf. on Data Engineering, tutorial Presentation (2004)
4. Saha, B., Stanoi, I., Clarkson, K.: Schema covering: a step towards enabling reuse in information integration. In: Proc. 26th Int. Conf. on Data Engineering, pp. 285–296 (2010)
5. Melnik, S.: Generic Model Management: Concepts and Algorithms. Springer (2004)
6. Lee, M., Yang, L., Hsu, W., Yang, X.: XCLUST: Clustering XML schemas for effective integration. In: Proceedings of the International Conference on Information and Knowledge Management (CIKM), pp. 292–299. ACM Press, McLean (2002)
7. Smith, K., Morse, M., Mork, P., Li, M., Rosenthal, A., Allen, D., Seligman, L.: The role of schema matching in large enterprises. In: CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA (January 2009)
8. An, Y., Borgida, A., Miller, R., Mylopoulos, J.: A semantic approach to discovering schema mapping expressions. In: Proceedings of the IEEE CS International Conference on Data Engineering, pp. 206–215 (2007)
9. Do, H., Rahm, E.: COMA - a system for flexible combination of schema matching approaches. In: Proc. 28th Int. Conf. on Very Large Data Bases, pp. 610–621 (2002)
10. Gal, A.: Uncertain Schema Matching. Synthesis Lectures on Data Management. Morgan & Claypool Publishers (2011)
11. He, B., Chang, K.C.-C.: Statistical schema matching across Web query interfaces. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 217–228. ACM Press, San Diego (2003)
12. Su, W., Wang, J., Lochovsky, F.H.: Holistic schema matching for Web query interfaces. In: Ioannidis, Y., et al. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 77–94. Springer, Heidelberg (2006)

[7] http://www.nisb-project.eu/

13. Madhavan, J., Bernstein, P., Rahm, E.: Generic schema matching with Cupid. In: Proc. 27th Int. Conf. on Very Large Data Bases, Rome, Italy, pp. 49–58 (September 2001)
14. Gal, A., Modica, G., Jamil, H., Eyal, A.: Automatic ontology matching using application semantics. AI Magazine 26(1), 21–32 (2005)
15. Berlin, J., Motro, A.: Autoplex: Automated discovery of content for virtual databases. In: Batini, C., Giunchiglia, F., Giorgini, P., Mecella, M. (eds.) CoopIS 2001. LNCS, vol. 2172, pp. 108–122. Springer, Heidelberg (2001)
16. Doan, A., Domingos, P., Halevy, A.: Reconciling schemas of disparate data sources: A machine-learning approach. In: Aref, W.G. (ed.) Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 509–520. ACM Press, Santa Barbara (2001)
17. Madhavan, J., Bernstein, P., Doan, A., Halevy, A.: Corpus-based schema matching. In: Proc. 21st Int. Conf. on Data Engineering, pp. 57–68. IEEE Computer Society, Los Alamitos (2005)
18. Lee, Y., Sayyadian, M., Doan, A., Rosenthal, A.: eTuner: tuning schema matching software using synthetic scenarios. VLDB J. 16(1), 97–122 (2007)
19. Karp, R.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
20. MOSEK, The MOSEK Optimization Tools Version 6.0 (revision 61) (2009), `http://www.mosek.com`
21. Sheth, A., Larson, J.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv. 22(3), 183–236 (1990)
22. Rahm, E., Bernstein, P.: A survey of approaches to automatic schema matching. VLDB J. 10(4), 334–350 (2001)
23. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. In: Spaccapietra, S. (ed.) Journal on Data Semantics IV. LNCS, vol. 3730, pp. 146–171. Springer, Heidelberg (2005)
24. Bellahsene, Z., Bonifati, A., Rahm, E. (eds.): Schema Matching and Mapping. Springer (2011)
25. Convent, B.: Unsolvable problems related to the view integration approach. In: Atzeni, P., Ausiello, G. (eds.) ICDT 1986. LNCS, vol. 243, pp. 141–156. Springer, Heidelberg (1986)
26. Hull, R.: Managing semantic heterogeneity in databases: A theoretical perspective. In: Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), pp. 51–61. ACM Press (1997)
27. He, B., Chang, K.-C.: Making holistic schema matching robust: an ensemble approach. In: Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 429–438 (2005)
28. Srivastava, B., Koehler, J.: Web service composition - Current solutions and open problems. In: Workshop on Planning for Web Services (ICAPS 2003), Trento, Italy (2003)
29. Melnik, S., Rahm, E., Bernstein, P.: Rondo: A programming platform for generic model management. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 193–204. ACM Press, San Diego (2003)
30. Miller, R., Hernàndez, M., Haas, L., Yan, L.-L., Ho, C., Fagin, R., Popa, L.: The Clio project: Managing heterogeneity. SIGMOD Record 30(1), 78–83 (2001)
31. Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Learning to map between ontologies on the semantic web. In: Proc. 11th Int. World Wide Web Conf., pp. 662–673. ACM Press (2002)