Merging Relational Views: A Minimization Approach

Xiang Li and Christoph Quix

Informatik 5 (Information Systems), RWTH Aachen University, 52056 Aachen, Germany
{lixiang,quix}@dbis.rwth-aachen.de

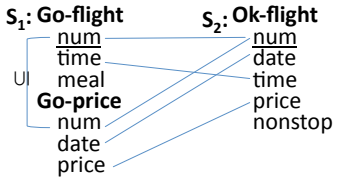
Abstract. Schema integration is the procedure to integrate several inter-related schemas to produce a unified schema, called the mediated schema. There are two major flavors of schema integration: data integration and view integration. The former deals with integrating multiple data sources to create a mediated query interface, while the latter aims at constructing a base schema, capable of supporting the source schemas as views. Our work builds upon previous approaches that address relational view integration using logical mapping constraints. Given a set of data dependencies over the source schemas as input, our approach produces a minimal information-preserving mediated schema with constraints, and it generates output mappings defining the source schemas as views. We extend previous approaches in several aspects. First, schema minimization is performed within a scope of Project-Join views that are information preserving and produce a smaller mediated schema than in existing work. Second, the input schema mapping language is expressive enough for not only query containment but also query equivalence. Third, source integrity constraints can be seamlessly incorporated into reasoning. Last but not least, we have evaluated our implementation over both real world data sets and a schema mapping benchmark.

1 Introduction

Integration multiple schemas can be valuable in many contexts. Building a data integration system requires integrating the local schemas to create a global schema for queries. In database design, a database schema can be obtained by integrating a set of desired user views [6, 21]. Bernstein and Melnik describe in [5] a scenario of schema evolution, in which an existing view is merged with another view representing newly incorporated information to construct an augmented view over an evolved database. In data warehousing, multiple data marts can be merged into a single materialized view to reduce maintenance costs.

Schema integration is the process of consolidating multiple related heterogeneous input schemas to produce a mediated schema. A typical schema integration procedure consists of three phases: schema matching, schema merging, and post-integration. The first phase identifies interschema relationships in the form of schema mappings. The second phase performs schema restructuring and transformation to construct a mediated schema. The last phase handles issues such as conflict resolution. This paper focus on the schema merging phase.

As clarified in [16], there are two semantically distinct flavors of schema merging, view integration [7, 6, 22, 14, 15, 3] and data integration [17, 8, 19, 20, 12]. From a



Keys are underlined and $\subseteq$ indicates foreign key constraints, while value correspondences across schemas are illustrated via solid lines. A mapping constraint between the two schemas can be expressed as $\pi_{num,date,time,price}(\text{Ok-flight}) = \pi_{num,date,time,price}(\text{Go-flight} \bowtie \text{Go-price})$.

Fig. 1. Running Example

model theoretic perspective, a main distinction is that the former flavor assumes the input schemas are views of a base schema and hence satisfy constraints on the instance level, while the latter need to integrate autonomous input schemas which do not necessarily conform to any instance level constraints. [16] also analyzed the different operational goals of the two types of schema merging. We study in this paper view integration, while our previous work[12] deals with data integration.

A possible result of a schema mapping discovery process, i.e., the first phase of schema integration, is described in Example 1 (adapted from [17]). Several challenges can be observed there. First, interschema relationships are captured by complex expressions involving multiple relations, and they are in the form of query equivalence instead of query containment. This requires a mapping language more expressive than that in data exchange [10]. Second, integrity constraints encoded in the source schemas provide significant information about the inner structure and hence they should not be ignored. Third, as the input has a non-ambiguous model theoretic semantics, the view integration process also needs to be backed by a rigorous instance level interpretation. Last but not least, in order to be executable, the output should include not only a mediated schema but also mappings defining input schemas as views and constraints over the mediated schema to restrict the instance scope of the mediated schema.

Example 1. Two travel agents *Go-flight* and *Ok-flight* have different schemas of flight information, as illustrated in Figure 1. The attribute *num* denotes flight numbers; *time* and *date* describe the departure time and date of a flight; *price* stands for the price of an air ticket; *meal* and *nonstop* are two boolean attributes indicating whether meals are offered on the flight and whether the flight is non-stop.

To address such challenges, we propose in this paper a novel approach to relational view integration. Our contributions include:

Formulation of Minimal Merge: The formal definition of a valid *Merge* as in [15, 3] requires the mediated schema to retain *all and only* the information of the source schemas. They do not consider minimizing schema size, and produce a mediated schema of the same size of the input. The formalism is extended in this paper to incorporate considerations on the size of the mediated schema, resulting in a characterization of *Minimal Merge*. We presented in [12] a merging algorithm that removed redundant columns from the source schemas in a data integration context. The approach described here considers, in addition, collapsing of relations, which leads to an even smaller schema.

Expressive Logical Mapping Language: Most of the early approaches (as surveyed in surveyed in [4]) to view integration focused more on the conceptual design of the

mediated schema and used conceptual alignment of entities as input mappings. Logical constraints were first used in [6, 7], but the joins are not allowed. However, schema mappings resulting from an automatic process, either mapping discovery or other mapping manipulation operations, are complex expressions such as in Fig. 1. Therefore, an expressive mapping language is crucial for a merging approach to be applicable in a model management workflow. In view of this, the input mapping language in our approach are specified in tuple-generating dependencies (tgds) and equality generating dependencies (egds), with the only restriction that the input mapping constraints admit a terminating chase [1]. Chase termination is in general undecidable, but there are sound syntactic conditions ensuring it (e.g., see [10]). With such an expressive language, we are able to make use of not only query containment constraints, but also query equivalence constraints and integrity constraints (ICs) shipped with the input schemas.

