Towards a General Framework for Effective Solutions to the Data Mapping Problem

George H.L. Fletcher[1,⋆] and Catharine M. Wyss[2]

[1] Department of Mathematics and Computer Science
Eindhoven University of Technology, The Netherlands
g.h.l.fletcher@tue.nl

[2] School of Informatics
Indiana University, Bloomington, USA
cmw@cs.indiana.edu

Abstract. Automating the discovery of mappings between structured data sources is a long standing and important problem in data management. We discuss the rich history of the problem and the variety of technical solutions advanced in the database community over the previous four decades. Based on this discussion, we develop a basic statement of the data mapping problem and a general framework for reasoning about the design space of system solutions to the problem. We then concretely illustrate the framework with the Tupelo system for data mapping discovery, focusing on the important common case of relational data sources. Treating mapping discovery as example-driven search in a space of transformations, Tupelo generates queries encompassing the full range of structural and semantic heterogeneities encountered in relational data mapping. Hence, Tupelo is applicable in a wide range of data mapping scenarios. Finally, we present the results of extensive empirical validation, both on synthetic and real world datasets, indicating that the system is both viable and effective.

Keywords: data mapping, data integration, schema matching, schema mapping, data exchange, metadata, data heterogeneity.

1 Introduction

The emerging networked world promises new possibilities for information sharing and collaboration. These possibilities will be fostered in large part by technologies which facilitate the cooperation of autonomous data sources. Created and evolving in isolation, such data sources are maintained according to local constraints and usage. Consequently, facilitating technologies must bridge a wide variety of heterogeneities, such as differences at the system level, differences in the structuring of data, and semantic pluralism in the interpretation of data.

The world-wide-web and its myriad supporting technologies have proven very successful for overcoming the system-level heterogeneities which arise in data

[⋆] Corresponding author.

S. Spaccapietra, L. Delcambre (Eds.): Journal on Data Semantics XIV, LNCS 5880, pp. 37–73, 2009.

FlightsA

Flights:

Carrier	Fee	ATL29	ORD17
AirEast	15	100	110
JetWest	16	200	220

FlightsB

Prices:

Carrier	Route	Cost	AgentFee
AirEast	ATL29	100	15
JetWest	ATL29	200	16
AirEast	ORD17	110	15
JetWest	ORD17	220	16

FlightsC

AirEast:

Route	BaseCost	TotalCost
ATL29	100	115
ORD17	110	125

JetWest:

Route	BaseCost	TotalCost
ATL29	200	216
ORD17	220	236

Fig. 1. Three airline flight price databases, each with the same information content

sharing. However, these technologies have not addressed the difficult forms of data-level heterogeneity. At the heart of overcoming data heterogeneity is the *data mapping problem*: automating the discovery of effective mappings between autonomous data sources. The data mapping problem remains one of the longest standing issues in data management [44,56]. Data mapping is fundamental in data cleaning [10,65], data integration [48], and semantic integration [36,62]. Furthermore, mappings are the basic glue for constructing large-scale semantic web and peer-to-peer information systems [41,70]. Consequently, the data mapping problem has a wide variety of manifestations such as schema matching [6,13,26,69], schema mapping [4,31,44,81], ontology alignment [16,38,75], and model matching [56].

Fully automating the discovery of data mappings has long been recognized as a "100-year" "AI-complete" problem [24,42,52,56,74]. Consequently, solutions have typically focused on discovering simple mappings such as attribute-to-attribute schema matching [64,69]. More robust solutions to the problem must not only discover such restricted mappings, but also facilitate the discovery of the structural transformations [21,47,58,78] and complex (many-to-one-attribute) semantic functions [12,34,36,62] which inevitably arise in coordinating heterogeneous information systems [39].

Example 1. Consider three relational databases Flights A, B, and C maintaining cost information for airline routes, as shown in Fig. 1. These databases, which exhibit three different natural representations of the same information, could be managed by independent travel agencies wishing to share data. Note that mapping between these databases requires (1) matching schema elements, (2) dynamic data-metadata restructuring, and (3) complex semantic mapping. For example, mapping data from FlightsB to FlightsA involves (1) matching the Flights and Prices table names and (2) promoting data values in the Route column to attribute names. Promoting these values will dynamically create as many new attribute names as there are `Route` values in the instance of `FlightsB`. Mapping the data in `FlightsB` to `FlightsC` requires (3) a complex semantic function mapping the sum of `Cost` and `AgentFee` to the `TotalCost` column in the relations of `FlightsC`. □

To better understand the design space of general solutions for the full data mapping problem, it is necessary to take a step back from particular instances of the problem. Such a broad perspective provides insight into the crucial aspects of the problem and into fundamental design techniques which can in turn be applied towards more robust and efficient solutions to particular data mapping scenarios.

Overview. Recognizing that data mapping is an AI-complete challenge, we study various facets of the problem with an eye towards developing a better understanding of the generic design space of data mapping solutions. In this investigation, we strive towards understanding both what data mappings are and how to go about discovering them. Our primary contributions are:

- A novel abstract definition of the data mapping problem and an application of this definition to the important special case of relational data sources (Section 2);
- a novel generic architecture for the design of effective solutions to the data mapping problem (Section 2.4); and
- an instantiation and evaluation of this architecture in the Tupelo data mapping system (Section 3), which applies an example-driven methodology for mapping discovery between relational data sources.

During the course of the paper, related research efforts are highlighted. We conclude with a discussion of research directions which build on these contributions (Section 4). The second half of this paper revises and extends [20].

2 The Data Mapping Problem

The data mapping problem has deep historical roots. We begin this Section with a brief account of the data mapping problem as it arose as a theoretical and technological problem. We then turn our attention to a formalization of this discussion, as a foundation for making practical contributions on the problem. Finally, we close the Section with an application of this formalism to the important special case of mapping between relational databases. An outcome of this discussion is a general design framework for mapping-discovery systems.

2.1 Perspectives on the Data Mapping Problem

We briefly highlight the historical roots of the data mapping problem. This account argues that data mapping is one of the oldest intellectual and practical concerns of science. We aim to show the ubiquity and generality of the problem, beyond technical motivations.

Philosophical, Linguistic, and Cognitive Roots. One could argue that the philosophical problem of *communication*, a concern since the earliest of Greek philosophers, is a manifestation of the data mapping problem. Indeed, the perplexing question of how it is that two speakers come to some common agreement during conversation can be recast as a question of how differences in perspective are resolved through mapping between world-views.

Early philosophical considerations set the stage for a wide-ranging discussion, which continues to this day, concerning the semantics, interpretation, and origins of natural and artificial languages (a nice overview of these investigations can be found in [33]). Of particular interest for background on the data mapping problem, semiotician Umberto Eco has documented the long struggle to overcome the perceived problems which stem from language and worldview heterogeneity [15]. Eco highlights early efforts on developing "universal" ontologies and artificial languages, such as those proposed in the 17th century by Dalgarno and Wilkins and more recent efforts such as the Esperanto movement and research on knowledge representation. In many ways, ongoing research efforts towards building universal knowledge bases are a continuation of this long-standing effort towards resolving, once-and-for-all, syntactic and semantic data heterogeneity [55]. Of course, outside of information systems research, investigators in linguistics and cognitive science have also focused intense sustained effort on resolving the inherent problems of mapping between heterogeneous conceptual models in biological and artificial communicative systems, e.g., [17].

Technological Roots. In the field of information systems, it was recognized early on that data mapping is a fundamental aspect of managing any large collection of data. From pioneering work on database reorganization in systems such as ExPress developed at IBM [68] in the mid-1970s, to work in the 1980s and 1990s in database schema integration [3], interoperability [54,67], and schema matching [64,69], data mapping has arisen in a wide variety of forms and guises. Moving to the late 1990's and 2000's, data mapping has resurfaced in recent work in ontology management [16,38,70,71]. In each of these areas, a key problem has been the discovery of transformations for mapping data between heterogeneous data representations. Much of this research has assumed that human users will provide these vital pieces which glue together information systems. Only recently have there been efforts to automate some aspects of the discovery of data mappings.

2.2 A Formal Presentation of the Data Mapping Problem

In this Section we give a formal generic presentation of the data mapping problem which generalizes and strives towards making more actionable the historical discussion of Section 2.1. This formalization allows us to focus on the essential aspects of the technological problem, and provides a foundation for further practical progress in the design and construction of automated data mapping solutions. There have been intense research efforts on data mapping formalisms. Recent key examples include the formalisms of Calvanese et al., Grahne & Kiricenko, and Lenzerini for data integration [9,29,48], the model management framework of Melnik et al. [56], and frameworks for ontology mapping [16,38,70,71]. Our formalism encompasses and extends its predecessors within a generalized statement of the technical *problem* of data mapping.

The Structure of Data Schemata. In data mapping, we are concerned with discovering mappings between *data schemata*, which are clearly delineated classes of structured *data objects*. For the purposes of our generic discussion in this Section, the internal structure of data objects and particular mechanisms for their construction are unimportant; therefore, we simply posit a universe $\mathbb{O}$ of data objects. A *data model* is a formalism for concretely describing the structuring of atomic data into data objects [1].

Definition 1. *A* data model $\mathfrak{M}$ *is a computable subset of* $\mathbb{O}$*.*

Examples of well-known concrete data models include the relational, XML, nested relational, and OODB data models [1].

A data(base) schema is a description of a class of data objects in terms of a particular data model [1]. For our purposes here, it is only important that a schema identifies a well-defined subset of the objects in a data model.

Definition 2. *A* data schema *S in a data model $\mathfrak{M}$ is a computable boolean function from $\mathfrak{M}$ to the set $\{\top, \bot\}$. A data object $D \in \mathfrak{M}$ is said to be* valid with respect to *S if $S(D) = \top$. We will call the set of all valid data objects with respect to S, denoted $\mathcal{D}_S = \{D \in \mathfrak{M} \mid S(D) = \top\}$, the* extension of *$S$.*

We will sometimes conflate a schema S and its extension $\mathcal{D}_S$, when it is clear from context.

The Structure of Data Mappings. We next define data mappings between schemata.

Definition 3. *A* data mapping from a schema *S* to a schema *T is a binary relation $\varphi \subseteq \mathcal{D}_S \times \mathcal{D}_T$.*

By not requiring data mappings to be *functional* relations,[1] this definition accommodates probabilistic, incomplete, and uncertain data management scenarios [11,29]. We further note that Definition 3 does not restrict us to considering *computable* data mappings. This flexibility in the formalism is likewise necessary to accommodate the wide range of possibilities for mapping scenarios. In many practical cases, however, the data mappings under consideration will not be quite so problematic.

Example 2. Consider data models

$$\mathfrak{M}_{\text{source}} = \{D^s_a, D^s_b, D^s_c, D^s_d\} \qquad \mathfrak{M}_{\text{target}} = \{D^t_a, D^t_b, D^t_c, D^t_d, D^t_e\},$$

[1] i.e., requiring that $\forall D \in \mathcal{D}_S$ it must be the case that $|\varphi(D)| = 1$.

schemata

$$\mathcal{D}_S = \{D^s_a, D^s_b\} \qquad\qquad \mathcal{D}_T = \{D^t_a, D^t_b, D^t_c\}$$

in $\mathfrak{M}_{\text{source}}$ and $\mathfrak{M}_{\text{target}}$, resp., and the following binary relations in $\mathfrak{M}_{\text{source}} \times \mathfrak{M}_{\text{target}}$:

$$\begin{aligned}
\varphi &= \{(D^s_a, D^t_a), (D^s_b, D^t_a)\} \\
\psi &= \{(D^s_a, D^t_a), (D^s_a, D^t_b), (D^s_a, D^t_c), (D^s_b, D^t_a)\} \\
\chi &= \{(D^s_a, D^t_a)\}.
\end{aligned}$$

Then φ is a left-total functional data mapping, ψ is a non-functional data mapping, and χ is partial functional data mapping, each from $\mathcal{D}_S$ to $\mathcal{D}_T$. □

This example illustrates the special case of *finite* (and hence computable) data mappings. If mappings are to be specified by a human expert, then this will indeed be the case. Recently, a theoretical analysis has been undertaken for this important scenario [18].

In mapping discovery, we are ultimately interested in specifying data mappings in some concrete syntax; we capture this as follows.

