Estimating Data Integration and Cleaning Effort

Sebastian Kruse[1], Paolo Papotti[2], Felix Naumann[1]
[1] Hasso Plattner Institute – Germany, [2] Qatar Computing Research Institute (QCRI), Qatar
sebastian.kruse@hpi.de,ppapotti@qf.org.qa,felix.naumann@hpi.de

ABSTRACT

Data cleaning and data integration have been the topic of intensive research for at least the past thirty years, resulting in a multitude of specialized methods and integrated tool suites. All of them require at least some and in most cases significant human input in their configuration, during processing, and for evaluation. For managers (and for developers and scientists) it would be therefore of great value to be able to estimate the *effort* of cleaning and integrating some given data sets and to know the pitfalls of such an integration project in advance. This helps deciding about an integration project using cost/benefit analysis, budgeting a team with funds and manpower, and monitoring its progress. Further, knowledge of how well a data source fits into a given data ecosystem improves source selection.

We present an extensible framework for the automatic effort estimation for mapping and cleaning activities in data integration projects with multiple sources. It comprises a set of measures and methods for estimating integration complexity and ultimately effort, taking into account heterogeneities of both schemas and instances and regarding both integration and cleaning operations. Experiments on two real-world scenarios show that our proposal is two to four times more accurate than a current approach in estimating the time duration of an integration process, and provides a meaningful breakdown of the integration problems as well as the required integration activities.

1. COMPLEXITY OF INTEGRATION AND CLEANING

Data integration and data cleaning remain among the most human-work-intensive tasks in data management. Both require a clear understanding of the semantics of schema and data – a notoriously difficult task for machines. Despite much research and development of supporting tools and algorithms, state-of-the-art integration projects involve significant human resource cost. In fact, Gartner reports that 10% of all IT cost goes into enterprise software for data integration and data quality[1,2], and it is well recognized that most of those expenses are for human labor. Thus, when embarking on a data integration and cleaning project, it is useful and important to estimate in advance the effort and cost of the project and to find out which particular difficulties cause these. Such estimations help deciding whether to pursue the project in the first place, planning and scheduling the project using estimates about the duration of integration steps, budgeting in terms of cost or manpower, and finally monitoring the progress of the project. Cost estimates can also help integration service providers, IT consultants, and IT tool vendors to generate better price quotes for integration customers. Further, automatically generated knowledge of how well and how easy a data source fits into a given data ecosystem improves source selection.

However, "*project estimation for [...] data integration projects is especially difficult, given the number of stakeholders involved across the organization as well as the unknowns of data complexity and quality.*" [14]. Any integration project has several steps and tasks, including requirements analysis, selection of data sources, determining the appropriate target database, data transformation specifications, testing, deployment, and maintenance. In this paper, we focus on exploring the database-related steps of *integration* and *cleaning* and automatically estimate their effort.

1.1 Challenges

There are simple approaches to estimate in isolation the complexity of individual mapping and cleaning tasks. For the mapping, evaluating its complexity can be done by counting the matchings (i.e., correspondences) among elements. For the cleaning problem, a natural solution is to measure its complexity by counting the number of constraints on the target schema. However, as several integration approaches have shown, the interactive nature of these two problems is particularly complex [5, 11, 13]. For example, a data exchange problem takes as input two relational schemas, a transformation between them (a mapping), a set of target constraints, and answers two questions: whether it is possible to compute a valid solution for a given setting and how. Interestingly, to have a solution, certain conditions must hold on the target constraints, and extending the setting to more complex languages or data models bring tighter restrictions on the class of tractable cases [6, 12].

In our work, the main challenge is to estimate complexity and effort in a setting that goes beyond these ad-hoc studies while satisfying four main requirements:

Generality: We require independence from the language used to express the data transformation. Furthermore, real cases often fail the existence of solution tests considered in formal frameworks

[1] *http://www.gartner.com/technology/research/it-spending-forecast/*
[2] *http://www.gartner.com/newsroom/id/2292815*

(e.g., weak acyclicity condition [11]), but an automatic estimation is still desirable for them in practice.

Completeness: Only a subset of the constraints that hold on the data are specified over the schema. In fact, business rules are commonly enforced at the application level and are not reflected in the metadata of the schemas, but should nevertheless be considered.

Granularity: Details about the integration issues are crucial for consumption of the estimation. For a real understanding and proper planning, it is important to know which source and/or target attributes are cause of problems and how (e.g., phone attributes in source and target schema have different formats). Existing estimators do not reason over actual data structures and thus make no statements about the causes of integration effort.

Configurability and extensibility: The actual effort depends on subjective factors like the capabilities of available tools and the desired quality of the output. Therefore, intuitive, yet rich configuration settings for the estimation process are crucial for its applicability. Moreover, users must be able to extend the range of problems covered by the framework.

These challenges cannot be tackled with existing syntactical methods to test the existence of solutions, as they work only in specific settings (*Generality*), are restricted to declarative specifications over the schemas (*Completeness*), and do not provide details about the actual problems (*Granularity*). On the other hand, as systems that compute solutions require human interaction to finalize the process [8, 13], they cannot be used for estimation purpose and their availability is orthogonal to our problem (*Configurability*).

1.2 Approaching Effort Estimation

Figure 1 presents our view on the problem of estimating the effort of data integration. The starting point is an integration scenario with a target database and one or more source databases. The right-hand side of the figure shows the actual integration process performed by an integration specialist, where the goal is to move all instances of the source databases into the target database. Typically, a set of integration tools are used by the specialist. These tools have access to the source and target and support her in the tasks. The process takes a certain effort, which can be measured, for instance as amount of work in hours or days or in a monetary unit.

Our goal is to find that effort without actually performing the integration. Moreover, we want to find and present the problems that cause this effort. To this end, we developed a two-phase process as shown on the left-hand side of Figure 1.

The first phase, the *complexity assessment*, reveals concrete integration challenges for the scenario. To address *generality*, these problems are exclusively determined by the source and target schemas and instances; if and how an integration practitioner deals with them is not addressed at this point. Thus, this first phase is independent of external parameters, such as the level of expertise of the specialist or the available integration tools. However, it is aided by the results of schema matching and data profiling tools, which analyze the participating databases and produce metadata about them (to achieve *completeness*). The output of the complexity assessment is a set of clearly defined problems, such as number of violations for a constraint or number of different value representations. This detailed breakdown of the problems achieves *granularity* and is useful for several tasks, even if not interpreted as an input to calculate actual effort. Examples of application are *source selection* [9], i.e., given a set of integration candidates, find the source with the best 'fit'; and support for *data visualization* [7], i.e., highlight parts of the schemas that are hard to integrate.

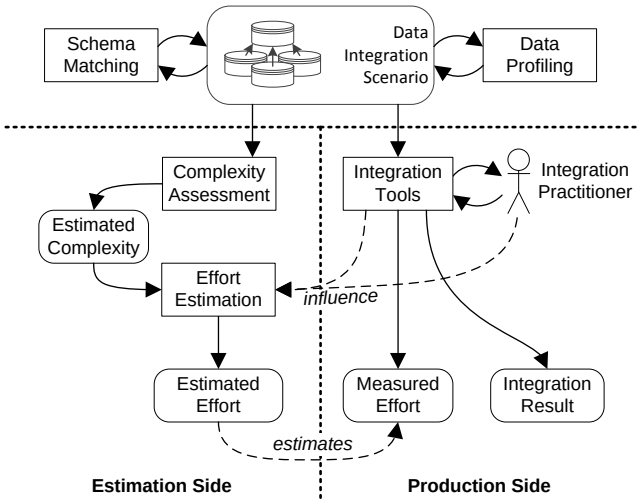


Figure 1: Overview of effort estimation and execution of data integration scenarios.

The second phase, *effort estimation*, builds upon the complexity assessment to estimate the actual effort for overcoming the previously revealed integration challenges in some useful unit, such as workdays or monetary cost. Thereby, this phase addresses *configurability* by taking external parameters into account, such as the experience of the integration practitioner and the features of the integration tools to be used.