Algorithms for Constructing Minimal Merge: We present algorithms for merging relational schemas, including schema minimization under dependencies, mediated schema construction, output view definition mapping generation, and rewriting input constraints to the mediated schema.

Evaluation on Real World Data Sets: Last but not least, we have implemented and tested our framework on real world data sets and a workload generated from a schema mapping benchmark which summarizes several most common mapping scenarios in practice.

The rest of the paper is organized as follows: section 2 presents a formal definition of *Minimal Merge*; section 3 describes schema minimization under dependencies; section 4 details how to generate the mediated schema and the output mappings from a minimized Project-Join view; the evaluation is presented in section 5; related work is discussed in section 6, and conclusion and outlook are in section 7.

2 The View Integration Problem

2.1 Modeling View Integration

A *relational schema* is a set of relation symbols, each of which has a fixed arity. Let $\mathbf{S}$ be a relational schema, $Inst(\mathbf{S})$ be the set of all instances, a schema mapping $\mathcal{M}$ between two schemas $\mathbf{S}$ and $\mathbf{T}$ can be syntactically represented by a triple $(\mathbf{S}, \mathbf{T}, \Sigma)$, with Σ being a set of data dependencies over $\mathbf{S} \cup \mathbf{T}$. The instances of the mapping $Inst(\mathcal{M})$ is a subset of the cartesian product $Inst(\mathbf{S}) \times Inst(\mathbf{T})$. We also denote by $dom(\mathcal{M})$ and $range(\mathcal{M})$ the domain and range of a mapping $\mathcal{M}$.

In order to define the semantics of view integration, we introduce first the notion of *confluence* of two mappings, which is also adopted in [3, 15]. Let $\mathbf{S}$, $\mathbf{S}_1$ and $\mathbf{S}_2$ be schemas where $\mathbf{S}_1$ and $\mathbf{S}_2$ share no common relation symbol. $\mathcal{M}_1$ ($\mathcal{M}_2$) is a mapping from $\mathbf{S}$ to $\mathbf{S}_1$ ($\mathbf{S}_2$ respectively). Then the confluence of $\mathcal{M}_1$ and $\mathcal{M}_2$, denoted by $\mathcal{M}_1 \oplus \mathcal{M}_2$ is $\{(I, J \cup K) | (I, J) \in \mathcal{M}_1, (I, K) \in \mathcal{M}_2\}$. Intuitively, the confluence of two mappings maps each source instance to a concatenation of the two target instances under the original mappings. We now define the *merge* operator.

Definition 1. *Let $\mathcal{M}$ be a schema mapping between $\mathbf{S}_1$ and $\mathbf{S}_2$. $(\mathbf{S}, \mathcal{M}_1, \mathcal{M}_2)$ is a* merge *of $\mathcal{M}$ if: 1) $dom(\mathcal{M}_1) = dom(\mathcal{M}_2)$; 2) $\mathcal{M}_1 \oplus \mathcal{M}_2$ is an injective function; and 3) $range(\mathcal{M}_1 \oplus \mathcal{M}_2) = \{J \cup K | (J, K) \in Inst(\mathcal{M})\}$.*

The first condition requires that the two output mappings are views over the same scope of databases. Condition 2) states the output mappings are view definitions and injectiveness guarantees the mediated schema does not carry any extra information not contained in the sources. Condition 3) ensures that every unified instance in the range of the confluence respect the input mapping constraints, that is, a mutually consistent pair of views can be retrieved.

The definition of *merge* does not concern the mapping language. We introduce now tgds and egds. A *tuple generating dependency (tgd)* [1], is a constraint in the form of: $\forall \boldsymbol{X}[\exists \boldsymbol{Y}\phi(\boldsymbol{X},\boldsymbol{Y}) \rightarrow \exists \boldsymbol{Z}\psi(\boldsymbol{X},\boldsymbol{Z})]$, where ϕ and ψ are conjunctions of atoms and $\boldsymbol{X}$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$ are mutually disjoint sets of variables. It is *full*, if there are no existential variables on the right hand side, otherwise it is an *embedded* tgd. An *equality-generating dependency (egd)* [1] has the form $\forall \boldsymbol{X}[\phi(\boldsymbol{X}) \rightarrow (X_i = X_j)]$, where $\phi(\boldsymbol{X})$ is a conjunction of atoms, and X_i, X_j are variables in $\boldsymbol{X}$. When emphasizing the direction of a schema mapping between a designated pair of source and target schemas, we use the term *source-to-target* tgds (s-t tgds). A *closure* of a set of full s-t tgsd Σ, denoted by Σ^*, is the set of dependencies replacing each containment constraint in Σ by an equivalence constraint. Intuitively, closure *closes* the scope of target instances to be exactly the unique view instance computed from a given source instance.

For a schema $\mathbf{S}$, we denote by $\hat{\mathbf{S}}$ a replica schema with each relation renamed. We also denote by $\hat{R}$ the replica relational symbol of R, and by $\hat{\Gamma}$ the dependencies obtained through renaming relational symbols in Γ to their replicas. An identity mapping Id is the mapping $\{(I,J)|I \subseteq J \wedge I \in Inst(\mathbf{S}) \wedge J \in Inst(\hat{\mathbf{S}})\}$. In the theorem below, we denote by Id_1 (resp. Id_2) the mapping copying $\hat{\mathbf{S}}_1$ to $\mathbf{S}_1$ (resp. $\hat{\mathbf{S}}_2$ to $\mathbf{S}_2$). The theorem states that a union of the replicas of the input schemas together with an identity mapping is a valid *merge*.