Definition 4. *Let $\mathfrak{M}_{source}$ and $\mathfrak{M}_{target}$ be data models. A $\mathfrak{M}_{source} \triangleright \mathfrak{M}_{target}$* mapping language *is a pair $\langle \mathcal{E}, [\![\cdot]\!] \rangle$, where:*

- *$\mathcal{E}$ is a computable set of finite strings over a finite alphabet, and*
- *$[\![\cdot]\!]$ is a computable function which maps each element of $\mathcal{E}$ to a data mapping φ, where φ is from a schema in $\mathfrak{M}_{source}$ to a schema in $\mathfrak{M}_{target}$.*

Elements of $\mathcal{E}$ are called mapping expressions. *We will use "$E \in \mathscr{L}$" as shorthand for the statement "$E \in \mathcal{E}$ for mapping language $\mathscr{L} = \langle \mathcal{E}, [\![\cdot]\!] \rangle$."*

Intuitively, $\mathcal{E}$ is the set of expressions (i.e., finite syntactic objects) of some well-defined mapping language (e.g., the relational algebra or XPath), and $[\![\cdot]\!]$ is the semantic evaluation function for the language which defines the meaning of expressions in terms of data objects in $\mathfrak{M}_{\text{source}}$ and $\mathfrak{M}_{\text{target}}$.

Before we move on to define the general data mapping problem, it is worthwhile to make the following observations. As we saw in Section 1 and Section 2.1, data mapping is pervasive in information systems and is intimately bound up not only in technological concerns but also in social concerns, since it is human activities and interests which are ultimately facilitated by these systems. In striving to capture this, it may appear that our abstract definitions of data models and mappings become too permissive and open-ended. We argue, however, that it is worthwhile to attempt to address as much of this problem space as possible at the outset, and then move on to special cases where technological progress can be made. Of course, our interests are strictly technological; when we consider specific data mapping scenarios, this abstract structure becomes grounded in actionable data models and mapping languages, as we will see in Section 2.3.

Defining the Data Mapping Problem. We are now prepared to state the general data mapping problem (DMP).

> DMP. Let S and T be data schemata in data models $\mathfrak{M}_{\text{source}}$ and $\mathfrak{M}_{\text{target}}$, respectively; φ be a data mapping from $\mathcal{D}_S$ to $\mathcal{D}_T$; and $\mathscr{L}$ be a $\mathfrak{M}_{\text{source}} \triangleright \mathfrak{M}_{\text{target}}$ mapping language. Does there exist a mapping expression $E \in \mathscr{L}$ such that $[\![E]\!] = \varphi$?

The intuition behind this characterization of the data mapping problem is as follows: during data mapping discovery, an ideal "oracle" mapping φ is typically elicited informally from a human expert (perhaps interactively using a graphical user interface in a piece-meal, step-wise fashion) or is otherwise assumed to exist (and to be verifiable), and the task at hand is to semi-automatically discover a mapping expression E in some appropriate concrete executable mapping language $\mathscr{L}$ (such as SQL, XSLT, or probabilistic relational algebra) such that the behavior of E on data objects in $\mathcal{D}_S$ is precisely that of φ.[2]

2.3 Data Mapping in Relational Databases

Note that DMP is really a *template* for specific data mapping problems. We concern ourselves in the balance of this paper with instances of DMP where the source and target schemata, S and T, are both relational, and the mapping language $\mathscr{L}$ is a relational database *query language*.[3] In this section we present the specific details of the data mapping problem for relational data sources. This concrete presentation will follow the formalism of Section 2.2. Unlike the formal presentation, however, we will now be concerned with the internal structure of data objects. Although we focus on relational databases, we note that the discussion which follows in the balance of this paper is illustrative of any data mapping scenario where the source and target schemata are structured, and the mapping language is an appropriate database query language. For example, our general approach can be readily transferred to mapping scenarios involving XML data sources (or a mix of sources from various structured data models) and mapping languages such as XPath or XSLT.

Relational Data Model. We follow a variation of the general framework for relational data objects as presented by Wyss et al. [78] and the uniform data model of Jain et al. [37]. In short, we have that: a *tuple* is a finite set of ordered pairs of uninterpreted symbols (i.e., attribute-value pairs); a *relation* is a named finite set of tuples; and a *database* is a named finite set of uniquely named relations. The *schema* of a relation is its name taken together with the set of attribute names of its constituent tuples; the schema of a database is its name

[2] It is also interesting to consider an extension of DMP, where E is only required to *approximate* the behavior of φ within a given error-bound.

[3] i.e., a language which specifies mappings on schemas in the relational data model which are computable and generic partial-functions [1].

taken together with the set of schemata of its relations. All symbols (including relation and database names) are assumed to be from some enumerable domain $\mathbb{U}$ of uninterpreted atomic objects (e.g., Unicode strings, JPEG images, MP3 music files, PDF documents, etc.).

Example 3. Consider the `FlightsA` database from Figure 1. In the relational data model, this database has encoding $\langle \texttt{FlightsA}, D\rangle$ where

$$D = \\ \Big\{\Big\langle \texttt{Flights}, \Big\{ \langle \texttt{Carrier}, \texttt{AirEast}\rangle, \langle \texttt{Fee}, \texttt{15}\rangle, \langle \texttt{ATL29}, \texttt{100}\rangle, \langle \texttt{ORD17}, \texttt{110}\rangle \Big\}\Big\rangle, \\ \Big\langle \texttt{Flights}, \Big\{ \langle \texttt{Carrier}, \texttt{JetWest}\rangle, \langle \texttt{Fee}, \texttt{16}\rangle, \langle \texttt{ATL29}, \texttt{200}\rangle, \langle \texttt{ORD17}, \texttt{220}\rangle \Big\}\Big\rangle\Big\}.$$

□

Relational Mapping Languages. Research on mapping languages (i.e., the set $\mathscr{L}$ in the DMP definition) for the relational data model has been going strong for over 30 years. At the core of almost all of these languages is the relational algebra (RA). In what follows, we assume familiarity with RA.

Example 4. Recall database $\langle \texttt{FlightsA}, D\rangle$ from Example 3. Suppose we wish to extract `Carrier` values from this database and place the output in a relation named `Companies`. The following RA query does the trick:

$$\rho^{\mathsf{rel}}_{\texttt{Flights}\to\texttt{Companies}}(\pi_{\texttt{Carrier}}(\langle \texttt{FlightsA}, D\rangle)) = \langle \texttt{FlightsA}, D'\rangle$$

where the superscript rel on the rename operator ρ indicates relation renaming, and

$$D' = \\ \{\langle \texttt{Companies}, \{\langle \texttt{Carrier}, \texttt{AirEast}\rangle\}\rangle, \langle \texttt{Companies}, \{\langle \texttt{Carrier}, \texttt{JetWest}\rangle\}\rangle\}.$$

□

For an overview of the rich variety of relational mapping languages, see [1].

Relational Data Mapping Problem. With the relational data model and RA as an example relational mapping language in hand, we are now in a position to turn to a concrete presentation of RelationalDMP, the DMP for relational data sources.

> RelationalDMP. Let S and T be relational data schemata, φ be a data mapping from $\mathcal{D}_S$ to $\mathcal{D}_T$, and $\mathscr{L}$ be a relational query language. Does there exist an expression $E \in \mathscr{L}$ such that $[\![E]\!] = \varphi$?

For the balance of this paper, we will be concerned with investigating this important subclass of DMP problems.

2.4 A Framework for Data Mapping Systems

We now turn to a general overview of the design and construction of solutions for instances of RelationalDMP. Our design is driven by the following crucial observation, which encapsulates a long-running analysis of the various subproblems of RelationalDMP in the data mapping literature (e.g., [36,39,43,47,56,58]):

> *Data mapping discovery for structured data sources consists of two distinct principle tasks: discovery of semantic functions and discovery of structural mapping queries.*

Semantic functions operate at the token level (i.e., operate on tokens in $\mathbb{U}$), mapping data values between data sources. These functions interpret the internal structure of atoms in $\mathbb{U}$, and hence rely on information which is external to the information systems of which a data mapping system is a component. This information can be codified (for example) in an ontology, but it is important to note that our work does not presuppose pre-existing ontologies, nor even a shared vocabulary of tokens.

In RelationalDMP, *mapping queries* operate at the structural level, corresponding to traditional structural database transformations between database schemas. Discovering semantic functions and discovering mapping queries both require unsupervised learning from data instances and/or supervised learning using domain knowledge. For these tasks it is possible to leverage the large body of techniques which have been developed over the last century in the Machine Learning, Artificial Intelligence, and Data Mining communities [61].

We propose a generic architecture for RelationalDMP solutions which reflects our design observation, illustrated in Figure 2. Input to the mapping discovery process includes, at the very least, source/target database schemas and instances. If available, the discovery process can also use domain knowledge elicited from external sources (e.g., human input, system logs, etc.). This architecture, which we now outline, cleanly captures the division of labor in data mapping implied by the design thesis.

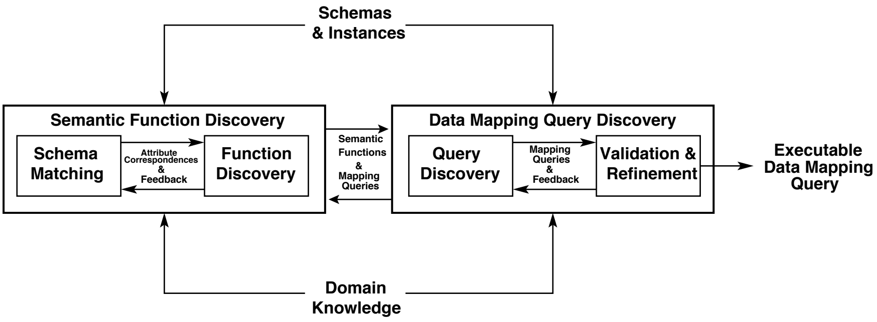

Schemas & Instances
Semantic Function Discovery
Schema Matching
Attribute Correspondences & Feedback
Function Discovery
Semantic Functions & Mapping Queries
Data Mapping Query Discovery
Query Discovery
Mapping Queries & Feedback
Validation & Refinement
Executable Data Mapping Query
Domain Knowledge

Fig. 2. Generic architecture for discovery of executable data mapping queries

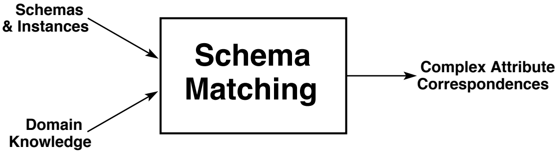


Fig. 3. Complex attribute correspondence discovery

Discovering Semantic Functions. The first step in data mapping is to discover semantic functions (Figure 2). This process involves (1) *schema matching* – identifying the correspondences between those attributes of the source and target schemas that are semantically related – and then (2) discovering the semantic functions which provide the actual mappings between the data values of these corresponding attributes. Note that this may be an iterative process, with information gained during function discovery used in further schema matching (Figure 2).

Discovery of Complex Attribute Correspondences. Schema matching takes as input the source and target schemas, S and T, and instances if available. If available, domain knowledge can also be used to supervise the discovery of attribute correspondences (Figure 3). Together, these inputs serve to guide the selection of good pairings between the schema elements of the source S and the target T. This process can be abstractly presented as follows. The relationship between S and T is encapsulated as a boolean function M on the set $\mathcal{P}(\mathsf{schema}(S)) \times \mathsf{schema}(T)$, and is typically defined externally to the schema matching mechanism itself.[4] The output of this component is a set of complex (i.e., many-one) correspondences [22,64] between the attributes of S and those of T that satisfy M:[5]

$$\{(\overline{\mathsf{a}}_{\text{source}}, \mathsf{a}_{\text{target}}) \mid \overline{\mathsf{a}}_{\text{source}} \subseteq \mathsf{schema}(S), \mathsf{a}_{\text{target}} \in \mathsf{schema}(T), \\ \& \ M(\overline{\mathsf{a}}_{\text{source}}, \mathsf{a}_{\text{target}}) = \top\}.$$

Example 5. We observed in Example 1 that, during mapping discovery with source schema `FlightsB` and target schema `FlightsC`, the schema matching process would output the correspondence ({`Cost`, `AgentFee`}, `TotalCost`). □

Our abstraction of the schema matching process follows in the spirit of other such formalisms in the literature, e.g., [23,56,64].

Discovery of Functions. After attribute correspondence discovery has been handled, the next step is to use these correspondences, together with the source/

[4] Where $\mathcal{P}(\cdot)$ is the powerset operator.

[5] The case of many-many (i.e., m-n) matchings in the literature [64] reduces to a special case of many-one matchings, namely, where one is interested in a set of n many-one correspondences.