1.3 Contributions and structure

Section 2 presents related work and shows that we are the first to systematically address a dimension of data integration and cleaning that has been passed over by the database community but is relevant to practitioners. In particular, we make the following contributions:

- Section 3 introduces the extensible Effort Estimation framework (EFES), which defines a two-dimensional modularization of the estimation problem.
- Section 4 describes an estimation module for structural conflicts between source and target data. This module incorporates a new formalism to compare schemas in terms of mappings and constraints.
- Section 5 reports an estimation module for value heterogeneities that captures formatting problems and anomalies in data that may be missed by structural conflicts.

These building blocks have been evaluated together in an experimental study on two real-world datasets and Section 6 reports on the results. Finally, we conclude our insights in Section 7.

2. RELATED WORK

When surveying computer science literature, a pattern becomes apparent: much technology claims (and experimentally shows) to reduce human effort. The veracity of this claim is evident – after all, any kind of automation of tedious tasks is usually helpful. While for scientific papers this reduction is enough of a claim, the absolute measures of effort and its reduction is rarely explained and measured.

General effort estimation. There are several approaches for effort estimation in different fields, however, none of them considers information coming from the datasets.

In the software domain, an established model to estimate the cost of developing applications is COCOMO [3, 4], which is based on parameters provided by the users such as the number of lines of existing code. Another approach decomposes an overall work task into a smaller set of tasks in a "work breakdown structure" [16]. The authors manually label business requirements with an effort class of simple, medium, or complex, and multiply each of them by the number of times the task must be executed.

In the ETL context, Harden [14] breaks down a project into various subtasks, including requirements, design, testing, data stewardship, production deployment, but also the actual development of the data transformations. For the latter he uses the number of source attributes and assigns for each attribute a weighted set of tasks (Table 1). In sum, he calculates slightly more than 8 hours of work for each source attribute.

Task	Hours per attribute
Requirements and Mapping	2.0
High Level Design	0.1
Technical Design	0.5
Data Modeling	1.0
Development and Unit Testing	1.0
System Test	0.5
User Acceptance Testing	0.25
Production Support	0.2
Tech Lead Support	0.5
Project Management Support	0.5
Product Owner Support	0.5
Subject Matter Expert	0.5
Data Steward Support	0.5

Table 1: Tasks and effort per attribute from [14].

One can find other lists of criteria to be taken into account when estimating the effort of an integration project[3]. These include factors we include in our complexity model, such as number of different sources and types, duplicates, schema constraints, and others we exclude for sake of space from our discussion, such as project management, deployment needs, and auditing. There are also mentions of factors that influence our effort model, such as familiarity with the source database, skill levels, and tool availability. However, merely providing a list of factors is only a first step, whereas we provide novel measures for the database-specific causes for complexity and effort. In fact, existing methods: (i) lack a direct numerical analysis of the schemas and datasets involved in the transformation and cleaning; (ii) do not regard the properties of the datasets at a fine grain and cannot capture the nature of the possible problems in the scenario, (iii) do not consider the interaction of the mapping and the cleaning problems.

Schema-matching for effort estimation. In our work, we exploit schema matching to bootstrap the process. This is along the lines of what authors of matchers suggested. For example, in [24] the authors have pointed out the multiple usages of schema matching tools beyond the concrete generation of correspondences for schema mappings. In particular, they mention "project planning" and "determining the level of effort (and corresponding cost) needed for an integration project". In a similar fashion, in the evaluation of the similarity flooding algorithm, Melnik et al. propose a novel measure "to estimate how much effort it costs the user to modify the proposed match result into the intended result" in terms of additions and deletions of matching attribute pairs [19].

Data-oriented effort estimation. In [25], the authors offer a "back of the envelope" calculation on the number of comparisons needed to be performed by human workers to detect duplicates in a dataset. According to them, the estimate depends on the way the potential duplicates are presented, in particular their order and their grouping. Their ideas fit well into our effort model and show that specific tooling indeed changes the needed effort, independently of the complexity of the problem itself. Complementary work on source selection has focused on the benefit of integrating a new source based on its marginal gain [9, 23].

Data-cleaning and data-transformation. Many systems (e.g., [8, 13]) address the problem of detecting violations over the data given a set of constraints, as we also do in one of our modules for complexity estimation. The challenge for these systems is mostly the automatic repair step, i.e., how to update the data to make it consistent wrt. the given constraints with a minimal number of changes. None of these systems provide tools to estimate the complexity of the repair nor the user effort before actually executing the methods to solve the integration problem. In fact, the challenge is that solving the problem involves the users, and estimating this effort (even in presence of these tools) is our main goal. Similar problems apply to data exchange and data transformation [1, 15].

In the field of model management, the use of metamodels has been investigated to represent in a more general language several alternative data models [2, 21]. Our cardinality-constrained schema graphs (Section 4) can be seen as a proposal for a metamodel with a novel static analysis of cardinalities to identify problems in the underlying schemas and the mapping between them.

3. AN EFFORT ESTIMATION FRAMEWORK

Real-world data integration scenarios host a large number of different challenges that must be overcome. Problems arise in common activities, such as the definition of a mapping between different schemas, the restructuring of the data, and the reconciliation of their value format. We first describe these problems and then introduce our solution.

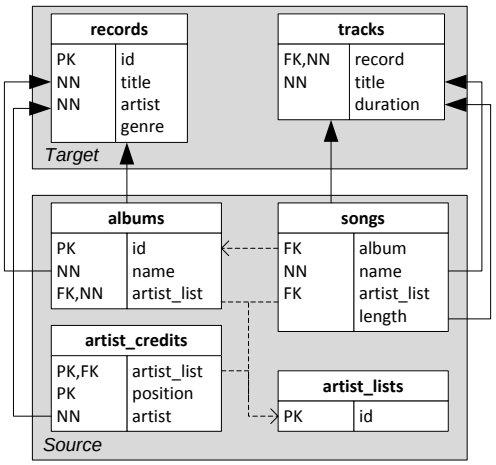
3.1 Data Integration Scenario

A *data integration scenario* comprises: (i) a set of source databases; (ii) a target database, into which the source databases shall be integrated; and (iii) correspondences to describe how these sources relate to the target. Each *source database* consists of a relational schema, an instance of this schema, and a set of constraints, which must be satisfied by that instance. Likewise, the *target database* can carry constraints and possibly already contains data as well that satisfies these constraints. Furthermore, each *correspondence* connects a source schema element with the target schema element, into which its contents should be integrated.

Oftentimes constraints are not enforced at the schema level but rather at the application level or simply in the mind of the integration expert. Even worse, for some sources (e.g., data dumps), a schema definition may be completely missing. To achieve *completeness*, techniques for schema reverse engineering and data profiling [20] can reconstruct missing schema descriptions and constraints from the data.

Example 3.1. *Figure 2 shows an integration scenario with music records data. Both source and target relational schemas (Figure 2a) define a set of constraints, such as primary keys (e.g.,* id *in*

[3]such as *http://www.datamigrationpro.com/data-migration-articles/2012/2/9/data-migration-effort-estimation-practical-techniques-from-r.html* and *http://www.information-management.com/news/1093630-1.html*



(a) Schemas, constraints, and correspondences.

record	title	duration
1	"Sweet Home Alabama"	"4:43"
1	"I Need You"	"6:55"
1	"Don't Ask Me No Questions"	"3:26"
...		

(b) Example instances from the target table tracks.

album	name	artist_list	length
s3	"Hands Up"	a1	215900
s3	"Labor Day"	a1	238100
s3	"Anxiety"	a2	218200
...			

(c) Example instances from the source table songs.

Figure 2: An example data integration scenario.

records*), foreign keys (*record *in* tracks*, represented with dashed arrows), and not nullable values (*title *in* tracks*).*

Solid arrows between attributes and relations represent correspondences, i.e., two attributes that store the same atomic information or two relations that store the same kind of instances. The source relation albums *corresponds to the target relation* records *and its source attribute* name *corresponds to the* title *attribute in the target. That means, that the albums from the source shall be integrated as records into the target, while the source album names serve as titles for the integrated records.* ◇

We assume correspondences between the source and target schemas to be given, as they can be automatically discovered with schema matching tools [10]. Notice that correspondences are not an executable representation of a transformation, thus they do not induce a *precise* mapping between sources and the target. However, they contain enough information to reason over the complexity of data integration scenarios and detect their integration challenges, as in the following example.