Theorem 1 ([14]). *Let $\mathcal{M}$ be a mapping specified by data dependencies Γ in first order logic over $\mathbf{S}_1$ and $\mathbf{S}_2$, then $(\hat{\mathbf{S}}_1 \cup \hat{\mathbf{S}}_2, \hat{\Gamma} \cup Id_1^*, \hat{\Gamma} \cup Id_2^*)$ is a merge of $\mathcal{M}$.*

Example 2 (Example 1 cont.). Following Theorem 1, a valid merge could be $(\hat{\mathbf{S}}_1 \cup \hat{\mathbf{S}}_2, \hat{\Gamma} \cup Id_1, \hat{\Gamma} \cup Id_2)$. $\hat{\Gamma}$ contains two parts: the integrity constraints including all key constraints and inclusion dependencies over $\hat{\mathbf{S}}_1$ and $\hat{\mathbf{S}}_2$, and the rewritten mapping constraints $\pi_{num,date,time,price}(\hat{\text{Ok-flight}}) = \pi_{num,date,time,price}(\hat{\text{Go-flight}} \bowtie \hat{\text{Go-price}})$. The identity mapping Id_2 can be expressed by full s-t tgds $\hat{\text{Ok-flight}}(n,d,t,p,s) \rightarrow \text{Ok-flight}(n,d,t,p,s)$. Mapping Id_1 is similar.

The result in example 2 is not minimal as *time*, *date*, *price* are stored more than once.

Arenas et al. [3] propose a merging algorithm taking full s-t tgds as input and admitting denial constraints over the mediated schema.We show their merge algorithm[1] on an adapted version of Example 2 that confines the input mapping to full s-t tgds.

Example 3 (Example 2 cont.). The input is adapted by stating *num* is a key of Go-price, and replacing the interschema constraint by $\text{Ok-flight}(n,d,t,p,s) \rightarrow \text{Go-price}(n,d,p)$. A valid merge can be $(\hat{\mathbf{S}}_1 \cup \hat{\mathbf{S}}_2, \hat{\Gamma}' \cup \Sigma_1^*, \hat{\Gamma}' \cup Id_2^*)$, in which Id_2 is the same as in Example 2, and Σ_1 and $\hat{\Gamma}'$ are specified below. Σ_1 contains the full s-t tgds $\hat{\text{Go-flight}}(n,t,m) \rightarrow$

[1] Example 3 slightly extends the original algorithm in [3] , as source integrity constraints are not treated there.

Go-flight(n, t, m), Go-$\hat{\text{p}}$rice$(n, d, p) \rightarrow$ Go-price(n, d, p), and Ok-$\hat{\text{f}}$light$(n, d, t, p, s) \rightarrow$ Go-price(n, d, p). $\hat{\Gamma}'$ is a union of rewritten source ICs as in Example 2 and a denial constraint $\neg$(Ok-$\hat{\text{f}}$light$(n, d, t, p, s) \wedge$ Go-$\hat{\text{p}}$rice$(n, d, p))$, which ensures that the relation Go-$\hat{\text{p}}$rice does not store tuples which are already in Ok-$\hat{\text{f}}$light.

Example 3 is also not minimal, as storing *date* and *price* in Ok-$\hat{\text{f}}$light is unnecessary.

2.2 Minimality of Mediated Schema

The *merge* definition ensures that the mediated schema incorporates *all and only* the information in the input mapping system, but it does not restrict how the information is organized in the mediated schema. In this section, we present a merge requirement that aims at reducing redundant representations of the same piece of information on the mediated schema.

Given a schema with dependencies, we consider a set of schemas resulting from two types of transformations: 1) collapsing fragmented relations representing the same entity, and 2) projecting out unnecessary columns of a schema.

Definition 2. *A* bidirectional inclusion dependency (BIND) *is a mapping constraint* $\pi_{\mathbf{A}_1}(R_1) = \pi_{\mathbf{A}_2}(R_2)$*, where* R_1 *and* R_2 *are two relations, and* $\mathbf{A}_1$ *(*$\mathbf{A}_2$ *resp.) is a list of non-repeating attributes in* R_1 *(*R_2 *resp.).*

In Example 1, a BIND implied by the input mapping constraint and the integrity constraints is $\pi_{num,date,price}$(Go-price) $= \pi_{num,date,price}$(Ok-flight).

Existence of a BIND is a hint of fragmented entities. Joining the fragments over shared attributes in the BIND will be information preserving, if the join is lossless. However, a lossless join on a Key-ForeignKey relationship may lead to denormalization. In order not to degrade the quality of the mediated schema as a base schema, we confine the minimization scope to fragmented entities implied by BINDs via a key, i.e., the shared attributes are a super key for both relations.

A column in a schema is *redundant*, if removing it does not lose information. More formally, the mapping removing it (expressed in full s-t tgds) is tgd invertible [9]. That is, there exists a backward mapping in full tgds recovering the unprojected database. We refer interested readers to [12] for more details.