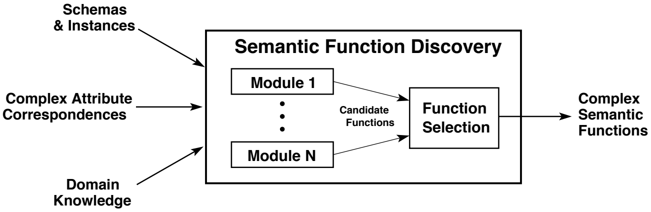


Fig. 4. Semantic function discovery

target schemas and instances, to discover the semantic functions which map between corresponding source and target data values (Figure 4). As with schema matching, domain knowledge can also be used to supervise the discovery process, if it is available. It is generally recognized that a *modular* approach to function discovery (Figure 4) is necessary to accommodate the wide variety of possibilities for token transformation scenarios [12]. Hence, our design architecture indicates specialized modules responsible for discovering specific classes of functions (e.g., string functions [12,73], date/time conversions, real-time currency exchange, image conversion, etc.). The last step then is to select the final functions from among the candidates suggested by these modules (Figure 4).

We can abstract the process of function discovery as follows. Concrete semantic functions map sets of atoms $\overline{\mathtt{s}}$ in a source token space (i.e., a subset of $\mathbb{U}$) to atoms $\mathtt{t}$ in a target token space (i.e., another subset of $\mathbb{U}$). These token spaces reside in source and target contexts of interpretation, respectively (Figure 5). In these contexts of interpretation, bijective "meaning" functions m_{source} and m_{target} associate these tokens with some objects $\overline{O}_{\mathtt{s}}$ and $O_{\mathtt{t}}$, respectively, in domains of discourse which are of interest to the users of the source and target information systems, respectively:

$$m_{\text{source}}(\overline{\mathtt{s}}) = \overline{O}_{\mathtt{s}} \qquad\qquad m_{\text{target}}(\mathtt{t}) = O_{\mathtt{t}}$$

Analogous to the match function M in schema matching, the relationships between objects in the source and target domains of discourse are encapsulated in a discourse-mapping function f^* (Figure 5), whose derivation is external to the function discovery process. Now, given a complex attribute correspondence $(\overline{\mathtt{a}}_{\text{source}}, \mathtt{a}_{\text{target}})$, the goal of semantic function discovery is to find a concrete function f (Figure 5) which maps,[6] on a per-tuple basis, any instance $\overline{\mathtt{s}}$ of attributes $\overline{\mathtt{a}}_{\text{source}}$ in the source token space to an instance $\mathtt{t}$ of attribute $\mathtt{a}_{\text{target}}$ in the target token space such that

$$f(\overline{\mathtt{s}}) = m_{\text{target}}(f^*(m_{\text{source}}(\overline{\mathtt{s}}))) = \mathtt{t}.$$

In other words, f abides by the semantics of both the source and target schemata in their contexts of use.

[6] For further discussion of the "semantics" of applying semantic functions, please see Section 3.5 below.

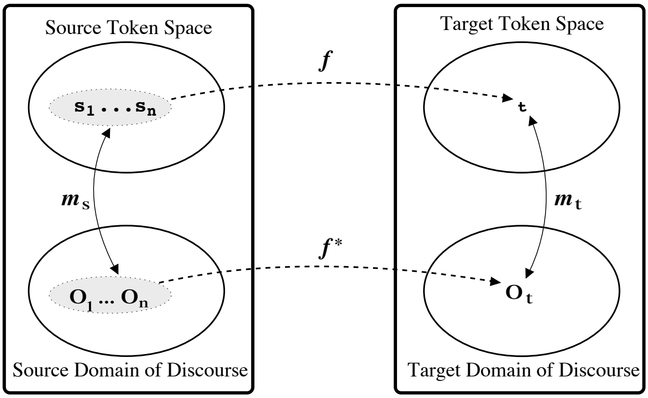


Fig. 5. Semantic mappings between contexts-of-interpretation

Example 6. As we saw in Example 5, the attributes `Cost` and `AgentFee` in database `FlightsB` correspond to the attribute `TotalCost` in the relations of database `FlightsC`. Once this correspondence has been determined, for mapping instances of `FlightsB` to instances of `FlightsC`, we require a semantic function $\boldsymbol{f}$ for this correspondence which interprets the tokens of all three attributes appropriately as numbers in the reals[7] and transforms them as follows:

$$\boldsymbol{f} : \texttt{Cost} + \texttt{AgentFee} \longmapsto \texttt{TotalCost}$$

applied to each tuple in the `Prices` relation of `FlightsB`. □

While solutions abound in the literature for one-to-one schema matching [64], the database community has only recently begun to make strong progress on the issues of *complex* (i.e., many-to-one) schema matching and semantic function discovery. Primarily, supervised approaches (i.e., using domain knowledge elicited via GUIs, etc.) have been explored in the literature [8,12,14,23,30,34,41,62,81]. The design and analysis of approaches to complex schema matching and semantic function discovery continues to be an extremely important area of investigation in data mapping.

Discovering Data Mapping Queries. After determining appropriate semantic functions, the second critical step in data mapping is to discover executable queries (Figure 2). These queries perform restructuring of data objects and apply the previously discovered semantic functions. Note that this process may be iterative. In the context of structural transformations which dynamically modify the input schemata, further rounds between semantic function discovery and query discovery may be necessary (Figure 2).

[7] Or, if appropriate, as currency values, applying exchange rates as necessary if multiple currencies are involved, etc.

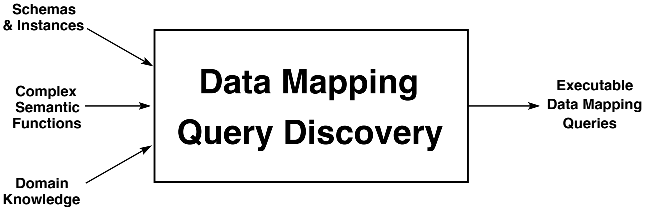

Schemas & Instances
Complex Semantic Functions
Domain Knowledge
Data Mapping Query Discovery
Executable Data Mapping Queries

Fig. 6. Data mapping query discovery

Example 7. Continuing Example 6, we recall from the discussion of Example 1 that structural mapping from the `FlightsB` schema to the `FlightsC` schema requires data-metadata transformation in a language such as the federated interoperable relational algebra, a variation of which we develop in Section 3.1 below. As we will see, this language is a natural extension of the RA that includes data-metadata structural transformations. In this algebra, the following mapping query restructures data under the `FlightsB` schema to conform with the `FlightsC` schema:

$$\underline{\mu}_{\texttt{Carrier, Cost}}(\wp_{\texttt{Carrier}}(\rho_{\texttt{Cost}\rightarrow\texttt{BaseCost}}(\lambda^{\texttt{TotalCost}}_{\boldsymbol{f},\ \texttt{Cost, AgentFee}}(\texttt{FlightsB})))).$$

Note the λ operator for application of semantic functions. In this case, the function $\boldsymbol{f}$ from Example 6 is applied to the values in the attributes `Cost` and `AgentFee`, and the results are placed in target attribute `TotalCost`. □

Source/target schemata, S and T, instances of these schemata (if available), the complex semantic functions discovered between these schemata in the previous step, and domain knowledge (if explicitly available) are given as input to the query discovery component of the mapping framework (Figure 6). Collectively, the constraints of these inputs delineate an ideal data mapping φ from S to T. The output of this process is a solution for the RelationalDMP instance $\langle S, T, \mathscr{L}, \varphi\rangle$, where $\mathscr{L}$ is an appropriate query language for S, T, and φ. In other words, the output of this module is an executable data mapping expression E (or a set of candidate mappings) in some concrete query language $\mathscr{L}$ that transforms any valid instance of schema S into a corresponding valid instance of schema T (i.e., respecting φ). The final sub-step of mapping discovery is validation and refinement of the discovered mapping(s) (Figure 2).

Query discovery is the least explored aspect of RelationalDMP(and of the general DMP). Only a handful of systems attack aspects of query discovery [4,12,31,32,40,57,59,60,66,81], and hence the problem is not very clearly recognized in the literature. Since it *is* generally recognized in the literature that executable data mapping queries are the crucial glue in the various manifestations of data mapping discussed in Section 1, clearly continued efforts on understanding DMP are warranted. We note here that to the best of our knowledge, our research in this Section and Section 3 below on building a mapping query discovery solution (i.e., an instantiation of the framework presented in this Section) is the first attempt at discovering full solutions to RelationalDMP. We postpone a fuller discussion of related query discovery research until Section 3.7, below.

Remarks. In this Section we gave an historical and novel formal presentation of the DMP. We applied this presentation to the development of the special case of RelationalDMP, and to the development of a generic framework for designing data mapping solutions for instances of RelationalDMP. This framework was based on the observation that such systems should clearly separate the discovery of transformations for semantic heterogeneity from the discovery of queries for structural heterogeneity. We next turn to the development of a system which instantiates the insights of this discussion.

3 Data Mapping as Search

We next present the Tupelo data mapping system for semi-automating the discovery of executable[8] data mapping expressions between heterogeneous relational data sources. Tupelo is an example driven system, generating mapping expressions for interoperation of heterogeneous information systems which involve schema matching, dynamic data-metadata restructuring, and complex (many-to-one) semantic functions. For example, Tupelo can generate the expressions for mapping between instances of the three airline databases in Figure 1. The design of Tupelo is guided by the generic framework for RelationalDMP solutions developed in Section 2.4.

Previous solutions have not clearly separated each of the subproblems associated with data mapping discovery – mixing, merging, and/or conflating various aspects of semantic function discovery and query discovery. This has lead to a somewhat opaque research literature with inconsistent terminology and duplication of effort. The development of Tupelo clarifies, complements, and extends the existing approaches in the literature. In particular, Tupelo is the first data mapping system to

- propose and validate the mapping query discovery process as an example-driven search problem;
- explicitly modularize the various aspects of data mapping query discovery;
- seamlessly incorporate complex semantic functions in a complete, executable mapping language; and
- generate mapping queries which incorporate the full range of data-metadata structural transformations necessary to overcome heterogeneity in relational data sources.

Data mapping in Tupelo is built on the novel perspective of mapping discovery as an example driven search problem. We develop the Tupelo mapping language $\mathscr{L}$ in Section 3.1 and the Rosetta Stone principle behind this example-driven approach in Section 3.2. We then discuss how Tupelo leverages Artificial Intelligence

[8] By *executable*, we mean that the discovered mapping queries must be in a concrete mapping language such as SQL or RA (cf., Sections 2.2-2.3, above).

(AI) search techniques to generate mapping expressions (Sections 3.3 and 3.4). After this, we discuss how generic query languages such as $\mathscr{L}$ can be extended naturally in this setting to accommodate complex semantic functions which have previously discovered (Section 3.5). We then present experimental validation of the system on a variety of synthetic and real world scenarios (Section 3.6) which indicates that the Tupelo approach to data mapping is both viable and effective. We conclude the Section with a discussion of related research (Section 3.7).

3.1 Dynamic Relational Data Mapping with Tupelo

Recall from Figure 2 and the discussion of Section 2.3 that a critical component of data mapping is the discovery of executable data mapping queries. Tupelo generates an effective mapping from a source relational schema S to a target relational schema T, under the assumption that semantic function discovery has been successfully completed. The system discovers this mapping using (1) example instances $\mathfrak{s}$ of S and $\mathfrak{t}$ of T and (2) illustrations of any complex semantic mappings between the schemas. Mapping discovery in Tupelo is a completely syntactic and structurally driven process which does not make use of a global schema or any explicit domain knowledge beyond that encapsulated in the input semantic functions [6].

The mapping language $\mathscr{L}$ used in Tupelo provides support for both simple schema matching and richer structural transformations.

FIRA. Recently, Wyss and colleagues have developed a relational language framework for *metadata integration* [77,78,79]. This framework consists of a federated relational data model, a variation of which was introduced in Section 2.3, and two equivalent relational query languages: the Federated Interoperable Relational Algebra (FIRA) and the Federated Interoperable Structured Query Language (FISQL). These languages, FIRA/FISQL, (1) are principled extensions to relational algebra/SQL (resp.) that include metadata querying and restructuring capabilities; and (2) generalize the notion of relational transpose, providing a notion of *transformational completeness* for relational metadata [78]. Applications of the FIRA/FISQL framework include OLAP, schema browsing, and real-time interoperability of relational sources in federated information systems [78]. True data integration presupposes metadata integration, and FIRA/FISQL contributes to the study of query languages specifically by advancing the understanding of languages that offer robust metadata integration capabilities.