Example 3.2. *The target schema requires exactly one artist value per record, whereas the source schema can associate an arbitrary number of artist credits to each album. This implies that integrating any source album with zero artist credits violates the not-null constraint on the target attribute* records.artist*. Moreover, two or more artist credits for a single source album cannot be naturally stored by the single target attribute. Integration practitioners will have to solve these conflicts. Hence, this schema heterogeneity increases the necessary effort to achieve the integration.* ◇

Not all kinds of integration issues can be detected by analyzing the schemas, though. The data itself is equally important. While we assume that every instance is valid wrt. its schema, when data is integrated new problems can arise. For example, all sources might be free of duplicates, but there still might be target duplicates when they are combined [22]. These conflicts can also arise between source data and pre-existing target data.

Example 3.3. *Tables 2b and 2c report sample instances of the* tracks *table and the* songs *table, respectively. The duration of tracks in the target database is encoded as a string with the format m:ss, while the length of songs is measured in milliseconds in the source. The two formats are locally consistent, but the source values need a proper transformation when integrated into the target column, thereby demanding a certain amount of effort.* ◇

3.2 A General Framework

Facing different kinds of data integration and cleaning actions, there is the need of different specialized models to decode their complexity and estimate their effort properly. We tackle this problem with our general effort estimation framework EFES. It handles different kinds of integration challenges by accepting a dedicated *estimation module* to cope with each of them independently. Such modularity makes it easier to revise and refine individual modules and establishes the desired *extensibility* by plugging new ones. In this work, we present modules for the three general and, in our experience, most common classes of integration activities: writing an executable mapping, resolving structural conflicts, and eliminating value heterogeneities. While the latter two are explained in subsequent sections, we present the *mapping module* in this section to explain our framework design. For *generality*, the modules do not depend on a fixed language to express the transformations.

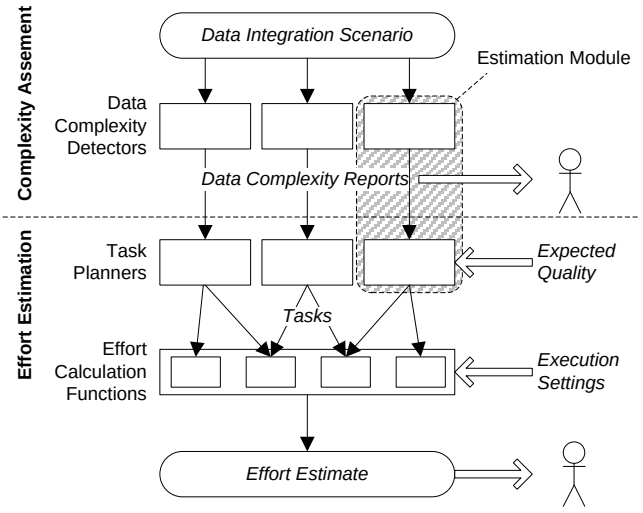


Figure 3: Architecture of EFES.

Figure 3 depicts the general architecture of EFES. The architecture implements our goal of delivering a set of integration problems and an effort estimate by explicitly dividing the estimation process into an objective *complexity assessment*, which is based on properties of schemas and data, followed by the context-dependent *effort estimation*. We now describe these two phases in more detail.

3.3 Complexity assessment

The goal of this first phase is to compute *data complexity reports* for the integration scenario. These reports serve as the basis for the

subsequent effort estimation but also are used to inform the user about integration problems within the scenario. This is particularly useful for source selection [9] and data visualization [7].

Each estimation module provides a *data complexity detector* that extracts complexity indicators from the given scenario and writes them into its report. There is no formal definition for such a report; rather, it can be tailored to the specific, needed complexity indicators. For example, the mapping module builds on the following idea: For each table in the target schema and each source database that provides data for that table, some connection has to be established to fetch the source data and write it into the target table. The overall complexity of the mapping creation is composed of the individual complexities for establishing each of these connections. Furthermore, every connection can be described in terms of certain metrics, such as the number of source tables to be queried, the number of attributes that must be copied, and whether new IDs for a primary key need to be generated.

Target table	Source tables	Attributes	Primary key
records	3	2	yes
tracks	3	2	no

Table 2: Mapping complexity report of the scenario in Figure 2.

Example 3.4. *The data complexity report for the scenario in Figure 2 can be found in Table 2. To fetch the data for the* records *table, the three source tables* albums*,* artist_lists*, and* artist_credits *have to be combined, two attributes must be copied, and unique* id *values for the integrated tuples must be generated.* ◇

3.4 Effort estimation

Based on the data complexity, the effort estimation shall produce a single estimate of the human work to address the different complexities. However, going from an objective complexity measure to a subjective estimate of human work requires external information about the context. We distinguish one aspect that is specific to the data integration problem, (i) the *expected quality* of the integration result, and, as a more common aspect, (ii) the *execution settings* for the scenario.

(i) Expected quality: Data cleaning is the operation of updating an instance such that it becomes consistent wrt. any constraint defined on its schema [8, 13]. However, such results can be obtained by automatically removing problematic tuples, or by manually solving inconsistencies involving a domain expert. Each choice implies different effort.

Example 3.5. *Consider again Example 3.3 with the duration format mismatch. As* duration *is nullable, one simple way to solve the problem is to drop the values coming from the new source. A better, higher quality solution is to transform the values to a common format, but this operation requires a script and a validation by the user, i.e., more effort [15].* ◇

(ii) Execution settings: The execution settings represent the circumstances under which the data integration shall be conducted. Examples of common context information are the expertise of the integration practitioners and their familarity with the data [4]. In our setting, we also model the level of automation of the available integration tools, and how critical the errors are, e.g., integrating medical prescriptions requires more attention (and therefore effort) than integrating music tracks.

Example 3.6. *Consider again the problem with the cardinality of record artists. There are* schema mappings *tools [18] that are able to automatically create a synthetic value in the target, if a source album has zero artist credits, and to automatically create multiple occurrences of the same album with different artists, if multiple artists are credited for a single source album. Such tools would reduce the mapping effort.* ◇

For the effort estimation, each estimation module has to provide a *task planner* that consumes its data complexity report and outputs *tasks* to overcome the reported issues. Each of these tasks is of a certain type, is expected to deliver a certain result quality, and comprises an arbitrary set of parameters, such as on how many tuples it has to be executed. We defined two instances of expected quality, namely *low effort* (removal of tuples) and *high quality* (updates). This criterion is extensible to other repair actions, but it already allows to choose between alternative cleaning tasks as shown in Example 3.5.

Example 3.7. *A complexity report for the scenario from Figure 2 states that there are albums without any artist in the source data that lead to a violation of a not-null constraint in the target. The corresponding task model proposes the alternative actions* Reject violating tuple *(low effort) or* Add missing value *(high quality) to solve the problem.* ◇

Once the list of tasks has been determined, the effort for their execution is computed. For this purpose, the user specifies in advance for each task type an *effort-calculation function* that can incorporate task parameters. As an example, we report the effort-calculation functions for the execution settings of our experiments in Table 9. The framework uses these functions to estimate the effort for each of the tasks. Finally, the total of all these task estimates forms the overall effort estimate.

Example 3.8. *We exemplify the effort-calculation functions for the tasks derived from the report in Table 2. The* Create mapping *task might be done manually with SQL queries. Then an adequate function would be*

$$\textit{effort} = 3mins \cdot \textit{tables} + 1min \cdot \textit{attributes} + 3mins \cdot \textit{PKs}$$

leading to an overall effort of 25 (18 + 4 + 3) minutes. However, if a tool can generate this mapping automatically based on the correspondences (e.g., [18]), then a constant value such as effort $= 2mins$ *can reflect this circumstance, leading to an overall effort of four minutes.* ◇

The above described task-based approach offers several advantages over an immediate complexity-effort mapping [14], where a formula directly converts statistics over the schemas into an effort estimation value. Our model enables *configurability*, as it treats execution settings as a first-class component in the effort-calculation functions and these can be arbitrarily complex as needed. Furthermore, instead of just delivering a final effort value, our effort estimate is broken down according to its underlying tasks. This *granularity* helps users understand the required work and explains how the estimate has been created, thus giving the users the opportunity to properly plan the integration process.