Definition 3. *A* merge *is* minimial *if there is neither a BIND via key nor a redundant column in the mediated schema.*

3 Schema Minimization under Data Dependencies

Assuming the input source schemas share no relation symbol, we refer to the union of them as *the source schema* (denoted by $\mathbf{S}$), and the union of interschema constraints and innerschema ICs as *input constraints* (denoted by Γ). Γ is assumed to admit a terminating chase. A *collapse-configuration* is a partial function mapping pairs of relations to a BIND via key that is implied by Γ. A *project-configuration* is a set of columns that are redundant under Γ. A *merge-configuration* is then a collapse-configuration plus a project-configuration. In this section we describe the schema minimization procedure, taking $\mathbf{S}$ and Γ as input and producing merge-configurations representing *minimal* Project-Join views.

3.1 Collapse-minimization using Maximal BINDs

A BIND is *maximal*, if there is no other BIND with the same relations and a superset of attribute pairs. We present in Algorithm 1 a procedure to find all maximal BINDs implied by Γ using the chase procedure [1] for reasoning over data dependencies. The algorithm starts by finding all inclusion dependencies (INDs) implied by Γ. For each relation R in the schema, a singleton tableau containing a single R-tuple with distinct variables is taken as a starting database. The singleton tableau is then chased against Γ. For each T-tuple in the chase result with a set of variables overlapping with the original R-tuple, an IND is recorded. In a second phase, all INDs are "joined" to produce BINDs. Finally, we prune those BINDs that are not maximal.

Input : A source schema $\mathbf{S}$, a set of dependencies Γ over $\mathbf{S}$ with terminating chase.
Output: A set of all maximal BINDs.

Initialize IND $\leftarrow \emptyset$
for *each relation* $R \in \mathbf{S}$ **do**
  Let I_R be a singleton R-tuple with all different variables
  Let I_R^Γ be the chase of I_R against Γ
  for *each relation* $T \in \mathbf{S}/\{R\}$ **do**
    if *there is a* T*-tuple* $t \in I_R^\Gamma$ *containing variables in* I_R **then**
      $L \leftarrow \emptyset$
      for *each variable* $x \in dom(I_R)$ **do**
        If x appears in position j in t and position i in the R-tuple of I_R, add (i, j) to L
      end
      add (R, T, L) to IND
    end
  end
end
BIND $\leftarrow \emptyset$
for *each pair of relations R and T* **do**
  for *each* $(R, T, L_{rt}) \in$ IND **do**
    for *each* $(T, R, L_{tr}) \in$ IND **do**
      $L \leftarrow \emptyset$
      For each $(i, j) \in L_{rt}$ and $(j, i) \in L_{tr}$, add (i, j) to L.
      If L is not empty, add (R, T, L) to BIND
    end
  end
end
Remove from BIND those that are non-maximal
Return BIND

Algorithm 1: $DiscoverMaximalBINDs(\mathbf{S}, \Gamma)$: find max-BINDs implied by Γ

To simplify the discovery process, we assume that there are no redundant columns within one relation, that is, there are no two distinct attributes A and B of a relation R such that $\Gamma \models \pi_A(R) = \pi_B(R)$. A simple preprocessing phase can easily remove redundant columns within one relation. Even without preprocessing, they will be removed anyway in the later project-minimization phase (section 3.2).

Theorem 2. *If there are no redundant columns within a relation, the algorithm* DiscoverMaximalBINDs *finds all and only the maximal BINDs implied by* Γ.[2]

Having discovered all maximal BINDs, we can easily prune those that are not via keys. Testing key dependencies can be also performed using chase [1]. If we denote the maximal BINDs via key of each pair of source relations as a set, then enumeration of maximal collapse-configurations can be done via enumerating elements of the cartesian product of these (non-empty) sets.

Example 4. Consider the input mapping system in Example 1. Chasing with a singleton Go-flight tuple does not produce any other fact, which means there is no IND originating from Go-flight. Chasing with Go-price(n, d, p) produces two other tuples Go-flight$(n, \perp_1, \perp_2)$ and Ok-flight$(n, d, \perp_1, p, \perp_3)$, which correspond to the INDs Go-price$[num] \subseteq$ Go-flight$[num]$ and Go-price$[num, date, price] \subseteq$Ok-flight[num, date, price]. In our data structure, we record (Go-price, Go-flight, $\{(1, 1)\}$) and (Go-price, Ok-flight, {(1,1), (2,2), (3,4)}). Similarly, chasing with a singleton Ok-flight tuple reveals INDs Ok-flight$[num, time] \subseteq$ Go-flight$[num, time]$ and Ok-flight$[num, date, price] \subseteq$ Go-price$[num, date, price]$, corresponding to (Ok-flight, Go-flight, {(1,1), (3,2)}) and (Ok-flight, Go-price, $\{(1, 1), (2, 2), (4, 3)\}$). A join on the INDs outputs only one BIND (Ok-flight, Go-price, $\{(1, 1), (2, 2), (4, 3)\}$). The only maximal collapse-configuration contains this BIND.

3.2 Project-minimization over Collapsed Schemas

We have described in [12] a procedure to test redundancy of a set of columns on the source schema with constraints, which is based on testing query rewritability of identity queries of source relations over the projected schema. The procedure can be extended to test redundancy of columns in a collapsed schema. We first define the lineage of a column in a collapsed schema: the *lineage* of column c is the set of source columns that are collapsed into c. Suppose we collapse Go-price and Ok-flight on $num, date, price$ into Ok-flight$'$, the lineage of $price$ in Ok-flight$'$ is {Go-price.$price$, Ok-flight.$price$}, while the lineage of Ok-flight$'$.$nonstop$ is {Ok-flight.$nonstop$}.