$\mathscr{L}$: Tupelo's Take on FIRA. Tupelo generates expressions in a fragment $\mathscr{L}$ of FIRA. The operators in this fragment extend the RA (Section 2.3) with dynamic structural transformations [47,65,78]. These include operators for dynamically promoting data to attribute and relation names (i.e., to "metadata"), a simple merge operator [77], and an operator for demoting metadata to data values. The $\mathscr{L}$ operations are intuitively summarized in Table 1. A more detailed discussion and comparison of full FIRA to the wealth of alternative relational query languages can be found in [78].

Table 1. $\mathscr{L}$ operators for dynamic relational data mapping

Operation	Effect
$\pi, \cup, \times$	*Regular relational operations.*
$\rho^{\texttt{att/rel}}_{\texttt{A}\to\texttt{B}}(R)$	*Rename* attribute/relation `A` to `B` in relation R.
$\uparrow^{\texttt{B}}_{\texttt{A}}(R)$	*Promote attribute* `A` *to metadata.* $\forall t \in R$, append a new active attribute named $t[\texttt{A}]$ with value $t[\texttt{B}]$.
$\to^{\texttt{B}}_{\texttt{A}}(R)$	*Dereference attribute* `A` *on* `B`. $\forall t \in R$, append a new active attribute named B with value $t[t[A]]$.
$\downarrow_{\texttt{A,B}}(R)$	*Demote metadata.* Cartesian product of relation R with a binary relation (with schema $\{\texttt{A}, \texttt{B}\}$) containing the metadata of R.
$\wp_{\texttt{A}}(R)$	*Partition on attribute* `A`. $\forall \texttt{V} \in \pi_{\texttt{A}}(R)$, output a new relation named `V`, where $t \in \texttt{V}$ iff $t \in R$ and $t[\texttt{A}] = \texttt{V}$.
$\Sigma(\mathcal{D})$	*Generalized union of database* $\mathcal{D}$. Outputs an unnamed outer union of all relations in $\mathcal{D}$.
$\text{ɥ}_{\texttt{A}}(R)$	*Drop* attribute `A` from relation R.
$\mu_{\overline{\texttt{A}}}(R)$	*Merge* tuples in R based on compatible values in attributes $\overline{\texttt{A}}$.

Example 8. Consider the transformation of instances from `FlightsB` to instances of `FlightsA` in Figure 1. This mapping can be expressed in $\mathscr{L}$ as

$$\rho^{\mathsf{att}}_{\texttt{AgentFee}\to\texttt{Fee}}(\rho^{\mathsf{rel}}_{\texttt{Prices}\to\texttt{Flights}}(\mu_{\texttt{Carrier}}(\text{ɥ}_{\texttt{Route}}(\text{ɥ}_{\texttt{Cost}}(\uparrow^{\texttt{Cost}}_{\texttt{Route}}(\texttt{FlightsB}))))))$$

which breaks down as follows:

$R_1 := \uparrow^{\texttt{Cost}}_{\texttt{Route}}(\texttt{FlightsB})$
Promote `Route` values to attribute names with corresponding `Cost` values.
$R_2 := \text{ɥ}_{\texttt{Route}}(\text{ɥ}_{\texttt{Cost}}(R_1))$
Drop attributes `Route` and `Cost`.
$R_3 := \mu_{\texttt{Carrier}}(R_2)$
Merge tuples on `Carrier` values.
$R_4 := \rho^{\mathsf{att}}_{\texttt{AgentFee}\to\texttt{Fee}}(\rho^{\mathsf{rel}}_{\texttt{Prices}\to\texttt{Flights}}(R_3))$
Rename attribute `AgentFee` to `Fee` and relation `Prices` to `Flights`.

The output relation R_4 is exactly `FlightsA`. □

The original FIRA algebra is complete for the full data-metadata mapping space for relational data sources [78]. The fragment we use in Tupelo maintains the full data-metadata restructuring power of this language. The operators in our

$\mathscr{L}$ focus on bulk structural transformations (via the $\rightarrow$, $\uparrow$, $\downarrow$, $\wp$, $\times$, $\underline{\cup}$, and μ operators) and schema matching (via the rename operator ρ). We view application of selections (σ) as a post-processing step to filter mapping results according to external criteria, since it is known that generalizing selection conditions is a nontrivial problem [41]. Hence, Tupelo does not consider applications of the relational σ operator. Note that using a language such as $\mathscr{L}$ for data mapping blurs the distinction between schema matching (i.e., finding appropriate renamings via ρ) [64] and schema mapping [59] since $\mathscr{L}$ encompasses these basic mapping disciplines. It is for this reason that we refer to RelationalDMP (and in general, DMP) as a *data* mapping problem.

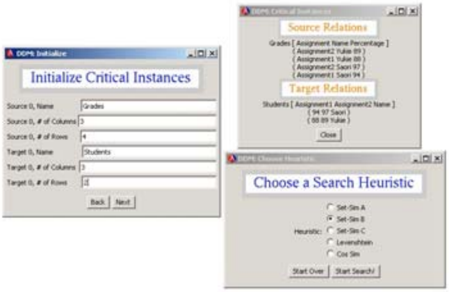
3.2 The Rosetta Stone Principle

An integral component of the Tupelo system is the notion of "critical" instances $\mathfrak{s}$ and $\mathfrak{t}$ which succinctly characterize the structure of the source and target schemas S and T, respectively. These instances illustrate the same information structured under both schemas. The *Rosetta Stone Principle* states that such critical instances can be used to drive the search for data mappings in the space of transformations delineated by the operators in $\mathscr{L}$ on the source instance $\mathfrak{s}$. Guided by this principle, Tupelo takes as input critical source and target instances which illustrate all of the appropriate restructurings between the source and target schemas.

Example 9. The instances of the three airline databases presented in Figure 1 illustrate the same information under each of the three schemas, and are examples of succinct critical instances sufficient for data mapping discovery between the `FlightsA`, `FlightsB`, and `FlightsC` databases. □

Critical Instance Input and Encoding. Critical instances can be easily elicited from a user via a visual interface akin to the Lixto data extraction system [28] or visual interfaces developed for interactive schema mapping [4,8,59,72]. In Tupelo, critical instances are articulated by a user via a front-end graphical user interface which has been developed for the system (Figure 7). Since critical instances essentially illustrate one entity under different schemas, we also envision that much of the process of generating critical instances can be semi-automated using techniques developed for entity/duplicate identification and record linkage [6,76].

Critical instances are encoded internally in *Tuple Normal Form* (TNF). This normal form, which encodes databases in single tables of fixed schema, was introduced by Litwin et al. as a standardized data format for database interoperability [53]. More recently, this flexible normal form for data has been successfully used in a variety of investigations and systems, e.g., [2,80]. Tupelo makes full use of this normal form as an internal data representation format. Given a relation R, the TNF of R is computed by first assigning each tuple in R a unique ID and then building a four column relation with attributes TID, REL, ATT, VALUE,



DDMI: Initialize

Initialize Critical Instances

Source 0, Name: Grades
Source 0, # of Columns: 3
Source 0, # of Rows: 4
Target 0, Name: Students
Target 0, # of Columns: 3
Target 0, # of Rows: 2

Source Relations

Grades [Assignment Name Percentage]
(Assignment2 Yukie 89)
(Assignment1 Yukie 88)
(Assignment2 Saori 97)
(Assignment1 Saori 94)

Target Relations

Students [Assignment1 Assignment2 Name]
(94 97 Saori)
(88 89 Yukie)

Choose a Search Heuristic

Heuristic: ○ Set-Sim A ● Set-Sim B ○ Set-Sim C ○ Levenshtein ○ Cos Sim

Fig. 7. Tupelo graphical user interface

corresponding to tuple ID, relation name, attribute name, and attribute value, respectively. The table is populated by placing each tuple in R into the new table in a piecemeal fashion. The TNF of a database is the single table consisting of the union of the TNF of each relation in the database.

Definition 5. *Let D be a database with name* d*. Given a tuple*

$$t = \{\langle a_1, v_1 \rangle, \ldots, \langle a_n, v_n \rangle\}$$

in relation r *of D, let $\mathring{t}$ denote the relation of n tuples:*

$$\mathring{t} = \Big\langle d, \{\langle \mathsf{TID}, t \rangle, \langle \mathsf{REL}, r \rangle, \langle \mathsf{ATT}, a_1 \rangle, \langle \mathsf{VALUE}, v_1 \rangle, \\ \ldots, \langle \mathsf{TID}, t \rangle, \langle \mathsf{REL}, r \rangle, \langle \mathsf{ATT}, a_n \rangle, \langle \mathsf{VALUE}, v_1 \rangle\}\Big\rangle$$

where $\mathsf{t} = f(t)$, for some injection f of D into $\mathbb{U}$.[9] *Then, the* tuple normal form *of D is the database*

$$\mathsf{TNF}(D) = \bigcup_{R \in D} \bigcup_{t \in R} \mathring{t}$$

containing a single relation named d*.*

Note that $\mathsf{TNF}(D)$ is well-defined (i.e., unique up to TID values). We will often blur the fact that $\mathsf{TNF}(D)$ is a database and treat it simply as a solitary unnamed relation.

Example 10. We illustrate TNF with the encoding of the instance of database `FlightsC` from Figure 1:

[9] i.e., t is a fresh symbol uniquely identifying tuple t.

TID	REL	ATT	VALUE
t_1	AirEast	Route	ATL29
t_1	AirEast	BaseCost	100
t_1	AirEast	TotalCost	115
t_2	AirEast	Route	ORD17
t_2	AirEast	BaseCost	110
t_2	AirEast	TotalCost	125
t_3	JetWest	Route	ATL29
t_3	JetWest	BaseCost	200
t_3	JetWest	TotalCost	216
t_4	JetWest	Route	ORD17
t_4	JetWest	BaseCost	220
t_4	JetWest	TotalCost	236

□

The TNF of a database can be built in SQL using the "system tables" of a DBMS [2,80]. The benefits of normalizing input instances in this manner with a fixed schema include (1) ease and uniformity of handling of the data; (2) both metadata and data can be handled directly in SQL; and (3) sets of relations are encoded as single tables, allowing natural multi-relational data mapping from databases to databases with the use of conventional technologies.

3.3 Data Mapping as a Search Problem

In Tupelo the data mapping problem is seen fundamentally as a search problem. Given Rosetta Stone critical instances $\mathfrak{s}$ and $\mathfrak{t}$ of the source and target schemas, data mapping is resolved as an exploration of the transformation space of $\mathscr{L}$ on the source instance $\mathfrak{s}$. Search successfully terminates when the target instance $\mathfrak{t}$ is located in this space. Upon success, the transformation path from the source to the target is returned.[10] This search process is illustrated in Figure 8. In this Section we describe this process in more detail.

Search Algorithms. We work in the classic problem-space model [45]. In particular, a *problem space* is a pair $\langle \mathbb{S}, \mathbb{F} \rangle$, where $\mathbb{S}$ is a set of *states*, and $\mathbb{F}$ is a set of *state transition* partial functions on $\mathbb{S}$. A *search problem* consists of a problem space $\langle \mathbb{S}, \mathbb{F} \rangle$, a designated *start state* $\mathfrak{s} \in \mathbb{S}$, and a *goal test* function goal which maps states in $\mathbb{S}$ to $\{\top, \bot\}$.[11] A *solution* to a search problem is a sequence of transitions $\tau_1, \ldots, \tau_n \in \mathbb{F}$ such that $\mathsf{goal}(\tau_n(\cdots \tau_1(\mathfrak{s}))) = \top$. In terms of this model, Tupelo takes as input to the search process: Rosetta Stone source instance $\mathfrak{s}$ and target instance $\mathfrak{t}$, the set of $\mathscr{L}$ transformations, and a goal test

[10] Note that there may be more than one path from $\mathfrak{s}$ to $\mathfrak{t}$; we just return the shortest solution path (i.e., smallest mapping expression). Although the current implementation does not do so, it is straightforward to extend Tupelo to (1) present a discovered solution to the user; (2) allow the user to *deny* or *confirm* the solution; and (3) if denied, continue the search for an acceptable solution. Such an adaptation is outside of the scope of this investigation.

[11] Note that in practice, the search space $\mathbb{S}$ is not explicitly represented, but rather is implicit in the start-state and set of transition functions.

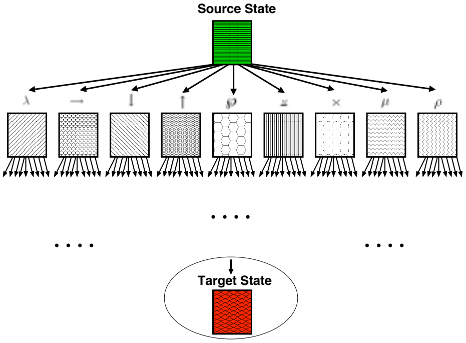


Fig. 8. Search space for data mapping discovery

which checks if a given state $\mathfrak{n}$ is a superset of $\mathfrak{t}$ (i.e., if $\mathfrak{t}$ is derivable from $\mathfrak{n}$ by filtering out tuples).