4. STRUCTURAL CONFLICTS

Structural heterogeneities between source and target data structures are a common problem in integration scenarios. This section describes a module to detect these problems and estimate the effort

arising out of them. It can be plugged into the framework architecture in Figure 3 with the following workflow: Its data complexity detector (*structure conflict detector*) analyzes how source and target data relate to each other, and counts the number of emerging structural conflicts. Based on those conflicts, the task planner (*structure repair planner*) then (i) derives a set of cleaning tasks to make the conflicting source data fit into the target schema, and (ii) estimates how often each such task has to be performed. These tasks can finally be fed into the effort calculation functions.

4.1 Structure Conflict Detector

In the first step of structural conflict handling, all source and target schemas of the given scenario are converted into *cardinality-constrained schema graphs* (short *CSG*), a novel modeling formalism that we specifically devised for our task. It offers a single, yet expressive constraint formalism with a set of inference operators that allow elegant comparisons of schemas. Additionally, it is more general than the relational model and can describe (integrated) database instances that do not conform to the relational model. For instance, an integrated tuple might provide multiple values for a single attribute, like in Example 3.2. The higher expressiveness of CSGs allows to reason about necessary cleaning tasks to make the integrated databases conform to the relational model. In the following, we formally define CSGs and explain how to convert relational databases into CSGs.

DEFINITION 1. *A CSG is a tuple $\Gamma = (N, P, \kappa)$, where N is a set of nodes and $P \subset N^2$ is a set of relationships. Furthermore, $\kappa\colon P \to 2^{\mathbb{N}}$ expresses schema constraints by prescribing cardinalities for relationships.*

DEFINITION 2. *A CSG instance is a tuple $\mathcal{I}(\Gamma) = (\mathcal{I}_N, \mathcal{I}_P)$, where $\mathcal{I}_N$ assigns a set of elements to each node in N and $\mathcal{I}_P$ assigns to each relationship links between those elements.*

To convert a relational schema, for each of its relations, a corresponding *table node* (rectangle) is created to represent the existence of tuples in that relation. Furthermore, for each attribute, an *attribute node* (round shape) is created and connected to its respective table node via a *relationship*. While these attribute nodes hold the *set of distinct* values of the original relational attribute, the relationships link tuples and their respective attribute values. With this proceeding, any relational database can be turned into a CSG without loss of information.

Example 4.1. *Figure 4 depicts two CSGs for the example scenario schemas in Figure 2a, one for the source and one for the target schema*[4]*. For instance, the example* tracks *tuple t = (1, "Sweet Home Alabama", "4:43") from Figure 2b is represented in the CSG instance as follows: The table node* tracks $\in N$ *holds an abstract element id_t, i.e. $id_t \in \mathcal{I}_N(\mathsf{tracks})$, representing the tuple's identity. Likewise,* record $\in N$ *holds exactly once the value 1, i.e. $1 \in \mathcal{I}_N(\mathsf{record})$ and the relationship $\rho_{\mathsf{tracks}\to\mathsf{record}}$ contains a* link *for these elements, i.e. $(id_t, 1) \in \mathcal{I}_P(\rho_{\mathsf{tracks}\to\mathsf{record}})$, thus stating that $t[\mathsf{record}] = 1$. The other values for the* title *and* duration *attributes are represented accordingly. Furthermore, foreign key relationships are represented by special* equality relationships *(dashed line) that link all equal elements of two nodes, e.g., all common values of the* id *and* record *nodes in the target CSG of Figure 4.* ◇

[4]Some correspondences between the schemas are omitted for clarity, but are not generally discarded.

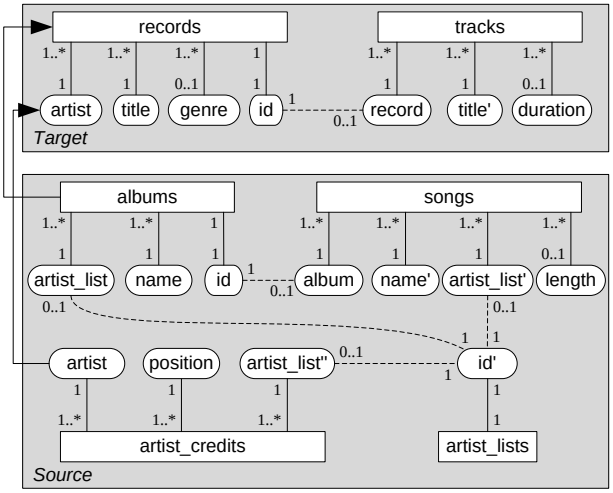


Figure 4: The integration scenario translated into cardinality-constrained schema graphs.

To express schema constraints in CSGs, all relationships are annotated with *prescribed cardinalities*, that restrict the number of elements and/or values of connected nodes that must relate to each other via the annotated relationship. For example, tracks.record is not nullable, which means, that each tracks tuple must provide exactly one record value. Translated to CSGs, this means that for each tuple t_i, the relationship $\rho_{\mathsf{tracks}\to\mathsf{record}}$ must contain exactly one link:

$$\forall t_i\colon\ |\{v \in \mathcal{I}_N(\mathsf{record}) \mid (id_{t_i}, v) \in \mathcal{I}_P(\rho_{\mathsf{tracks}\to\mathsf{record}})\}| = 1 \quad .$$

Formally, this is expressed by $\kappa(\rho_{\mathsf{tracks}\to\mathsf{record}}) = \{1\}$, which is also graphically annotated in Figure 4. However, tracks.record is not subject to a unique-constraint. In consequence, every record value can be found in one or more tuples. Therefore, $\kappa(\rho_{\mathsf{record}\to\mathsf{tracks}}) = 1..* = \{1, 2, 3, \ldots\}$. By means of prescribed cardinalities, unique, not-null, and foreign key constraints can be expressed, as well as two conformity rules for relational schemas: each tuple can have at most one value per attribute, and each attribute value must be contained in a tuple.

As stated above, another important feature of CSGs is the ability to combine relationships into *complex relationships* and to analyze their properties. As one effect, prescribing cardinalities not only to atomic but also to complex relationships further allows to express n-ary versions of the above constraints and functional dependencies. We devised the following relationship construction operators:

- ∘ The *composition* concatenates two adjacent relationships. Formally, $\mathcal{I}_P(\rho_1 \circ \rho_2) \overset{def}{=} \mathcal{I}_P(\rho_1) \circ \mathcal{I}_P(\rho_2)$.
- ∪ The *union* of two relationships $\rho_1 \cup \rho_2$ contains all links of the two relationships, i.e., $\mathcal{I}_P(\rho_1 \cup \rho_2) \overset{def}{=} \mathcal{I}_P(\rho_1) \cup \mathcal{I}_P(\rho_2)$. This is particularly useful, when multiple source relationships need to be combined.
- ⋈ The *join* operator connects links from relationships $\rho_{A\to C}$, $\rho_{B\to C}$ with equal codomain values, thereby inducing a relationship between $A \times B$ and C. Formally, $\mathcal{I}_P(\rho_{A\to C} \bowtie \rho_{B\to C}) \overset{def}{=} \{((a,b),c) : (a,c) \in \mathcal{I}_P(\rho_{A\to C}) \wedge (b,c) \in \mathcal{I}_P(\rho_{A\to B})\}$. The join can be combined with other operators to express n-ary uniqueness constraints.

$\|$ The *collateral* of two relationships $\rho_{A\to B}\|\rho_{C\to D}$ induces a relationship between $A \times C$ and $B \times D$: $\mathcal{I}_P(\rho_{A\to B}\|\rho_{C\to D}) \stackrel{def}{=} \{((a,c),(b,d)) : (a,b) \in \mathcal{I}_P(\rho_{A\to B}) \wedge (c,d) \in \mathcal{I}_P(\rho_{C\to D})\}$. The collateral can be applied to express n-ary foreign keys.

Based on these definitions, efficient algorithms can be devised to infer the constraints of complex relationships.