Proposition 1. *A column on a collapsed schema is redundant if and only if all columns in its lineage are redundant.*

With the above result, we can extend the project-minimization algorithm in [12] to be over collapsed schemas, through replacing the redundancy test by testing redundancy of the lineage over the source schemas. The procedure is presented in Algoirthm 2. Two sub-procedures are used in the algorithm: $IsRedudant$ and $Enum$. The former is the column redundancy test, while the latter is a depth-first procedure enumerating all possible sets of columns over the collapsed schema $\mathbf{T}$ with a-priori pruning. For brevity, we refer interested readers to [12] for more details on the two sub-procedures.

[2] Proofs of this theorem and the following are in the appendix, accessible at `http://dbis.rwth-aachen.de/cms/staff/li/er11`

Input : A source schema $\mathbf{S}$, a set of dependencies Γ over $\mathbf{S}$ with terminating chase, a target schema $\mathbf{T}$ that is a collapsing result, a set of s-t tgds Σ_{st} between $\mathbf{S}$ and $\mathbf{T}$ representing the collapsing, and a set of columns in $\mathbf{T}$

Output: A list of maximal sets of redundant columns.

```
1  Initialize isMaximal to TRUE
2  for each extension of the redundant columns P' ∈ Enum(P, T) do
3  |  if IsRedundant(S, Γ, Σ_st, P') then
4  |  |  isMaximal ← FALSE
5  |  |  ProjectMinimize(S, Γ, T, Σ_st, P')
6  |  end
7  end
8  if isMaximal then
9  |  Add P to result
10 end
```

Algorithm 2: $ProjectMinimize(\mathbf{S}, \Gamma, \mathbf{T}, \Sigma_{st}, P)$: p-min over collapsed schema

Example 5 (Example 4 cont.). After collapsing, there are two relations Go-flight(num, $time$, $meal$) and Ok-flight$'$($num, date, time, price, nonstop$). Only the column $time$ in Ok-flight' is redundant, since the data dependencies imply $\pi_{num,date,price,nonstop}$(Ok-flight$'$)$\bowtie$ $\pi_{num,time}$(Go-flight) $\equiv$ Ok-flight$'$.

4 Generating Minimal Merge

A merge-configuration describes how a minimal mediated schema can be constructed as a view over the source schema. We address now how to generate a minimal merge, including constraints over the mediated schema and output mappings.

4.1 Constructing Mediated Schema

Given a merge-configuration, constructing a mediated schema is straightforward. We show that via the running example. The minimization algorithm will produce a collapse-configuration consisting of the BIND (Ok-flight, Go-price, $\{(1,1),(2,2),(4,3)\}$) as computed in Example 4, and the set of redundant columns $\{$Go-flight.$time\}$. Following the collapse-configuration, we can join together Ok-flight and Go-price on num, $date$, $price$ and removing the *time* column, while the relation Go-flight is copied. The mediated schema consists of two relations Ok-flight$'$ and Go-flight$'$, which is a result of transformations defined by a set of full s-t tgds: Go-flight($num, time, meal$) $\rightarrow$ Go-flight$'$($num, time, meal$) and Ok-flight(n, d, t, p, s), Go-price(n, d, p) $\rightarrow$ Ok-flight$'$ (n, d, p, s). We denote the transformation mapping by Σ.

4.2 Constructing Output Mappings

We can show that the transformation mapping Σ in the previous section is tgd invertible.

Proposition 2. *Let Σ be the transformation mapping corresponding to a merge configuration, Γ_s be the input mapping constraints over the source schema* $\mathbf{S}$*,* $\mathbf{T}$ *be the mediated schema, then the mapping* $(\mathbf{S}, \mathbf{T}, \Gamma_s \cup \Sigma)$ *is tgd invertible.*

The output mapping supporting source schemas as views can be computed as a tgd-inverse of Σ. It is shown in [9] that a tgd-inverse of s-t tgds can be expressed in *full* tgds. Here we describe the procedure following [9]. Let $\mathbf{S}$ be a source schema, Γ_s a set of finitely chaseable tgds and egds over $\mathbf{S}$, Σ a set of tgd invertible full s-t tgds. We can create a tgd inverse Σ^{-1} in full s-t tgds as follows. For each source relation R, we denote by I_R the singleton instance of all distinct variables for each column of R, $I_R^{\Gamma_s}$ the result of chasing I_R against Γ_s, J_R the result of chasing $I_R^{\Gamma_s}$ against Σ, add to Σ^{-1}: $J_R \rightarrow I_R^{\Gamma_s}$.

Let's consider the running example. The procedure above creates a tgd inverse consisting of the following tgds : Go-flight$'(n, t, m) \rightarrow$ Go-flight(n, t, m), Ok-flight$'(n, d, p, s)$, Go-flight$'(n, t, m) \rightarrow$ Go-price(n, d, p), and Ok-flight$'(n, d, p, s)$, Go-flight$'(n, t, m) \rightarrow$ Ok-flight(n, d, t, p, s).

4.3 Rewriting Input Constraints to the Mediated Schema

In order to maintain one-to-one instance level mapping between the mediated schema and the source schema, the input constraints have to be rewritten over the mediated schema. It is easy to see that a conjunctive query (CQ) over the source schema can always be rewritten as a CQ over the mediated schema under the tgd-inverse. Let $rewrite(.)$ be the procedure rewriting each CQ over the source schema $\mathbf{S}$ to a CQ over the mediated schema $\mathbf{T}$, while leaving the CQ over $\mathbf{T}$ and equalities intact. For each tgd $\phi(\mathbf{x}) \rightarrow \psi(\mathbf{x})$ in $\Sigma^* \cup \Gamma_s$, add to Γ_t: $rewrite(\phi(\mathbf{x})) \rightarrow rewrite(\psi(\mathbf{x}))$. Egds are handled similarly. Rewriting of Γ_s ensures that the view instances computed from the mediated schema always respect the input constraints, while rewriting Σ^* ensures the mediated schema can be computed as a view from the consistent view instances.