There are two general methodologies for discovering solutions to search problems: *uninformed* search and *informed* search. Both are systematic approaches to discovering a path of transformations in a search space, differing primarily in their use of external knowledge not explicit in the graph structure of the space. In particular, uninformed methods are brute-force approaches for traversing a search space, and informed methods use some external hints, in the form of *heuristics*, to guide the process. At any point during search, a choice needs to be made in selecting the next search node to visit. For each neighbor $\mathfrak{n}$ of the current state, we can assign a *cost* value via an *evaluation function*, estimating the cost to get from the start state $\mathfrak{s}$, through $\mathfrak{n}$, to a target state:

$$\mathsf{eval}(\mathfrak{n}) = \mathsf{past}(\mathfrak{n}) + \mathsf{future}(\mathfrak{n})$$

where $\mathsf{past}(\mathfrak{n})$ is the known cost of reaching $\mathfrak{n}$ from $\mathfrak{s}$, and $\mathsf{future}(\mathfrak{n})$ is an estimate of the cost of getting from $\mathfrak{n}$ to a goal state. A node with lowest cost amongst unexplored nodes is selected next for exploration. Uninformed methods ignore (or rather, do not have access to) $\mathsf{future}(\mathfrak{n})$ during this calculation of cost. Informed methods make full use of $\mathsf{future}(\mathfrak{n})$ in determining $\mathsf{eval}(\mathfrak{n})$ (and may or may not ignore $\mathsf{past}(\mathfrak{n})$). The classic "Best-First" search algorithm can be specialized to the standard uniformed Breadth-First search and informed A* search (described below) algorithms in this manner [61].

In many domains, it is difficult to construct useful heuristic $\mathsf{future}(\cdot)$ functions [61]. In such cases, one is often limited to variations of brute-force search. The branching factor of the data mapping search space is proportional to $|\mathfrak{s}| + |\mathfrak{t}|$ and hence quite high, ruling out the use of such search methods. Fortunately, data mapping is a domain where it *is* possible to develop useful search heuristics (and would be impractical if this were not the case), and using them for intelligent exploration of the search space greatly reduces the number of states visited. Hence, we focus on informed search methodologies in the balance of our discussion. We return to the issue of developing heuristics for data mapping in Section 3.4.

Informed Search in Tupelo. Due to their simplicity and effectiveness, we chose to implement the heuristic based A^*, *Iterative Deepening A** (IDA), and *Recursive Best-First Search* (RBFS) search algorithms from the AI literature [45,46,61]. In the exploration of a state space, these algorithms use a heuristic function to rank states and selectively search the space based on the rankings. The evaluation function eval() for ranking a search state $\mathfrak{n}$ is calculated as above, with past($\mathfrak{n}$) equal to the number of transitions to reach $\mathfrak{n}$ from $\mathfrak{s}$, and where future($\mathfrak{n}$) $= h(\mathfrak{n})$, for some heuristic function h which makes an "educated guess" of the distance of $\mathfrak{n}$ from the target state. Search begins at the source critical instance $\mathfrak{s}$ and continues until the current search state is a structurally identical superset of the target critical instance $\mathfrak{t}$ (i.e., the current state contains $\mathfrak{t}$). The transformation path from $\mathfrak{s}$ to $\mathfrak{t}$ gives a basic mapping expression in $\mathscr{L}$. After this expression has been discovered, filtering operations (via relational selections σ) must be applied if necessary according to external criteria, as discussed in Section 3.1. The final output of Tupelo is an expression for mapping instances of the source schema to corresponding instances of the target schema.

A^* is a special case of the general best-first search strategy [61]. A^* search is just best-first search called with an eval() function such that the future() component never *overestimates* the distance to a goal state. We used A^* search to develop search heuristics in early implementations of Tupelo. Unfortunately, the cost of maintaining the search queues quickly becomes impractical (given an exponential search space). Hence we were driven to explore memory-limited alternatives to best-first search.

The two search algorithms finally used in Tupelo, IDA and RBFS, operate under more practical conditions. In particular, each of these algorithms uses memory linear in the depth of search; although they both perform redundant explorations (compared to best-first search), they do not suffer from the exponential memory use of basic A^* best-first search which led to the ineffectiveness of early implementations of Tupelo. Furthermore, they both achieve performance asymptotically equivalent to A^*, as most of the work is done on the final search-frontier during a successful search.[12] In a nut-shell, these algorithms operate as follows:

- IDA performs a depth-bounded depth-first search of the state space using the eval()-rankings of states as the depth bound, iteratively increasing this bound until the target state is reached [45].
- RBFS performs a localized, recursive best-first exploration of the state space, keeping track of a locally optimal eval()-value and backtracking if this value is exceeded [46].

These two simple algorithms proved to be effective in the discovery of mapping expressions. To further improve performance of the search algorithms, we also

[12] In fact, they may even run faster than A^* in some cases due to lower memory management overhead [45].

employed the simple rule of thumb that “obviously inapplicable” transformations should be disregarded during search. For example if the current search state has all attribute names occurring in the target state, there is no need to explore applications of the attribute renaming operator. We incorporated several such simple rules in Tupelo.

3.4 Search Heuristics

Heuristics are used to intelligently explore a search space, as discussed in Section 3.3. A search heuristic $h(\mathfrak{n})$ estimates the distance, in terms of number of intermediate search states, of a given database $\mathfrak{n}$ from the target database $\mathfrak{t}$. A variety of heuristics were implemented and evaluated. This section briefly describes each heuristic used in Tupelo.

Set Based Similarity Heuristics. Three simple heuristics measure the overlap of values in database states. Heuristic h_1 measures the number of relation, column, and data values in the target state which are missing in state $\mathfrak{n}$:

$$\begin{aligned} h_1(\mathfrak{n}) = \quad & |\pi_{\mathsf{REL}}(\mathfrak{t}) - \pi_{\mathsf{REL}}(\mathfrak{n})| \\ & + |\pi_{\mathsf{ATT}}(\mathfrak{t}) - \pi_{\mathsf{ATT}}(\mathfrak{n})| \\ & + |\pi_{\mathsf{VALUE}}(\mathfrak{t}) - \pi_{\mathsf{VALUE}}(\mathfrak{n})|. \end{aligned}$$

Here, π is relational projection on the TNF of $\mathfrak{n}$ and $\mathfrak{t}$, and $|R|$ denotes the cardinality (i.e., number of tuples) of relation R. Heuristic h_2 measures the minimum number of data promotions ($\uparrow$) and metadata demotions ($\downarrow$) needed to transform $\mathfrak{n}$ into the target $\mathfrak{t}$:

$$\begin{aligned} h_2(\mathfrak{n}) = \quad & |\pi_{\mathsf{REL}}(\mathfrak{t}) \cap \pi_{\mathsf{ATT}}(\mathfrak{n})| \\ & + |\pi_{\mathsf{REL}}(\mathfrak{t}) \cap \pi_{\mathsf{VALUE}}(\mathfrak{n})| \\ & + |\pi_{\mathsf{ATT}}(\mathfrak{t}) \cap \pi_{\mathsf{REL}}(\mathfrak{n})| \\ & + |\pi_{\mathsf{ATT}}(\mathfrak{t}) \cap \pi_{\mathsf{VALUE}}(\mathfrak{n})| \\ & + |\pi_{\mathsf{VALUE}}(\mathfrak{t}) \cap \pi_{\mathsf{REL}}(\mathfrak{n})| \\ & + |\pi_{\mathsf{VALUE}}(\mathfrak{t}) \cap \pi_{\mathsf{ATT}}(\mathfrak{n})|. \end{aligned}$$

Heuristic h_3 takes the maximum of h_1 and h_2 on $\mathfrak{n}$:

$$h_3(\mathfrak{n}) = max\{h_1(\mathfrak{n}), h_2(\mathfrak{n})\}.$$

Databases as Strings: The Levenshtein Heuristic. Viewing a database as a *string* leads to another heuristic. Suppose $\mathfrak{x}$ is a database in TNF with m tuples

$$\langle t_1, r_1, a_1, v_1\rangle, \ldots, \langle t_n, r_m, a_m, v_m\rangle.$$

For each such tuple, let $s_i = r_i \star a_i \star v_i$, where $\star$ is string concatenation. Define $\texttt{string}(\mathfrak{x})$ to be the string $s_1 \star \cdots \star s_m$, where $s_1, \ldots, s_m$ is a lexicographic ordering of the m strings s_i, potentially with repetitions.

Example 11. Recall the TNF of database `FlightsC` from Example 10:

$$\langle \texttt{t1 AirEast Route ATL29} \rangle, \langle \texttt{t1 AirEast BaseCost 100} \rangle,$$
$$\ldots, \langle \texttt{t4 JetWest BaseCost 220} \rangle, \langle \texttt{t4 JetWest TotalCost 236} \rangle.$$

Transforming each tuple into a string and then sorting these strings, we have

$$\texttt{string(FlightsC)} =$$
$$\texttt{AirEastBaseCost100AirEastRouteATL29} \cdots \texttt{JetWestTotalCost236}.$$

□

The *Levenshtein distance* between string w and string v, $L(w, v)$, is defined as the least number of single character insertions, deletions, and substitutions required to transform w into v [49]. Using this metric, we define the following *normalized Levenshtein heuristic*:

$$h_L(\mathfrak{n}) = round\left(k \frac{L(\texttt{string}(\mathfrak{n}), \texttt{string}(\mathfrak{t}))}{max\{|\texttt{string}(\mathfrak{n})|, |\texttt{string}(\mathfrak{t})|\}} \right)$$

where $|w|$ is the length of string w, $k \geqslant 1$ is a scaling constant (scaling the interval $[0, 1]$ to $[0, k]$), and $round(y)$ is the integer closest to y.

Databases as Term Vectors: Euclidean Distance. Another perspective on a database is to view it as a document vector over a set of terms [5]. Let $A = \{a_1, \ldots, a_n\}$ be the set of tokens occurring in the source and target critical instances (including attribute and relation names), and let

$$\mathbb{T} = \{\langle a_1, a_1, a_1 \rangle, \ldots, \langle a_n, a_n\, a_n \rangle\}$$

be the set of all n^3 triples over the tokens in A. Given a search database $\mathfrak{x}$ in TNF with tuples $\langle t_1, r_1, a_1, v_1 \rangle, \ldots, \langle t_\ell, r_m, a_m, v_m \rangle$, define $\bar{\mathfrak{x}}$ to be the n^3-vector $\langle \mathfrak{x}_1, \ldots, \mathfrak{x}_{n^3} \rangle$ where $\mathfrak{x}_i$ equals the number of occurrences of the ith triple of $\mathbb{T}$ in the list

$$\langle r_1, a_1, v_1 \rangle, \ldots, \langle r_m, a_m, v_m \rangle.$$

This term vector view on databases leads to several natural search heuristics. The standard Euclidean distance in term vector space from state $\mathfrak{n}$ to target state $\mathfrak{t}$ gives us a *Euclidean heuristic* measure:

$$h_E(\mathfrak{n}) = round\left(\sqrt{\sum_{i=1}^{n^3} (\mathfrak{n}_i - \mathfrak{t}_i)^2} \right)$$

where $\mathfrak{x}_i$ is the ith element of the database vector $\bar{\mathfrak{x}}$.

Normalizing the vectors for state $\mathfrak{n}$ and target $\mathfrak{t}$ gives a *normalized Euclidean heuristic* for the distance between $\mathfrak{n}$ and $\mathfrak{t}$:

$$h_{|E|}(\mathfrak{n}) = round\left(k \sqrt{\sum_{i=1}^{n^3} \left[\frac{\mathfrak{n}_i}{|\bar{\mathfrak{n}}|} - \frac{\mathfrak{t}_i}{|\bar{\mathfrak{t}}|} \right]^2} \right)$$

where $k \geqslant 1$ is a scaling constant and $|\bar{\mathfrak{x}}| = \sqrt{\sum_{i=1}^{n^3} \mathfrak{x}_i^2}$, as usual.

Databases as Term Vectors: Cosine Similarity. Viewing databases as vectors, we can also define a *cosine similarity heuristic* measure, with scaling constant $k \geqslant 1$:

$$h_{\cos}(\mathfrak{n}) = round\left(k\left[1 - \frac{\sum_{i=1}^{n^3} \mathfrak{n}_i \mathfrak{t}_i}{|\bar{\mathfrak{n}}||\bar{\mathfrak{t}}|}\right]\right)$$

Cosine similarity measures the cosine of the angle between two vectors in the database vector space. If $\mathfrak{n}$ is very similar to the target $\mathfrak{t}$ in this space, then $h_{\cos}$ returns a low estimate of the distance between them.