LEMMA 1. *Let $\rho_1, \rho_2 \in P$ be two relationships in a graph Γ and ρ_1's end node is ρ_2's start node. Then the cardinality κ of $\rho_1 \circ \rho_2$ can be inferred as*

$$\begin{aligned} & \kappa(\rho_1 \circ \rho_2) \stackrel{def}{=} \kappa(\rho_1) \circ \kappa(\rho_2) \\ = \;& a_1..b_1 \circ a_2..b_2 \stackrel{def}{=} (\operatorname{sgn} a_1 \cdot a_2)..(b_1 \cdot b_2) \end{aligned}$$

where $\operatorname{sgn}(0) = 0$ *and* $\operatorname{sgn}(n) = 1$ *for* $n > 0$.

LEMMA 2. *Let $\rho_1, \rho_2 \in P$ be two relationships in a graph Γ. Then the cardinality of $\rho_1 \cup \rho_2$ can be inferred as*

$$\kappa(\rho_1 \cup \rho_2) \stackrel{def}{=} \begin{cases} \kappa(\rho_1) \cup \kappa(\rho_2) & \text{if } \mathcal{I}_P(\rho_1) \text{ and } \mathcal{I}_P(\rho_2) \text{ have disjoint domains} \\ \kappa(\rho_1) + \kappa(\rho_2) & \text{if } \mathcal{I}_P(\rho_1) \text{ and } \mathcal{I}_P(\rho_2) \text{ have equal domains but disjoint codomains} \\ \kappa(\rho_1) \hat{+} \kappa(\rho_2) & \text{if } \mathcal{I}_P(\rho_1) \text{ and } \mathcal{I}_P(\rho_2) \text{ have equal domains and overlapping codomains} \end{cases}$$

where $\kappa_1 + \kappa_2 \stackrel{def}{=} \{a + b : a \in \kappa_1 \wedge b \in \kappa_2\}$ *and* $\kappa_1 \hat{+} \kappa_2 \stackrel{def}{=} \{c : a \in \kappa_1 \wedge b \in \kappa_2 \wedge \max\{a,b\} \le c \le (a+b)\}$.

Note, that Lemma 2 can also be applied to relationships with partially overlapping domains by splitting those into the overlapping and the disjoint parts.

LEMMA 3. *Let $\rho_1, \rho_2 \in P$ be two relationships in a graph Γ with a common end node and let $m = \min\{\max \kappa(\rho_1), \max \kappa(\rho_2)\}$. Then the cardinality of $\rho_1 \bowtie \rho_2$ can be inferred as*

$$\kappa(\rho_1 \bowtie \rho_2) \stackrel{def}{=} \begin{cases} \emptyset & \text{if } m = 0 \vee m = \bot \\ 1..m & \text{otherwise} \end{cases}$$

and its inverse cardinality as

$$\begin{aligned} & \kappa((\rho_1 \bowtie \rho_2)^{-1}) \\ & \stackrel{def}{=} (\min \kappa(\rho_1) \cdot \min \kappa(\rho_2))..(\max \kappa(\rho_1) \cdot \max \kappa(\rho_2)) \end{aligned}$$

LEMMA 4. *Let $\rho_1, \rho_2 \in P$ be two relationships in a graph Γ. Then the cardinality of $\rho_1 \| \rho_2$ can be inferred as*

$$\kappa(\rho_1 \| \rho_2) \stackrel{def}{=} 0..(\max \kappa(\rho_1) \cdot \max \kappa(\rho_2))$$

Given the means to combine relationships and infer their cardinality, it is now possible to compare the structure of source and target schemas. As data integration seeks to populate the target relationships with data from the sources, the structure conflict detector must determine how the atomic target relationships are represented in the source schemas. In general, target relationships can correspond to arbitrarily complex source relationships, in particular to compositions. The composition operator particularly allows to treat the matching of target to source relationships as a graph search problem, as will be exemplified with the atomic target relationship records $\to$ artist from Figure 4.

First, the relationship's start and end node are matched to nodes in the source schema via the correspondences, in this case to albums and artist. Then, a path is sought between those nodes. In the example, there are two possible paths, namely albums $\to$ artist_list $\to$ id$'$ $\to$ artist_list$''$ $\to$ artist_credits $\to$ artist, and albums $\to$ id $\to$ album $\to$ songs $\to$ artist_list$'$ $\to$ id$'$ $\to$ artist_list$''$ $\to$ artist_credits $\to$ artist. To resolve this ambiguity, it is assumed that the most *concise* detected source relationship is the best match for the atomic target relationship. A relationship is more concise than another relationship, if its (inferred) cardinality κ_1 is more specific than the other relationship's cardinality κ_2, i.e., $\kappa_1 \subset \kappa_2$. In the case of equal cardinalities, the shorter relationship is preferred, according to the Occam's razor principle[5]. Here, both detected relationships have the same inferred cardinality $0..*$ according to Lemma 1, but the former is shorter and therefore selected as match.

Having matched a target relationship to a source relationship, comparing these two can finally reveal structural conflicts. The example target relationship records $\to$ artist has the annotated cardinality 1, but its corresponding source relationship is less concise, having an inferred cardinality of $0..*$. This lower conciseness causes a structural conflict: The target schema accepts only one artist value per record, while the source potentially offers an arbitrary amount of artists per album. To refine the statement about this violation, we can count the number of albums in the source data, that are associated to no or more than one artist, hence, determining the number of actually conflicting data elements. This *violation count* is applicable to any database constraint that can be expressed in CSG as listed above. Supporting more advanced constraints in CSGs like *conditional functional dependencies* [8] is left for future work.

Constraint in target schema	Violation count in source data
$\kappa(\rho_{\text{records}\to\text{artist}}) = 1$	503
$\kappa(\rho_{\text{artist}\to\text{records}}) = 1..*$	102

Table 3: Complexity report of the structure conflict detector.

The above described matching and checking process is performed for each target relationship. In the example scenario, there is only one more structural violation: artist $\to$ records has $0..*$ as inferred cardinality, so there may be artists with no albums. Afterwards, all collected structure violations, depicted in Table 3, are forwarded to the structure repair planner.

4.2 Structure Repair Planner

The structure repair planner proposes necessary cleaning tasks to cope with the structural violations in an integration scenario, that form the base for the following effort calculation. It ships with ten such cleaning tasks listed in Table 4; one per type of violation (e.g., of a not-null constraint) and expected result quality (low or high). The structure conflict detector can automatically select exactly those tasks that the integration practitioner has to perform in the data integration scenario to fix structural violations.

However, simply designating a task for each given violation is not sufficient, as data cleaning operations usually have side effects that can cause new violations. For instance, the structure conflict detector reveals that there are 102 artists in the source data that have no albums and can thus not be represented in the target schema.

[5]Among competing hypotheses, the one with the fewest assumptions should be selected.

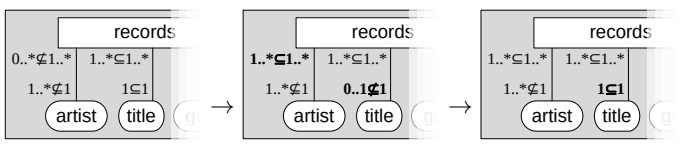
Constraint	Result quality	
	Low effort	High quality
Not null violated	Reject tuple	Add missing value
Unique violated	Set values to null	Aggregate tuples
Multiple attribute values	Keep any value	Merge values
Value w/o enclosing tuple	Drop value	Create enclosing tuple
FK violated	Delete dangling value	Add referenced value

Table 4: Structural conflicts and their corresponding cleaning tasks.

The high-quality solution is to apply the task *Create tuples for detached values*, that creates record tuples to store these artists, so that they do not have to be discarded. These new tuples would violate the not-null constraint on the title attribute, though, so subsequent cleaning tasks will be necessary. To account for such impacts, we simulate applied cleaning tasks on *virtual CSG instances* as exemplified in Figure 5. In addition to the prescribed cardinalities, the target CSG is annotated with *actual cardinalities*. In contrast to the prescribed cardinalities, those do not *prescribe* schema constraints but *describe* the state of the (conceptually) integrated source data – in terms of its relationships' cardinalities. Hence, the actual cardinalities are initialized with the inferred cardinalities from the source database. Figure 5a depicts this initial state. As long as there are actual cardinalities (on the left-hand side) that are not subsets of the prescribed ones, the CSG instance is invalid wrt. its constraints. Now, if the structure repair planner has chosen a cleaning task, e.g., adding new records tuples for artists without albums, its (side) effects are simulated by modifying the actual cardinalities, as shown in Figure 5b with bold print. So, amongst others the actual cardinality of artist $\rightarrow$ records is changed from $0..*$ to $1..*$, reflecting that all artists appear in a record after the task, and the cardinality of records $\rightarrow$ title is altered from 1 to $0..1$, stating that some records will then have no title. The latter forms a new constraint violation. Now, a successive repair task can be applied on this altered CSG instance, e.g., the task *Add missing values*, which leads to the state of Figure 5c.