Taking the running example as an input, constraint rewriting produces Γ_t as a set of tgds and egds, which can be simplified to: Go-flight$'(n, t_1, m_1)$, Go-flight$'(n, t_2, m_2) \rightarrow t_1 = t_2, m_1 = m_2$, Ok-flight$'(n, d_1, p_1, s_1)$, Ok-flight$'(n, d_2, p_2, s_2) \rightarrow d_1 = d_2, t_1 = t_2, p_1 = p_2, s_1 = s_2$, and Ok-flight$'(n, d, p, s) \rightarrow$ Go-flight$'(n, t, m)$. We can show that the rewriting procedure is information preserving.

Proposition 3. *The constraint rewriting procedure satisfies* $\Gamma_t \cup (\Sigma^{-1})^* \equiv \Sigma^* \cup \Gamma_s$.

Now we can conclude on the correctness of our merging algorithm. Let Γ_s be the input mapping constraints, c be a merge configuration resulting from the schema minimization procedure. Let $\mathbf{T}$, Γ_t and Σ^{-1} be respectively, the mediated schema, rewritten constraints, and generated output mapping according to c. As Σ^{-1} is in full tgds, it can be split to full tgds with only a single predicate in the head. We denote by Σ_1^{-1} (resp. Σ_2^{-1}) the subset of Σ^{-1} with head predicates only from $\mathbf{S}_1$ (resp. $\mathbf{S}_2$).

Theorem 3. $(\mathbf{T}, \Gamma_t \cup (\Sigma_1^{-1})^*, \Gamma_t \cup (\Sigma_2^{-1})^*)$ *is a minimal merge.*

5 Evaluation

The system is developed using Java SE 6 and SWI-Prolog 5.8.0. A chase procedure is implemented in Prolog for reasoning over data dependencies. The experiments have been carried out on a 2.5GHz dual core computer, with a maximal heap size of 512M. Disk I/O costs are excluded from profiling, while communication costs between Java and Prolog are included. Two collections of data sets are used in our experiments: data sets from the Illinois Semantic Integration Archive[3], which include five real world data sets (*Courses*, *Real Estate II*, *Inventory*, *Faculty* and *Real Estate I*), and a workload generated by a schema mapping benchmark *STBenchmark* [2]. The schemas are defined as XML schemas, which are flattened to relational schemas beforehand.

5.1 Expressiveness over Real World Mapping Scenarios

All five data sets in the Illinois archive are able to be represented by tgds and egds. Complex relationships involving joins of relations arise frequently in the data sets, which confirms the necessity of an expressive language such as tgds. Furthermore, we see it is crucial to be able to specify integrity constraints of source schemas. Without the presence of keys in the source no attribute is redundant, even if many attributes are asserted to be equivalent. This is in line with information preservation: when there is no functional dependency, a tuple is only retrievable when all components are kept.

Eight out of ten basic mapping scenarios of STBenchmark are expressible in our mapping language: *Copy*, *Flattening*, *Nesting*, *Denormalization*, *SelfJoin*, *SurrogateKey*, *AtomicValueManagement* and *VerticalPartitioning*. The scenario *Fusion* is not able to be expressed because it requires a language of disjunctive tgds, which is beyond the scope of our current implementation. The scenario *HorizontalPartitioning* cannot be expressed perfectly as it requires expressing selection in the head of a tgd. This requires a representation system like *conditional tables* [11], while the database with labelled nulls used in our approach is in essence *v-tables* [11].

Value conversion functions arising in the data sets (e.g., in *Real Estate II* and *AtomicValueManagement*) are handled by skolem functions and invertible functions are handled with helper predicates and rules as explained in [12].

5.2 Scalability of Schema Minimization

Though there are multiple possible minimal mediated schemas, enumerating all is very costly. There are cases, e.g., project-minimization in the *Copy* scenario, that have an exponential number of possibilities. We focus on the scalability of finding one minimal mediated schema. For each scenario of STBenchmark, we generate 10 input mappings of various sizes. Three strategies of minimizations, project-minimization (p-min), collapse-minimization (c-min) and collapse-project-minimization (c-p-min), are tested over the generated workload. Fig. 2 depicts the running time. The x-axis is the size of the schema (i.e., the sum of the arities of all relations in the schema), while the y-axis