3.5 Supporting Complex Semantic Mappings

The mapping operators in the language $\mathscr{L}$ (Table 1) accommodate dynamic data-metadata structural transformations in addition to simple one-to-one schema matchings. However, as discussed in Section 2.3, many mappings involve complex semantic transformations [14,34,62,64]. As examples of such mappings, consider several basic complex mappings for bridging semantic differences between two tables.

Example 12. A semantic mapping f_1 from airline names to airline ID numbers:

Carrier		CID
AirEast	$\xmapsto{f_1}$	123
JetWest		456

A complex function f_2 which returns the concatenation of passenger first and last names:

Last	First		Passenger
Smith	John	$\xmapsto{f_2}$	John Smith
Doe	Jane		Jane Doe

The complex function f_3 between `FlightsB` and `FlightsC` which maps `AgentFee` and `Cost` to `TotalCost`:

CID	Route	Cost	AgentFee		CID	Route	TotalCost
123	ATL29	100	15		123	ATL29	115
456	ATL29	200	16	$\xmapsto{f_3}$	456	ATL29	216
123	ORD17	110	15		123	ORD17	125
456	ORD17	220	16		456	ORD17	236

□

Other examples include functions such as date format, weight, and international financial conversions, and semantic functions such as the mapping from employee name to social security number (which can not be generalized from examples), and so on.

Support for Semantic Mapping Expressions. Any complex semantic function is unique to a particular information sharing scenario. Incorporating such functions in a non-ad hoc manner is essential for any general data mapping solution. Although there has been research on discovering specific complex semantic functions [12,34], no general approach has been proposed which accommodates these functions in larger mapping expressions.

Tupelo supports discovery of mapping expressions with such complex semantic mappings in a straight-forward manner without introducing any specialized domain knowledge. We can cleanly accommodate these mappings in the system by extending $\mathscr{L}$ with a new operator λ which is parameterized by a complex function f and its input-output signature:

$$\lambda^{\mathtt{B}}_{f,\bar{\mathtt{A}}}(R).$$

Example 13. As an illustration of the operator, the mapping expression to apply function f_3 in Example 12 to the values in the `Cost` and `AgentFee` attributes, placing the output in attribute `TotalCost`:

$$\lambda^{\mathtt{TotalCost}}_{f_3,\mathtt{Cost},\ \mathtt{AgentFee}}(\mathtt{FlightsB}).$$

This is precisely the semantic transformation used in Example 7. □

The semantics of λ is as follows: for each tuple t in relation R, apply the mapping f to the values of t on attributes $\bar{\mathtt{A}} = \langle \mathtt{A}_1, \ldots, \mathtt{A}_n \rangle$ and place the output in attribute $\mathtt{B}$. The operator is well defined for any tuple t of appropriate schema (i.e., appropriate type), and is the identity mapping on t otherwise. Note that this semantics is independent of the actual mechanics of the function f. Function symbols are assumed to come from a countably infinite set $\mathscr{F} = \{f_i\}_{i=0}^{i=\infty}$.

Discovery of Semantic Mapping Expressions. Tupelo generates data mapping expressions in $\mathscr{L}$. Extending $\mathscr{L}$ with the λ operator allows for the discovery of mapping expressions with arbitrary complex semantic mappings. Given critical input/output instances and indications of complex semantic correspondences f between attributes $\bar{\mathtt{A}}$ in the source and attribute $\mathtt{B}$ in the target, the search is extended to generate appropriate mapping expressions which also include the λ operator (Figure 8).

For the purpose of searching for mapping expressions, λ expressions are treated just like any of the other operators. During search all that needs to be checked is that the applications of functions are well-typed. The system does not need any special semantic knowledge about the symbols in $\mathscr{F}$; they are treated simply as "black boxes" during search. The actual "meaning" of a function f, maintained perhaps as a stored procedure, is retrieved during the execution of the mapping expression on a particular database instance. Apart from what can be captured in search heuristics, this is probably the best that can be hoped for in general semantic integration. That is, all data semantics from some external sources of domain knowledge must be either encapsulated in the functions f or somehow introduced into the search mechanism, for example via search heuristics.

This highlights a clear separation between semantic functions which interpret the symbols in the database, such as during the application of functions in $\mathscr{F}$, and syntactic, structural transformations, such as those supported by generic languages like $\mathscr{L}$. This separation also extends to a separation of labor in data mapping discovery, as indicated in our system design framework of Section 2.3: discovering particular complex semantic functions and generating executable data mapping expressions are treated as two separate issues in Tupelo.

Discovering complex semantic functions is a difficult research challenge. Some recent efforts have been successful in automating the discovery of restricted classes of complex functions [12,34]. There has also been some initial research on optimization of mapping expressions which contain executable semantic functions [10].

Focusing on the discovery of data mapping expressions, Tupelo assumes that the necessary complex functions between the source and target schemas have been discovered and that these correspondences are articulated on the critical instance inputs to the system. These correspondences can be easily indicated by a user via a visual interface, such as those discussed in Section 3.2. Internally, complex semantic maps are just encoded as strings in the VALUE column of the TNF relation. This string indicates the input/output type of the function, the function name, and the example function values articulated in the input critical instance.

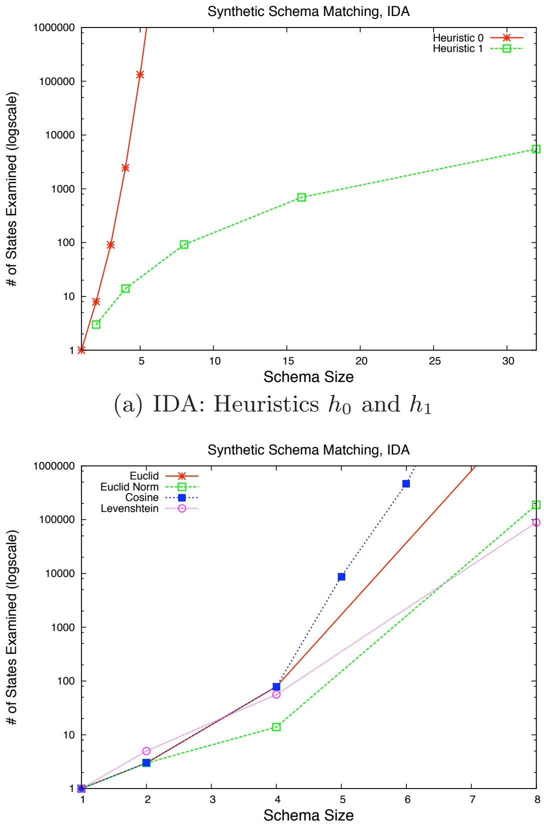
3.6 Empirical Evaluation

The Tupelo system has been fully implemented in Scheme. In this section we discuss extensive experimental evaluations of the system on a variety of synthetic and real world data sets. Our aim in these experiments was to explore the interplay of the IDA and RBFS algorithms with the seven heuristics described in Section 3.4. We found that overall RBFS had better performance than IDA. We also found that heuristics h_1, h_3, normalized Euclidean, and Cosine Similarity were the best performers on the test data sets.

Experimental Setup. All evaluations were performed on a Pentium 4 (2.8 GHz) with 1.0 GB main memory running Gentoo Linux (kernel 2.6.11-gentoo-r9) and Chez Scheme (v6.9c). In all experiments, the performance measure is the number of states examined during search. We also included the performance of heuristic h_0 for comparison with the other heuristics. This heuristic is constant on all values ($\forall \mathfrak{n}, h_0(\mathfrak{n}) = 0$) and hence induces brute-force blind search (comparable to breadth-first search for both IDA and RBFS [45]). Through extensive empirical evaluation of the heuristics and search algorithms on the data sets described below, we found that the following values for the heuristic scaling constants k give overall optimal performance:

	Norm. Euclidean	Cosine Sim.	Levenshtein
IDA	$k = 7$	$k = 5$	$k = 11$
RBFS	$k = 20$	$k = 24$	$k = 15$

These constant k values were used in all experiments presented below.



(a) IDA: Heuristics h_0 and h_1

(b) IDA: Term-vector and string heuristics

Fig. 9. Experiment 1: Number of states examined using IDA for schema matching on synthetic schemas

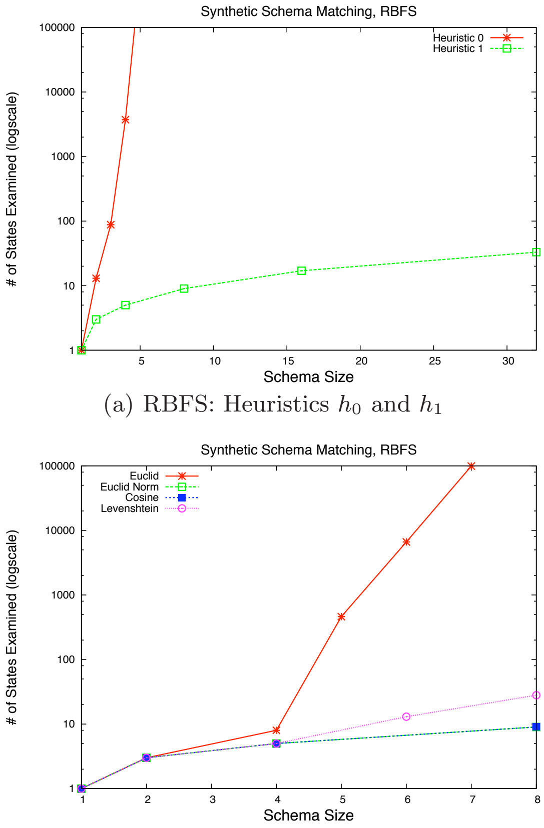
Experiment 1: Mapping on Synthetic Data. In the first experiment, we measured the performance of IDA and RBFS using all seven heuristics on a simple schema matching task.

Data Set. Pairs of schemas with $n = 2, \ldots, 32$ attributes were synthetically generated and populated with one tuple each illustrating correspondences between each schema:

$$\left\langle \frac{\texttt{A1}}{\texttt{a1}}, \frac{\texttt{B1}}{\texttt{a1}} \right\rangle \left\langle \frac{\texttt{A1 A2}}{\texttt{a1 a2}}, \frac{\texttt{B1 B2}}{\texttt{a1 a2}} \right\rangle \cdots \left\langle \frac{\texttt{A1} \cdots \texttt{A32}}{\texttt{a1} \cdots \texttt{a32}}, \frac{\texttt{B1} \cdots \texttt{B32}}{\texttt{a1} \cdots \texttt{a32}} \right\rangle$$

Each algorithm/heuristic combination was evaluated on generating the correct matchings between the schemas in each pair (i.e., $\texttt{A1}\leftrightarrow\texttt{B1}$, $\texttt{A2}\leftrightarrow\texttt{B2}$, etc.).

Results. The performance of IDA on this data set is presented in Figure 9, and the performance of RBFS is presented in Figure 10. Heuristic h_2 performed



(a) RBFS: Heuristics h_0 and h_1

(b) RBFS: Term-vector and string heuristics

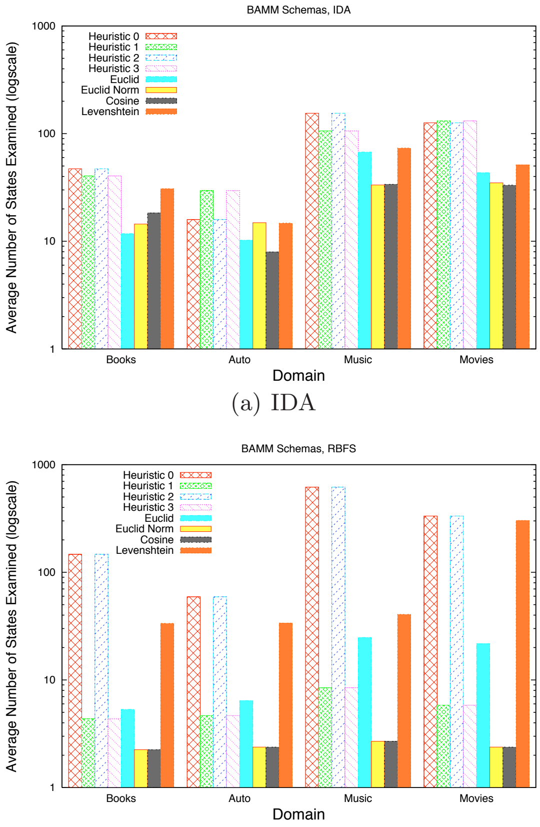
Fig. 10. Experiment 1: Number of states examined using RBFS for schema matching on synthetic schemas

identically to h_0, and heuristic h_3's performance was identical to h_1. Hence they are omitted in Figures 9(a) & 10(a). RBFS had performance superior to IDA on these schemas, with the h_1, Levenshtein, normalized Euclidean, and Cosine Similarity heuristics having best performance.