This procedure of picking a task and simulating its effects is repeated until the virtual CSG instance contains no more violations. Furthermore, the structure repair planner orders the repair tasks, so that tasks that cause new structural violations (or might break an already fixed violation) precede the task that fixes this violation. This is not computationally expensive, because we need to order only tasks that affect a common relationship, but doing so allows for the detection of "infinite cleaning loops", where the execution order of cleaning tasks forms a cycle[6]. Additionally, the knowledge of the

[6]In most cases, these cycles are a consequence of contradicting



(a) Initial state. (b) State after *Add new tuples* for records. (c) State after *Add missing values* for title.

Figure 5: Extract of a virtual CSG instance as cleaning tasks are performed on it.

Task	Repetitions	Effort
Add tuples (records)	102	5 mins
Add missing values (title)	102	204 mins
Merge values (title)	503	15 mins
Total		224 mins

Table 5: High-quality structure repair tasks and their estimated effort using the effort calculation functions from Table 9.

necessary cleaning tasks in a data integration scenario, including their order, are a valuable aid that can positively impact the integration effort spent on coping with structural conflicts. Therefore, the ordered task list is provided to the user. Finally, the determined cleaning tasks are fed into the user-defined effort calculation functions, which automatically determine the effort for dealing with structural violations in the given scenario. Table 5 presents this effort for the example scenario.

5. VALUE HETEROGENEITIES

Value heterogeneities are a frequent class of data integration problems with a common factor: corresponding attributes in the source and target schema use different representations for their values. For instance, in Example 3.3 the target table tracks stores song durations as strings, whereas the source table songs stores these durations in milliseconds as integers. An integration practitioner might therefore want to convert or discard the source values to avoid having different value representations in the tracks.duration attribute. Thus, value heterogeneities can increase the integration effort.

This section presents a module in EFES to estimate the effort caused by value heterogeneities. The data complexity is computed by the *value fit detector*, which analyzes the source and target data to detect different types of value heterogeneities between them. These heterogeneities are then reported to the *value transformation planner*, the task model that proposes data cleaning tasks in response to the heterogeneity issues. Finally, the effort for the proposed tasks can be calculated.

5.1 Value Fit Detector

The basic approach of the value fit detector is to aggregate source and target data into statistics and compare these statistics to detect heterogeneities. Statistics are eligible for this evaluation, because they allow efficient comparison for large amounts of data, while enabling extensibility (as new functions can be added) and completeness (as issues that are not captured by available metadata can be discovered). Furthermore, statistics help to detect the especially meaningful, general data properties that characterize the data as a whole. In particular, if the source data does not match the observed or specified characteristics of the target dataset, plainly integrating this source data would impair the overall quality of the integration result: integration practitioners might want to spend effort to make source data consistent with the target data characteristics.

The value fit detector implements this idea as follows: Given an integration scenario, it processes all pairs of source and target attributes that are connected by a correspondence. For each such pair, statistic values of both attributes are calculated, with the target attribute's datatype designating which exact statistic types to use. In particular, we consider the following statistics:

repair tasks. EFES only proposes consistent repair strategies.

- The *fill status* counts the null values in an attribute and the values that cannot be cast to the target attribute's datatype.
- The *constancy* is the inverse of Shannon's information entropy and is useful to classify whether the values of an attribute come from a discrete domain [17].
- The *text pattern statistic* collects frequent patterns in a string attribute.
- *Character histogram* captures the relative occurrences of characters in a string attribute.
- The *string length statistic* determines the average string length and its standard deviation for a string attribute.
- Similarly, the *mean statistic* collects the mean value and standard deviation of a numeric attribute.
- The *histogram statistic* describes numeric attributes as histograms.
- *Value ranges* are used to determine the minimum and maximum value of a numeric attribute.
- For attributes with values from a discrete domain, the *top-k values statistic* identifies the most frequent values.

For Example 3.3, the string-typed duration target attribute designates the fill status, the text pattern statistic, the character histogram, the string length statistic, and the top-k values as interesting statistics to be collected.

In the next step, a decision model identifies, based on the gathered statistics values, the different types of value heterogeneities within the inspected attribute pair. Algorithm 1 outlines this decision model, which consists of a sequence of rules. The evaluation of each rule has its own, mostly simple, logic. The first rule ($substantiallyFewerSourceValues$), for instance, is evaluated by comparing the fill status statistics of the source and target attribute.

Algorithm 1: Detect value heterogeneities.

Data: source attribute statistics $\mathcal{S}_s$, target attribute statistics $\mathcal{S}_t$
Result: value heterogeneities V

if *substantiallyFewerSourceValues*$(\mathcal{S}_s, \mathcal{S}_t)$ **then**
  add *Too few source elements* to V;
if *hasIncompatibleValues*$(\mathcal{S}_s)$ **then**
  add *Different value representations (critical)* to V;
if *domainRestricted*$(\mathcal{S}_s) \wedge \neg$*domainRestricted*$(\mathcal{S}_t)$ **then**
  add *Too coarse-grained source values* to V;
else if $\neg$*domainRestricted*$(\mathcal{S}_s) \wedge$ *domainRestricted*$(\mathcal{S}_t)$ **then**
  add *Too fine-grained source values* to V;
else if *domainSpecificDifferences*$(\mathcal{S}_s, \mathcal{S}_t)$ **then**
  add *Different value representations* to V;

For the above example attribute pair, the fill-statuses are for both attributes near 100 %, there are no incompatible source values (integers can always be cast to strings), and neither of the attributes is domain-restricted. Still, possible domain-specific differences between them might be present. The evaluation of this last rule is more complex. For this purpose, a set of statistics, that are specific to the target attribute's datatype, are computed, e.g., the string format and string length statistic for the string-typed, not domain-restricted duration attribute. To compare these statistics among attributes, for each of them of type τ, an *importance score* $i\big(\mathcal{S}_t(\tau)\big)$ and a *fit value* $f\big(\mathcal{S}_s(\tau), \mathcal{S}_t(\tau)\big)$ are calculated. These calculations are specific to the actual statistics. Intentionally, the importance score describes how important the statistic type at hand is for the target attribute. For example, in the duration attribute, all values have the same text pattern *[number ":" number]*, so the string format statistic is presumably an important characteristic and will therefore have a high importance score. If it had many different text patterns in contrast, its importance would be close to 0. In addition, the fit value measures to what extent the source attribute statistics fit into the target attribute statistics. For instance, the length attribute provides only values with the differing pattern *[number]* leading to a low fit value. Eventually, the fit values for all applied statistic types are averaged using the importance scores as weights:

$$f \stackrel{def}{=} \sum_{\tau} \Big(i\big(\mathcal{S}_t(\tau)\big) \cdot f\big(\mathcal{S}_s(\tau), \mathcal{S}_t(\tau)\big) \Big)$$

This overall fit value tells to what extent the source attribute fulfills the most important characteristics of the target attribute. If it falls below a certain threshold, we assume domain-specific differences in between the compared attributes and Algorithm 1 issues an according value heterogeneity, e.g., *Different value representations* between the attributes length and duration. In experiments with importance scores and fit values between 0 and 1, we found 0.9 to be a good threshold to separate seamlessly integrating attribute pairs from those that had notably different characteristics.

The set of all value heterogeneities for all attribute pairs forms the complexity report of the value fit detector that can in the following be processed by the value transformation planner. Table 6 shows the complexity report for the example scenario. Note that the value heterogeneities can carry additional information that are derived from the attribute statistics as well and that can be useful to produce accurate estimates. These parameters are not further described in this paper.

Value heterogeneity	Additional parameters
Different value representation (length → duration)	274.523 source values, 260.923 distinct source values

Table 6: Complexity report of the value fit detector.