[3] `http://pages.cs.wisc.edu/~anhai/wisc-si-archive/`

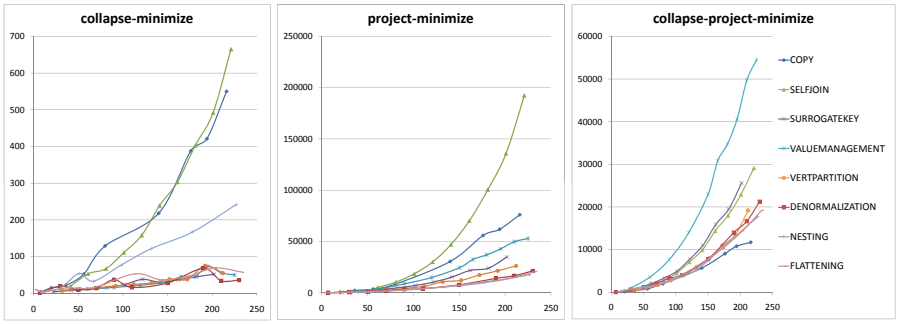


Fig. 2. Running Time (ms) vs Schema Size Categorized by Scenarios of STBenchmark

is the time in milliseconds. A cross comparison among the three minimization strategies reveals that *c-min* is the fastest, while *c-p-min* is faster than *p-min*. *C-min* is fastest as it requires only invoking reasoning procedures in so many times as the number of relations in the input. *P-min* is the most expensive, as it invokes reasoning procedures in proportional to the number of columns in the input. *C-p-min* is faster than *p-min* because the collapsing phase merges columns and hence decreases the input size for *p-min*. *C-min* scales quite well regarding schema size, less than 1 second even for the largest input. *Copy* and *SelfJoin* require a longer time than the other scenarios. This is due to two factors: 1) the portion of attributes identified as overlapping via BINDs in these two scenarios is high; and 2) they have the longest dependency length under a given size of schema. *AtomicValueManagement* also has a long dependency length, but there is no BIND in this scenario since values undergo conversions in the form of a function. Therefore, *c-min* for this scenario is faster. *Nesting* is slightly more costly than the rest as, for the same schema size, the number of relations is larger in this scenario with many leaf relations from normalizing a tree. In *p-min* the reasoning procedure invocations are proportional to the schema size, and the complexity of dependencies is a dominating factor. This explains why the three scenarios *Copy*, *SelfJoin* and *AtomicValueManagement* with the longest dependency take more time than others. *C-p-min* is more scalable than *p-min* because the collapsing minimization phase greatly reduce the input size of the project-minimization phase. Interestingly, now *AtomicValueManagement* is the most expensive scenario. This is because no reduction of schema is achieved during *c-min* for the scenario and hence the cost is almost the same as direct *p-min*.

5.3 Effectiveness of Minimization

We evaluated the effectiveness of our approach using the real world data sets from Illinois archive. We perform three strategies of minimization and present in Fig. 3 the effects of minimization. We measure the ratio of reduced columns to the input schema size, i.e., $1 - \frac{|\mathbf{T}|}{|\mathbf{S}|}$, which indicates how much redundancy has been removed through minimization. As a comparison, [14] and [3] do not head for reducing schema size and have a constantly 0% reduction ratio. [17] considers retaining incompleteness in data integration and may result in a negative reduction ratio, i.e., a schema larger than the

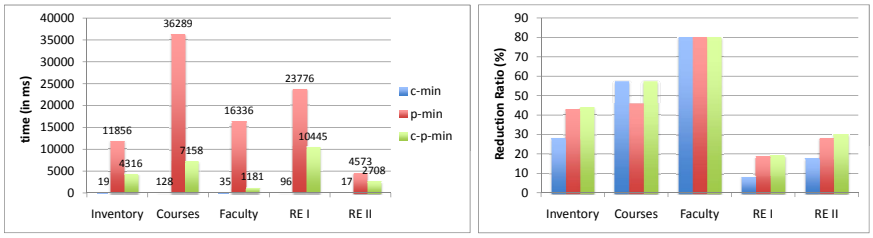


Fig. 3. Schema Reduction Ratio and Running Time of Illinois Data Sets

input. The reduction ratio is the highest for *Faculty*, in which 80% of the input schema is redundant. A close look into the data set reveals that this input consists of five replicas of the same schema. *C-p-min* gives the highest reduction ratio, while whether *c-min* or *p-min* achieves a higher reduction ratio depends on the input scenario.

6 Related Work

Batini et al. [4] provide an early survey covering classical view/data integration approaches, in which schemas are usually modeled in a variant of the ER model. Interschema assertions [22, 18] are a popular language specifying set-based relationships (e.g., inclusion, disjoint, and equal) between extensions of concepts across schemas. The approaches usually undergo a collapse-resolve procedure: first, collapse equivalent concepts and then resolve the conflicts arising in collapsing. This line of work does not focus on creating output mappings, but on designing the mediated schema.

Logical constraints are first used for view integration in [6] and [7] targeting at a minimal mediated schema. Their interschema assertions are one-to-one relation-wise constraints; key constraints have also to be present. Except for the disjoint constraints stating two relations are disjoint on the instance level, the mapping language is strictly less expressive than the language considered in this paper.

Melnik (Theorem 4.2.4 in [14]) proposes a straightforward algorithm for view integration. The mediated schema is taken to be a disjoint union of the source schemas, with source integrity constraints (ICs) and input mappings encoded as constraints over the mediated schema. Output mappings are identity mappings copying part of the mediated schema to the source schemas. Arenas et al. [3] extend the work to achieve a smaller instance for the mediated schema by adding denial constraints to the mediated schema, while confining input mappings to full source-to-target tgds (full s-t tgds). Similar to the previous approaches, we also require a one-to-one mapping between input schemas and mediated schema on the instance level; output view definition mappings are also in full tgds; and constraints are constructed over the mediated schema. However, we go one step further on schema minimization. We find minimal mediated schemas within a class of Project-Join views of the source schemas, resulting in smaller mediated schemas than those created in [3] and [14]. Our input mapping language is (finitely chaseable) tgds and egds, which is neither restricted to be source-to-target, nor restricted to deny existential variables. Constraints over the mediated schema are expressed in tgds and egds, which are more common than the denial constraints used in [3].