Experiment 2: Mapping on the Deep Web. In the second experiment we measured the performance of IDA and RBFS using all seven heuristics on a set of over 200 real-world query schemas extracted from deep web data sources.

Data Set. The Books, Automobiles, Music, and Movies (BAMM) data set from the UIUC Web Integration Repository[13] contains 55, 55, 49, and 52 schemas from deep web query interfaces in the Books, Automobiles, Music, and Movies domains, respectively. The schemas each have between 1 and 8 attributes. In this

[13] http://metaquerier.cs.uiuc.edu/repository



(a) IDA

(b) RBFS

Fig. 11. Experiment 2: Average number of states examined for mapping discovery in the four BAMM domains

experiment, we populated the schemas of each domain with critical instances. We then measured the average cost of mapping from a fixed schema in each domain to each of the other schemas in that domain.

Results. The average performance of IDA on each of the BAMM domains is presented in Figure 11(a). Average RBFS performance on each of the BAMM domains is given in Figure 11(b). The average performance of both algorithms across all BAMM domains is given in Figure 12. We found that RBFS typically examined fewer states on these domains than did IDA. Overall, we also found that the Cosine Similarity and normalized Euclidean heuristics had the best performance.

Experiment 3: Complex Semantic Mapping. In the third experiment we evaluated the performance of Tupelo on discovering complex semantic mapping expressions for real-world data sets in the real estate and business inventory domains.

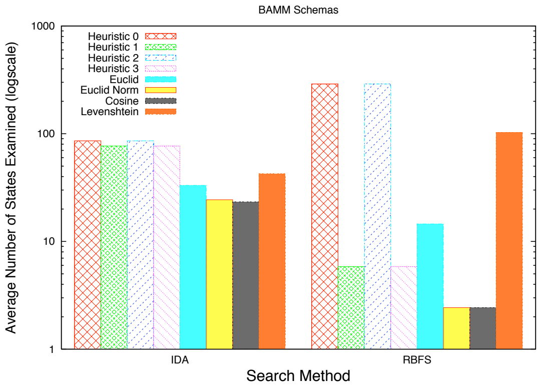


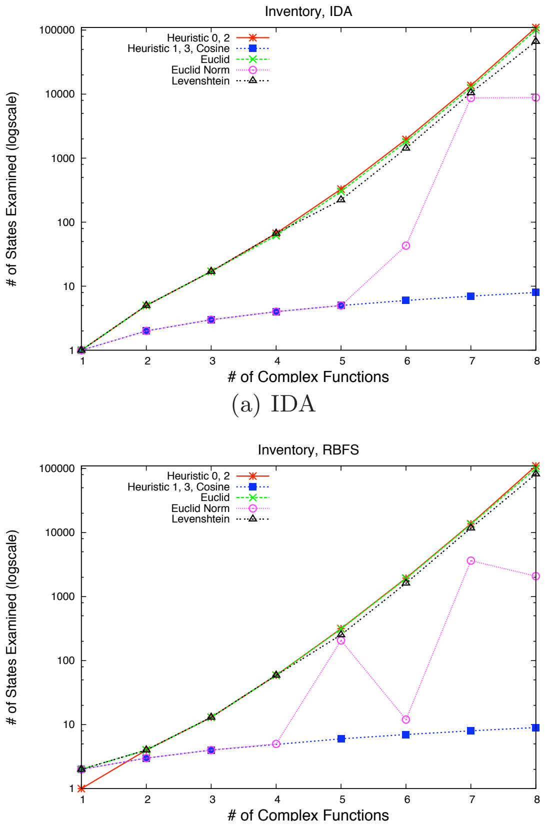
Fig. 12. Experiment 2: Average number of states examined by IDA and RBFS for mapping discovery across all BAMM domains

Data Set. We measured performance of complex semantic mapping with the schemas for the Inventory and Real Estate II data sets from the Illinois Semantic Integration Archive.[14] In the Inventory domain there are 10 complex semantic mappings between the source and target schemas, and in the Real Estate II domain there are 12. We populated each source-target schema pair with critical instances built from the provided datasets.

Results. The performance on both domains was essentially the same, so we present the results for the Inventory schemas. The number of states examined for mapping discovery in this domain for increasing numbers of complex semantic functions is given in Figure 13(a) for IDA and in Figure 13(b) for RBFS. On this data, we found that RBFS and IDA had similar performance. For the heuristics, the best performance was obtained by the h_1, h_3 and cosine similarity heuristics.

Discussion of Empirical Results. The goal of the experiments discussed in this section was to measure the performance of Tupelo on a wide variety of schemas. We found that Tupelo was effective for discovering mapping expressions in each of these domains, even with the simple heuristic search algorithms IDA and RBFS. It is clear from these experiments that RBFS is in general a more effective search algorithm than IDA. Although we found that heuristic h_1 exhibited consistently good performance, it is also clear that there was no perfect all-purpose search heuristic. Tupelo has also been validated and shown effective for examples involving the data-metadata restructurings illustrated in Figure 1 [19]. It was found in that domain that no particular heuristic had consistently superior performance. We can conclude from these observations that work still needs to be done on developing more sophisticated search heuristics.

[14] http://anhai.cs.uiuc.edu/archive/



(a) IDA

(b) RBFS

Fig. 13. Experiment 3: Number of states for complex semantic mapping discovery in the Inventory domain

3.7 Related Work

The problem of overcoming structural and semantic heterogeneity has a long history in the database [14] and AI [62] research communities. In Section 2 we have already situated Tupelo in the general research landscape of the data mapping problem. We now briefly highlight related research not discussed elsewhere in the paper.

Schema Matching. A wide variety of existing systems have leveraged AI and machine learning techniques for solving different aspects of schema matching and mapping. These include neural networks, Bayesian learning, and genetic programming approaches [13,51,60,66]. The Tupelo view on data mapping as search complements this body of research. As we have discussed, $\mathscr{L}$ includes (direct) schema matching as a special case and hence Tupelo introduces a novel approach to this research problem.

Data-Metadata Transformations. Few data mapping systems have considered the data-metadata structural transformations used in the Tupelo mapping language $\mathscr{L}$. Systems that have considered some aspects of these transformations include [7,12,59,81]. The systems closest to Tupelo in terms of metadata transformations are the recent work on contextual schema matching by Bohannon et al. [7] and composite approach to schema mapping by Xu and Embley [81], wherein the schema matching paradigm is extended to structural transformations. This work, however, only considers limited subsets of the $\mathscr{L}$ structural transformations. Furthermore, Tupelo complements these systems with the novel perspective of mapping discovery as search.

Example-Driven Data Mapping. The notion of example-based data mapping is an ancient idea, by some accounts dating back at least to the 4th century [63]. Recent work most closely related to the example driven approach of Tupelo include [50,63,66]. The results of this Section can be viewed as contributions towards extending this line of research, including a richer mapping language and new perspectives on mapping discovery.

Executable Mapping Expressions. Most schema matching systems do not address the issue of generating executable mapping expressions, which is in general considered to be an open hard problem [56]. Several notable systems that do generate such expressions include [4,25,31,32,57,59,66]. Our contributions to this research area include extending mapping expressions to include semantic transformations and introducing the Rosetta Stone Principle for expression discovery.

The Data Exchange Problem. A problem closely related to the data mapping problem is the *data exchange problem* [44], proposed to formalize aspects of the Clio schema mapping system developed at IBM [31,59]. This framework has been recently extended to consider the case of data-metadata transformations [35]. Briefly, the data exchange problem is as follows: given a source schema S, target schema T, source instance I, and a set $\Sigma_{S,T}$ of source-to-target dependencies in some logical formalism, find a target instance J that satisfies $\Sigma_{S,T}$ [44]. Fagin et al. have characterized solutions to the data exchange problem and have explored query answering in data exchange settings [44]. A limitation of these results is a restriction of the logical formalism for expressing $\Sigma_{S,T}$ to fragments of first order logic which do not always adequately express naturally occurring data mappings. Furthermore, in data exchange it is assumed (1) that these dependencies are given as input and (2) the target schema T is fixed. In the data *mapping* problem we are concerned precisely with discovering meaningful source to target constraints, given S, T, and perhaps (I, J) as input where the target schema T is potentially dynamic, as we saw in the mapping from `FlightsB` to `FlightsA` (Figure 1), which creates as many new route attributes in `FlightsA` as there are `Route` values in `FlightsB`.

In summary, Tupelo complements and extends the research in each of these areas by (1) attacking the data mapping problem as a basic search problem in a state space and by (2) addressing a broader class of mapping expressions including data-metadata transformations and complex semantics functions. We reiterate that, to the best of our knowledge, Tupelo is the first system for data mapping to take an end-to-end modular perspective on the problem and generate such a broad class of database transformations.

Remarks. In this Section we presented and illustrated the effectiveness of the Tupelo system for discovering data mapping expressions between relational data sources. Novel aspects of the system include (1) example-driven generation of mapping expressions which include data-metadata structural transformations and complex semantic mappings and (2) viewing the data mapping problem as fundamentally a search problem in a well defined search space. Mapping discovery is performed in Tupelo using only the syntax and structure of the input examples without recourse to any domain-specific semantic knowledge. The implementation of Tupelo was described and the viability of the approach illustrated on a variety of synthetic and real world schemas. We concluded the Section with an overview of related research results.

4 Concluding Remarks

In this paper, we have studied the long-standing problem of mapping discovery for data sharing and coordination between autonomous data sources. Our aim was the development of a robust perspective on the generic design space of data mapping solutions. In our investigations, we strove towards a deeper understanding of both what data mappings are and how to effectively go about discovering them. The contributions of this paper advance the state of the art of both theoretical and engineering aspects of the data mapping problem, and provide foundations for further progress on both of these fronts. We close by indicating some of the research directions supported by these foundations.

Abstract Formalisms for Data Mapping. We have taken a "set-theoretic" perspective in our development of DMP which lends itself to direct specification depending on problem context (e.g., our definition of RelationalDMP for relational sources in Section 2.3). There are several other high-level frameworks, however, which take a "categorical" perspective on data objects and their mappings. Examples of such abstract frameworks include information-flow theory [38] and institutions [27]. Although these formalisms were not proposed with an eye towards mapping discovery for structured data sources, it would be quite interesting to investigate the connections and disconnections between these perspectives and that of Section 2.

The Tupelo *Data Mapping System.* There are several promising avenues for future work on Tupelo (Section 3). As is evident from the empirical evaluation presented in Section 3.6, further research remains on developing more sophisticated search heuristics. The Levenshtein, Euclidean, and Cosine Similarity based search heuristics mostly focus on the content of database states. Successful heuristics must measure both content and structure. Is there a good multi-purpose search heuristic? Recently, Gillis and Van den Bussche have studied search heuristics for discovering queries involving negation [25]. Also, we have only applied straightforward approaches to search with the IDA and RBFS algorithms. Further investigation of search techniques developed in the AI literature is warranted. Finally, the perspective of data mapping as search is not limited to relational data sources. In particular, the architecture of the Tupelo system can be applied to the generation of mapping expressions in other mapping languages and for other data models. Based on the viability of the system for relational data sources, this is a promising avenue for future research.