5.2 Value Transformation Planner

The value transformation planner proposes tasks to solve value heterogeneities as specified in Table 7. In contrast to the structure repair tasks from Section 4.2, those tasks do not have interdependencies. Therefore, the value transformation planner can simply propose an appropriate task for each given value heterogeneity based on the expected result quality of the data integration. For the four different types of value heterogeneities, there are only five different tasks, because for a low-effort integration result, value heterogeneities can in most cases be simply ignored. So, the *Different value representations* between the duration and length attributes might either be neglected (leading to no additional effort) or, for a high-quality integration result, the value fit detector will issue the task *Convert values*. This task is then again fed into the effort calculation functions that compute the effort that is necessary for the task completion. Table 8 illustrates the resulting effort estimate.

6. EXPERIMENTS

To show the viability of the general effort estimation architecture and its different models, we conducted experiments with real-world data from two different domains. In the following, we first

Value heterogeneity	Result quality	
	low effort	high quality
Too few elements	-	Add values
Different representations		
critical	Drop values	Convert values
uncritical	-	Convert values
Too specific	-	Generalize values
Too general	-	Refine values

Table 7: Value heterogeneities and corresponding cleaning tasks.

Task	Parameters	Effort
Convert values (length → duration)	274.523 values, 260.923 distinct values	15 mins
Total		15 mins

Table 8: Value transformation tasks and their estimated effort.

Task	Effort function (mins)
Aggregate values	$3 \cdot$ #repetitions
Convert values	(if #dist-vals < 120) 30, (else) $0.25 \cdot$ #dist-vals
Generalize values	$0.5 \cdot$ #dist-vals
Refine values	$0.5 \cdot$ #values
Drop values	10
Add values	$2 \cdot$ #values
Create enclosing tuples	10
Delete detached values	0
Reject tuples	5
Keep any value	5
Add tuples	5
Aggregate tuples	5
Delete dangling values	5
Add referenced values	5
Delete dangling tuples	5
Unlink all but one tuple	5
Write mapping	$3 \cdot$ #FKs $+ 3 \cdot$ #PKs $+$ #atts $+ 3 \cdot$ #tables

Table 9: Effort calculation functions used for the experiments.

introduce the system and its configuration. We then describe the scenarios and how we created the ground truth effort by manually integrating them. Finally, we compare our system to a baseline approach from the literature and to the measured effort in creating the ground truth.

6.1 Setup

The EFES prototype is a Java-based implementation of the effort estimation architecture presented in the paper, along with the three modules discussed. It offers multiple configuration options via an XML file and a command-line interface. As input, the prototype takes correspondences between a source and a target dataset, all stored in PostgreSQL databases. The prototype and the datasets are available for download.[7]

Configuration and External Factors. In our experiments, the only available software for conducting the integration are (i) manually written SQL queries, and (ii) a basic admin tool like pgAdmin. We also assume the user has not seen the datasets before and that she is familiar with SQL. Based on these external factors, corresponding effort conversion functions are reported in Table 9. For example, the intuition behind the formula for adding values is that it takes a practitioner two minutes to investigate and provide a single missing value. In contrast, deleting tuples with missing values requires five minutes, because independently from the number of violating tuples, one can write an appropriate SQL query to perform this task. Furthermore, we fine-tuned these settings for our experiments as we explain in Section 6.2.

Integration Scenarios. We considered two real-world case studies. The first is the well-known Amalgam dataset from the bibliographic domain, which comprises four schemas with between 5 and 27 relations, each with 3 to 16 attributes. The second is a new case study we created with a set of three datasets with discographic data. In those datasets, there are three schemas with between 2 and 56 relations and between 2 and 19 attributes each. Links to the original datasets can be found on the web page mentioned above.

For each case study, we created four integration scenarios, each consisting of a source and target database and manually created correspondences, because we do not want the evaluation results to depend on the quality of a schema matcher at this point. Within each domain, we included a data integration scenario with identical source and target schema and three other, randomly selected scenarios with different schemas.

Effort Estimates. In order to obtain effort estimations, we applied the following procedure to each data scenario twice, once striving for a low-effort integration result, once for a high-quality result. At first, we executed EFES on the scenario to obtain the data complexity reports and a set of initially proposed mapping and cleaning tasks. If a data complexity aspect was properly recognized but we preferred a different integration task, we have adapted the proposed tasks. For instance, in one scenario, our prototype proposed to provide missing *FreeDB IDs* for music CDs to obtain a high-quality result; this ID is calculated from the CD structure with a special algorithm. Since there was no way for us to obtain this value, we exchanged this proposal with *Reject tuples* to delete source CDs without such a disc ID instead.

Ground Truth Effort. Finally, we gathered the ground truth of necessary integration tasks manually and conducted them with SQL scripts and *pgAdmin*, thereby measuring the execution time of each task. We believe these two manually integrated scenarios with time annotations are a contribution per se, as they can be used also for benchmarking of mapping and cleaning systems for other data integration projects.

6.2 Experimental Results

The correspondences that were created for the case study datasets have been fed into EFES to compare its effort estimates to the actual effort. As a baseline, we used the standard approach based on attribute counting [14], as discussed in Section 2. To obtain fair calibrations of EFES and this baseline model, we employed cross validation: We used the effort measurements from the bibliographic domain to calibrate the parameters of EFES and the attribute counting approach for the estimation of the music domain scenarios, and vice versa. Thus, we have for both scenarios different training and test data and both models can be regarded as equally well-trained. To compare the two models against the mea-

[7] *http://hpi.de/naumann/projects/repeatability/data-integration/estimating-data-integration-and-cleaning-effort.html*

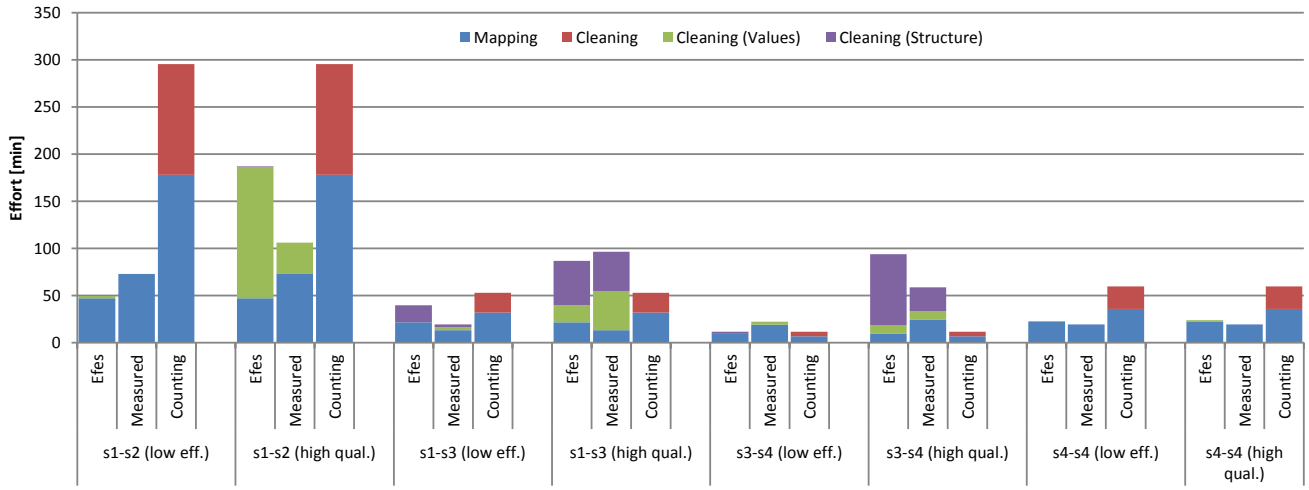


Figure 6: Effort estimates (Efes), actual effort (Measured), and baseline estimates (Counting) of the Bibliographic scenario.

sured effort, we applied the root-mean-square error (rmse):

$$\text{rmse} = \sqrt{\frac{\sum_{s \in S} \left(\frac{\text{measured}(s) - \text{estimated}(s)}{\text{measured}(s)} \right)^2}{\#\text{scenarios}}}$$

where S is a set of integration scenarios.