Merging schemas in the data integration context differs from view integration, as instances of the source schemas do not necessarily satisfy any mapping constraints. Pottinger and Bernstein [17] use pairs of conjunctive queries (CQs) to specify overlapping information between two schemas. Source integrity constraints are not used in their approach. Chiticariu et al. [8] propose an interactive schema merging approach taking attribute correspondences as input. Logical entities (chases of singleton relations) are extracted from the source schemas as concepts. The space of plausible collapsing of concepts is navigated by the user in an interactive manner. Since each extracted concept has a particular join path in the source schemas, two concepts and value correspondences between them comprise an implicit Global-Local-As-View mapping. However, when deemed as such, the mapping is still more restricted than tgds in the sense that all joins are performed only along referential constraints, e.g., Key-ForeignKey constraints. The work is extended in [19] to generate top-k mediated schemas, with a ranking of candidate mediated schemas, which is a topic not explored in this paper.

In previous work [12, 13], we presented a merging algorithm for data integration, which takes finitely chaseable tgds and egds as input mapping language. The constraints are not regarded as hard constraints over the stored extensions of the data sources (extensional database, EDB), but as intended relationships over an integrated global database (intensional database, IDB). Instead of requiring the mediated schema to preserve the EDB, we require preserving the IDB. We ensure that every source CQ has a rewritten CQ over the mediated schema so that the certain answer of the query in the IDB is the same as the certain answer of the rewriting under the output mapping. We consider removing redundant columns from the source schema. The output mapping created is a mapping representing computing the IDB from EDB composed with the mapping removing redundant columns. The output mapping language needed is second order data dependencies. In this paper, we extend the schema minimization raised there to allow both collapsing and projection. Due to the difference in merge requirements, our formulation of minimal merge for view integration is also different from the requirements of a minimal query interface described there.

7 Conclusion and Outlook

We presented a novel view integration approach which aims at minimizing the mediated schema while still maintaining information preservation. We extend existing work by admitting a more expressive input mapping language consisting of finitely chaseable tgds and egds, and produce output view definition mappings in full s-t tgds. Source integrity constraints are seamlessly incorporated as part of the input constraints. Schema minimization is performed within a class of information preserving Project-Join views, to achieve smaller mediated schemas than in existing solutions. Input constraints are rewritten as tgds and egds over the mediated schema.

We see several possibilities of future work. First, we studied merging relational views and it would be interesting to see how the approach extends to nested structures like XML. Second, we focussed on collapsing relations, but the approach can be extended to collapsing logical entities, i.e., chases of singleton relations. Third, recent

results on disjunctive chase suggest that the input mapping language can be extended to disjunctive tgds.

Acknowledgements: This work is supported by the DFG Research Cluster on Ultra High-Speed Mobile Information and Communication (UMIC, `http://www.umic.rwth-aachen.de`).

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. B. Alexe, W. C. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *PVLDB*, 1(1):230–244, 2008.
3. M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Foundations of schema mapping management. In *Proc. PODS*, pages 227–238, 2010.
4. C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
5. P. A. Bernstein and S. Melnik. Model management 2.0: Manipulating richer mappings. In *Proc. SIGMOD*, pages 1–12, Beijing, China, 2007.
6. J. Biskup and B. Convent. A formal view integration method. In *Proc. SIGMOD*, pages 398–407, Washington, D.C., 1986.
7. M. A. Casanova and V. M. P. Vidal. Towards a sound view integration methodology. In *Proc. PODS*, pages 36–47, Atlanta, GA, 1983. ACM.
8. L. Chiticariu, P. G. Kolaitis, and L. Popa. Interactive generation of integrated schemas. In *Proc. SIGMOD*, pages 833–846, 2008.
9. R. Fagin. Inverting schema mappings. *ACM Transactions on Database Systems*, 32(4), 2007.
10. R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336:89–124, 2005.
11. T. Imielinski and W. Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.
12. X. Li, C. Quix, D. Kensche, and S. Geisler. Automatic schema merging using mapping constraints among incomplete sources. In *Proc. CIKM*, pages 299–308. ACM, 2010.
13. X. Li, C. Quix, D. Kensche, S. Geisler, and L. Guo. Automatic mediated schema generation through reasoning over data dependencies. In *Proc. ICDE*, 2011.
14. S. Melnik. *Generic Model Management: Concepts and Algorithms*. PhD thesis, Universität Leipzig, 2004.
15. S. Melnik, P. A. Bernstein, A. Y. Halevy, and E. Rahm. Supporting executable mappings in model management. In *Proc. SIGMOD*, pages 167–178. ACM Press, 2005.
16. R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In *Proc. VLDB*, pages 120–133, 1993.
17. R. Pottinger and P. A. Bernstein. Schema merging and mapping creation for relational sources. In *Proc. EDBT*, 2008.
18. C. Quix, D. Kensche, and X. Li. Generic schema merging. In *Proc. CAiSE'07*, volume 4495 of *LNCS*, pages 127–141, 2007.
19. A. Radwan, L. Popa, I. R. Stanoi, and A. A. Younis. Top-k generation of integrated schemas based on directed and weighted correspondences. In *Proc. SIGMOD*, pages 641–654, 2009.
20. A. D. Sarma, X. Dong, and A. Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *Proc. SIGMOD*, pages 861–874, 2008.
21. S. Spaccapietra and C. Parent. View integration: A step forward in solving structural conflicts. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):258–274, 1994.
22. S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogeneous schemas. *VLDB Journal*, 1(1):81–126, 1992.