Acknowledgments. We thank the reviewers, the members of the Indiana University database group, and Jan Van den Bussche for their support and many helpful comments.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Agrawal, R., Somani, A., Xu, Y.: Storage and Querying of E-Commerce Data. In: VLDB, Rome, Italy, pp. 149–158 (2001)
3. Batini, C., Lenzerini, M., Navathe, S.B.: A Comparative Analysis of Methodologies for Database Schema Integration. ACM Comput. Surv. 18(4), 323–364 (1986)
4. Bernstein, P.A., Melnik, S., Mork, P.: Interactive Schema Translation with Instance-Level Mappings. In: VLDB, Trondheim, Norway, pp. 1283–1286 (2005)
5. Berry, M.W., Drmač, Z., Jessup, E.R.: Matrices, Vector Spaces, and Information Retrieval. SIAM Review 41(2), 335–362 (1999)
6. Bilke, A., Naumann, F.: Schema Matching using Duplicates. In: IEEE ICDE, Tokyo, Japan, pp. 69–80 (2005)
7. Bohannon, P., Elnahrawy, E., Fan, W., Flaster, M.: Putting Context into Schema Matching. In: VLDB, Seoul, Korea, pp. 307–318 (2006)
8. Bossung, S., Stoeckle, H., Grundy, J.C., Amor, R., Hosking, J.G.: Automated Data Mapping Specification via Schema Heuristics and User Interaction. In: IEEE ASE, Linz, Austria, pp. 208–217 (2004)
9. Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Logical Foundations of Peer-To-Peer Data Integration. In: ACM PODS, Paris, France, pp. 241–251 (2004)
10. Carreira, P., Galhardas, H.: Execution of Data Mappers. In: ACM SIGMOD Workshop IQIS, Paris, France, pp. 2–9 (2004)
11. Dalvi, N.N., Suciu, D.: Management of Probabilistic Data: Foundations and Challenges. In: PODS, Beijing, pp. 1–12 (2007)
12. Dhamankar, R., Lee, Y., Doan, A., Halevy, A.Y., Domingos, P.: iMAP: Discovering Complex Mappings between Database Schemas. In: ACM SIGMOD, Paris, France, pp. 383–394 (2004)
13. Doan, A., Domingos, P., Halevy, A.: Learning to Match the Schemas of Databases: A Multistrategy Approach. Machine Learning 50(3), 279–301 (2003)
14. Doan, A., Noy, N.F., Halevy, A.Y.: Special Issue on Semantic Integration. SIGMOD Record 33(4) (2004)
15. Eco, U.: The Search for the Perfect Language. Blackwell, Oxford (1995)
16. Euzenat, J., et al.: State of the Art on Ontology Alignment. Technical Report D2.2.3, IST Knowledge Web NoE (2004)
17. Feng, Y., Goldstone, R.L., Menkov, V.: A Graph Matching Algorithm and its Application to Conceptual System Translation. Int. J. AI Tools 14(1-2), 77–100 (2005)

18. Fletcher, G.H.L., Gyssens, M., Paredaens, J., Van Gucht, D.: On the Expressive Power of the Relational Algebra on Finite Sets of Relation Pairs. IEEE Trans. Knowl. Data Eng. 21(6), 939–942 (2009)
19. Fletcher, G.H.L., Wyss, C.M.: Mapping Between Data Sources on the Web. In: IEEE WIRI, Tokyo, Japan, pp. 173–178 (2005)
20. Fletcher, G.H.L., Wyss, C.M.: Data Mapping as Search. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 95–111. Springer, Heidelberg (2006)
21. Fletcher, G.H.L., Wyss, C.M., Robertson, E.L., Van Gucht, D.: A Calculus for Data Mapping. ENTCS 150(2), 37–54 (2006)
22. Gal, A.: On the Cardinality of Schema Matching. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 947–956. Springer, Heidelberg (2005)
23. Gal, A.: Why is Schema Matching Tough and What Can We Do About It?. SIGMOD Record 35(4), 2–5 (2006)
24. Garcia-Molina, H.: Web Information Management: Past, Present, Future. In: ACM WSDM, Palo Alto, CA (2008)
25. Gillis, J., Van den Bussche, J.: Induction of relational algebra expressions. In: ILP, Leuven (2009)
26. Giunchiglia, F., Shvaiko, P.: Semantic Matching. Knowledge Eng. Review 18(3), 265–280 (2003)
27. Goguen, J.A.: Information Integration in Institutions. In: Moss, L. (ed.) Jon Barwise Memorial Volume. Indiana University Press (2006)
28. Gottlob, G., Koch, C., Baumgartner, R., Herzog, M., Flesca, S.: The Lixto Data Extraction Project - Back and Forth between Theory and Practice. In: ACM PODS, Paris, France, pp. 1–12 (2004)
29. Grahne, G., Kiricenko, V.: Towards an Algebraic Theory of Information Integration. Information and Computation 194(2), 79–100 (2004)
30. Grundy, J.C., Hosking, J.G., Amor, R., Mugridge, W.B., Li, Y.: Domain-Specific Visual Languages for Specifying and Generating Data Mapping Systems. J. Vis. Lang. Comput. 15(3-4), 243–263 (2004)
31. Haas, L.M., Hernández, M.A., Ho, H., Popa, L., Roth, M.: Clio Grows Up: From Research Prototype to Industrial Tool. In: ACM SIGMOD, Baltimore, MD, pp. 805–810 (2005)
32. Habegger, B.: Mapping a Database into an Ontology: a Relational Learning Approach. In: IEEE ICDE, Istanbul, pp. 1443–1447 (2007)
33. Harris, R.: The Language Connection: Philosophy and Linguistics. Thoemmes Press, Bristol (1997)
34. He, B., Chang, K.C.-C., Han, J.: Discovering Complex Matchings Across Web Query Interfaces: a Correlation Mining Approach. In: ACM KDD, Seattle, WA, pp. 148–157 (2004)
35. Hernández, M.A., Papotti, P., Tan, W.-C.: Data Exchange with Data-Metadata Translations. In: VLDB, Auckland, New Zealand (2008)
36. Hull, R.: Managing Semantic Heterogeneity in Databases: a Theoretical Perspective. In: ACM PODS, Tucson, AZ, pp. 51–61 (1997)
37. Jain, M.K., Mendhekar, A., Van Gucht, D.: A Uniform Data Model for Relational Data and Meta-Data Query Processing. In: COMAD, Pune, India (1995)
38. Kalfoglou, Y., Schorlemmer, M.: Ontology Mapping: the State of the Art. Knowledge Eng. Review 18(1), 1–31 (2003)

39. Kashyap, V., Sheth, A.: Semantic and Schematic Similarities Between Database Objects: A Context-Based Approach. VLDB J. 5(4), 276–304 (1996)
40. Kedad, Z., Xue, X.: Mapping Discovery for XML Data Integration. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 166–182. Springer, Heidelberg (2005)
41. Kementsietsidis, A., Arenas, M., Miller, R.J.: Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In: ACM SIGMOD, San Diego, CA, pp. 325–336 (2003)
42. Kent, W.: The Unsolvable Identity Problem. In: Extreme Markup Languages, Montréal, Quebec, Canada (2003)
43. Kim, W., Seo, J.: Classifying Schematic and Data Heterogeneity in Multidatabase Systems. IEEE Computer 24(12), 12–18 (1991)
44. Kolaitis, P.G.: Schema Mappings, Data Exchange, and Metadata Management. In: ACM PODS, Baltimore, MD, pp. 61–75 (2005)
45. Korf, R.E.: Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. Artif. Intell. 27(1), 97–109 (1985)
46. Korf, R.E.: Linear-Space Best-First Search. Artif. Intell. 62(1), 41–78 (1993)
47. Krishnamurthy, R., Litwin, W., Kent, W.: Language Features for Interoperability of Databases with Schematic Discrepancies. In: ACM SIGMOD, Denver, CO, pp. 40–49 (1991)
48. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: ACM PODS, Madison, WI, pp. 233–246 (2002)
49. Levenshtein, V.I.: Dvoichnye Kody s Ispravleniem Vypadenii, Vstavok i Zameshchenii Simvolov. Doklady Akademii Nauk SSSR 163(4), 845–848 (1965)
50. Levy, A.Y., Ordille, J.J.: An Experiment in Integrating Internet Information Sources. In: AAAI Fall Symposium on AI Applications in Knowledge Navigation and Retrieval, Cambridge, MA, pp. 92–96 (1995)
51. Li, W.-S., Clifton, C.: SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Networks. Data & Knowl. Eng. 33(1), 49–84 (2000)
52. Litwin, W.: Bridging a Great Divide: Past, Present, and Future in Multidatabase Interoperability. In: InterDB, Namur, Belgium (2005)
53. Litwin, W., Ketabchi, M.A., Krishnamurthy, R.: First Order Normal Form for Relational Databases and Multidatabases. SIGMOD Record 20(4), 74–76 (1991)
54. Litwin, W., Mark, L., Roussopoulos, N.: Interoperability of Multiple Autonomous Databases. ACM Comput. Surv. 22(3), 267–293 (1990)
55. Matuszek, C., Cabral, J., Witbrockand, M., DeOliveira, J.: An Introduction to the Syntax and Content of Cyc. In: Baral, C. (ed.) Technical Report SS-06-05, pp. 44–49. AAAI Press, Menlo Park (2006)
56. Melnik, S.: Generic Model Management: Concepts and Algorithms. Springer, Berlin (2004)
57. Melnik, S., Bernstein, P.A., Halevy, A.Y., Rahm, E.: Supporting Executable Mappings in Model Management. In: ACM SIGMOD, Baltimore, MD, pp. 167–178 (2005)
58. Miller, R.J.: Using Schematically Heterogeneous Structures. In: ACM SIGMOD, Seattle, WA, pp. 189–200 (1998)
59. Miller, R.J., Haas, L.M., Hernández, M.A.: Schema Mapping as Query Discovery. In: VLDB, Cairo, Egypt, pp. 77–88 (2000)
60. Morishima, A., Kitagawa, H., Matsumoto, A.: A Machine Learning Approach to Rapid Development of XML Mapping Queries. In: IEEE ICDE, Boston, MA, pp. 276–287 (2004)

61. Nilsson, N.J.: Artificial Intelligence: A New Synthesis. Morgan Kaufmann, San Francisco (1998)
62. Noy, N.F., Doan, A., Halevy, A.Y.: Special Issue on Semantic Integration. AI Magazine 26(1) (2005)
63. Perkowitz, M., Doorenbos, R.B., Etzioni, O., Weld, D.S.: Learning to Understand Information on the Internet: An Example-Based Approach. J. Intell. Inf. Syst. 8(2), 133–153 (1997)
64. Rahm, E., Bernstein, P.A.: A Survey of Approaches to Automatic Schema Matching. VLDB J. 10(4), 334–350 (2001)
65. Raman, V., Hellerstein, J.M.: Potter's Wheel: An Interactive Data Cleaning System. In: VLDB, Roma, Italy, pp. 381–390 (2001)
66. Schmid, U., Waltermann, J.: Automatic Synthesis of XSL-Transformations from Example Documents. In: IASTED AIA, Innsbruck, Austria, pp. 252–257 (2004)
67. Sheth, A.P., Larson, J.A.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. ACM Comput. Surv. 22(3), 183–236 (1990)
68. Shu, N.C., Housel, B.C., Taylor, R.W., Ghosh, S.P., Lum, V.Y.: EXPRESS: a Data EXtraction, Processing, and Restructuring System. ACM Trans. Database Syst. 2(2), 134–174 (1977)
69. 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)
70. Stuckenschmidt, H., van Harmelen, F.: Information Sharing on the Semantic Web. Springer, Berlin (2005)
71. Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-based integration of information – a survey of existing approaches. In: IJCAI (2001)
72. Wang, G., Goguen, J.A., Nam, Y.-K., Lin, K.: Critical Points for Interactive Schema Matching. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (eds.) APWeb 2004. LNCS, vol. 3007, pp. 654–664. Springer, Heidelberg (2004)
73. Warren, R.H., Tompa, F.W.: Multi-Column Substring Matching for Database Schema Translation. In: VLDB, Seoul, Korea, pp. 331–342 (2006)
74. Wiederhold, G.: The Impossibility of Global Consistency. OMICS 7(1), 17–20 (2003)
75. Wiesman, F., Roos, N.: Domain Independent Learning of Ontology Mappings. In: AAMAS, New York, NY, pp. 846–853 (2004)
76. Winkler, W.E.: The State of Record Linkage and Current Research Problems. Technical Report RR99/04, U.S. Bureau of the Census, Statistical Research Division (1999)
77. Wyss, C.M., Robertson, E.L.: A Formal Characterization of PIVOT/UNPIVOT. In: ACM CIKM, Bremen, Germany, pp. 602–608 (2005)
78. Wyss, C.M., Robertson, E.L.: Relational Languages for Metadata Integration. ACM Trans. Database Syst. 30(2), 624–660 (2005)
79. Wyss, C.M., Van Gucht, D.: A Relational Algebra for Data/Metadata Integration in a Federated Database System. In: ACM CIKM, Atlanta, GA, USA, pp. 65–72 (2001)
80. Wyss, C.M., Wyss, F.I.: Extending Relational Query Optimization to Dynamic Schemas for Information Integration in Multidatabases. In: ACM SIGMOD, Beijing (2007)
81. Xu, L., Embley, D.W.: A Composite Approach to Automating Direct and Indirect Schema Mappings. Information Systems 31(8), 697–732 (2006)