We start our analysis with the Amalgam dataset with the results reported in Figure 6. EFES consistently outperforms the counting approach in all scenarios. This is explained by the fact that the baseline has no concept of heterogeneity between values in the datasets, but it is one of the main complexity drivers in this integration scenario. In terms of the root-mean-square error, EFES achieves 0.47, while the baseline obtains 1.90 (lower values indicates better estimations), thus there is an improvement in the effort estimation by a factor of four. Moreover, EFES not only provides the total number of minutes, but also a detailed break down of where the effort is to be expected. This turned out to be particular useful to revise the effort estimate as described above, thus enriching the estimation process with further input. In fact, it makes a significant difference if an integration practitioner has to add hundreds of missing values or if tuples with missing values will be dropped. The baseline approach also distinguishes between mapping and cleaning efforts, but it relates them neither to integration problems nor actual tasks. The *s4-s4* scenario demonstrates this: source and target database have the same schema and similar data, so there are no heterogeneities to deal with. While we can detect this, the counting approach estimates considerable cleaning effort.

When we move to the music datasets, the results in Figure 7 show a smaller difference between the two estimation approaches. In fact, EFES outperforms the baseline four times, in three cases baseline does a better job, and in one case the estimate is basically the same. The explanation is that in this domain, there are fewer problems at the data level and the effort is dominated by the mapping, which strongly depends on the schema. However, when we look at the root-mean-square error, EFES achieves 1.05, while the baseline obtains 1.64. Therefore, even in cases where EFES cannot exploit all of its modules, and when counting should perform at its best, our systematic estimation is better.

It is important to consider the generality of the presented comparison. The two case studies are based on real-world data sets with different complexity and quality. When putting the results over the eight scenarios together, EFES achieves a root-mean-square error of 0.84, while the baseline obtains 1.70. In terms of execution time, EFES relies on simple SQL queries only for the analysis of the data and completes within seconds for databases with thousands of tuples. This overhead can be neglected in the context of the dominating integration cost.

7. CONCLUSIONS

We have tackled the problem of estimating the complexity and the effort for data integration scenarios. As data integration is composed of many different activities, we proposed a novel system, EFES, that integrates different ad-hoc estimation modules in a unified fashion. We have introduced three modules that take care of estimating the complexity and effort of (i) mapping activities, (ii) cleaning of structural conflicts arising because of different structures and integrity constraints, and (iii) resolving heterogeneity in integrated data, such as different formats. Experimental results show that our system outperforms the standard baseline up to a factor of four in terms of precision of the estimated effort time in minutes. When compared to the effective time required by an human to achieve integration, EFES provides a close estimate for most of the cases.

We believe that our work is only a first step in this challenging problem. One possible general direction is to integrate EFES with approaches that measure the benefit of the integration, such as the marginal gain [9]. This integration would allow the possibility to plot cost-benefit graphs for the integration: the more effort, the better the quality of the result.

A rather technical challenge in our system is to drop the assumption that correspondences among schemas are given. In practice, the effort for creating quality correspondences cannot be completely neglected – although, in our experience it takes considerably less time than other integration activities – and automatically generated correspondences introduce uncertainty wrt. the produced estimates. The accuracy measure as proposed Melnik et.al. [19] seems to be a good starting point to tackle this issue.

8. REFERENCES

[1] B. Alexe, W.-C. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *Proceedings of*

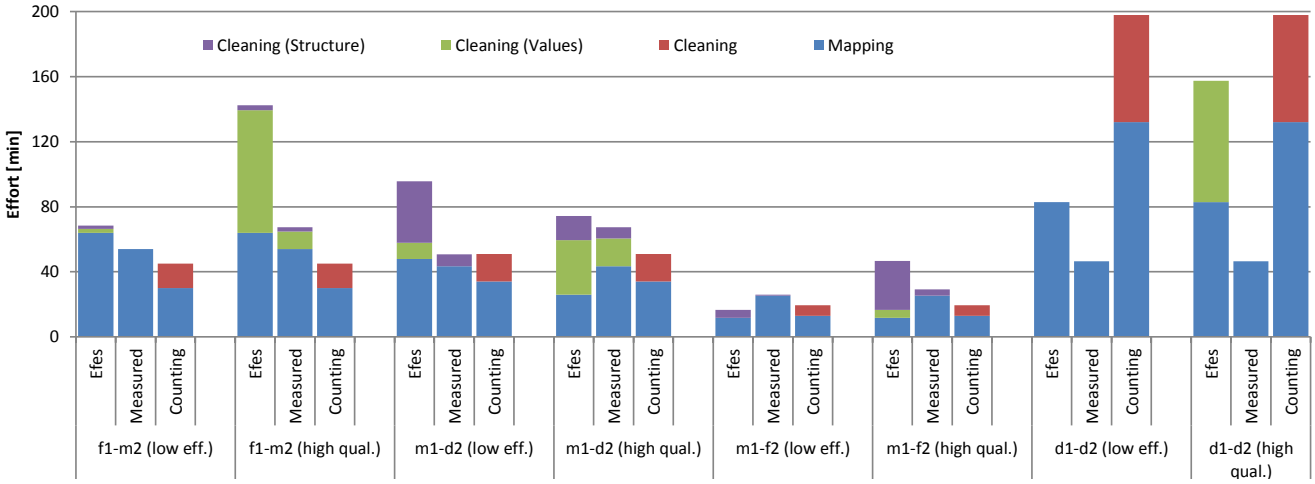


Figure 7: Effort estimates (Efes), actual effort (Measured), and baseline estimates (Counting) of the Music scenario.

the VLDB Endowment, 1(1):230–244, Aug. 2008.

[2] P. Atzeni, G. Gianforme, and P. Cappellari. A universal metamodel and its dictionary. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems I*, pages 38–62. Springer, 2009.

[3] B. Boehm. *Software Engineering Economics*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[4] B. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece. *Software Cost Estimation with COCOMO II*. Prentice-Hall, Englewood Cliffs, NJ, 2000.

[5] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Data integration under integrity constraints. *Information Systems*, 29(2):147–163, 2004.

[6] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 77–86, 2009.

[7] M. P. Consens, I. F. Cruz, and A. O. Mendelzon. Visualizing queries and querying visualizations. *SIGMOD Record*, 21(1):39–46, 1992.

[8] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. Ilyas, M. Ouzzani, and N. Tang. Towards a commodity data cleaning system. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 541–552, 2013.

[9] X. L. Dong, B. Saha, and D. Srivastava. Less is more: Selecting sources wisely for integration. *Proceedings of the VLDB Endowment*, 6(2):37–48, 2012.

[10] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer-Verlag, Heidelberg (DE), 2nd edition, 2013.

[11] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 207–224, Siena, Italy, 2003.

[12] R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan:. Composing schema mappings: Second-order dependencies to the rescue. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 83–94, Paris, France, 2004.

[13] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. Mapping and Cleaning. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 232–243, 2014.

[14] B. Harden. Estimating extract, transform, and load (ETL) projects. Technical report, Project Management Institute, 2010.

[15] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, 2011.

[16] A. Kumar P, S. Narayanan, and V. M. Siddaiah. COTS integrations: Effort estimation best practices. In *Computer Software and Applications Conference Workshops*, 2010.

[17] D. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[18] B. Marnette, G. Mecca, P. Papotti, S. Raunich, and D. Santoro. ++Spicy: an opensource tool for second-generation schema mapping and data exchange. *Proceedings of the VLDB Endowment*, 4(12):1438–1441, 2011.

[19] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 117–128, 2002.

[20] F. Naumann. Data profiling revisited. *SIGMOD Record*, 42(4):40–49, 2013.

[21] P. Papotti and R. Torlone. Schema exchange: Generic mappings for transforming data and metadata. *Data & Knowledge Engineering (DKE)*, 68(7):665–682, 2009.

[22] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

[23] T. Rekatsinas, X. L. Dong, L. Getoor, and D. Srivastava. Finding quality in quantity: The challenge of discovering valuable sources for integration. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2015.

[24] K. P. Smith, M. Morse, P. Mork, M. H. Li, A. Rosenthal, M. D. Allen, and L. Seligman. The role of schema matching in large enterprises. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2009.

[25] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. CrowdER: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11):1483–1494, 2012.