Maik Thiele, Tim Kiefer, Wolfgang Lehner

**Cardinality estimation in ETL processes**

*Erstveröffentlichung in / First published in:*

*Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP.*

DOI: [https://doi.org/10.1145/1651291.1651302](https://doi.org/10.1145/1651291.1651302)

Diese Version ist verfügbar / This version is available on:

[https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-767565](https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-767565)
ABSTRACT

The cardinality estimation in ETL processes is particularly difficult. Aside from the well-known SQL operators, which are also used in ETL processes, there are a variety of operators without exact counterparts in the relational world. In addition to those, we find operators that support very specific data integration aspects. For such operators, there are no well-examined statistic approaches for cardinality estimations. Therefore, we propose a black-box approach and estimate the cardinality using a set of statistic models for each operator. We discuss different model granularities and develop an adaptive cardinality estimation framework for ETL processes. We map the abstract model operators to specific statistic learning approaches (regression, decision trees, support vector machines, etc.) and evaluate our cardinality estimations in an extensive experimental study.

Categories and Subject Descriptors
H.2.4 [Database Management]: Systems; H.2.4 [Database Management]: Database Administration—Data warehouse and repository

General Terms
Algorithms, Management, Performance

Keywords
Cardinality Estimation, ETL, Real-Time Data Warehouse

1. INTRODUCTION

Data consolidation in data warehouse systems is performed with the help of so-called ETL processes, i.e., extraction, transformation, and loading. During the extraction stage, data from heterogeneous sources is brought into the data warehouse’s base data area, and during the loading phase, it is loaded into the data warehouse. The actual key element of any ETL process, however, is the transformation phase, during which data is cleansed, checked, and transformed into a common schema. The operators used during transformation can be of very diverse nature, e.g., classic relational operators, pivot, normalizer, regex evaluations, validity checks (zip code, credit card number, etc.), and there may even be manual data cleansing and transformation based on domain knowledge.

We know a variety of techniques and heuristics for cardinality estimation from the field of relational query optimization. However, their possible application to ETL processes is limited. On the one hand, there exist many ETL operators without any relational counterpart (e.g., unfold or pivot). On the other hand, operators in ETL processes can be defined freely with the help of various host languages. But there is another difference between query trees and ETL processes: while a query is often only executed once (or only a few times) in database systems, ETL processes have a much longer life cycle, comparable to standing processes. The long life span and the hardly dynamical behavior of ETL processes justify increased efforts for cardinality estimation when compared to classical query optimization. Instead of using the often histogram-based approaches from the database field, it is therefore beneficial to employ complex learning algorithms. The model training can be performed asynchronously with the data transformation, so the actual data processing is not interrupted. The long life cycle of ETL processes also allows the use of a self-correcting feedback loop, which means that the estimations’ quality is continuously monitored and estimators may be adjusted appropriately.

Due to the above mentioned heterogeneity of ETL operators, individual cardinality estimation models for each operator can only be developed with intensive efforts. Thus, we propose a black-box approach in this paper that does not require any a-priori knowledge of the operator logic or assumptions on the data distribution. Each operator is abstracted in terms of its input and one output cardinality, independent from its internal structure, which then serve as input parameters for respective estimators. Thereby, it becomes possible to map even those operators that we do not know in detail. For this purpose, we propose a framework that comes with a set of cardinality estimators for each individual ETL operator. The best of these estimators will then be selected during the ETL process’ runtime. Aside from this generic approach, it is also possible to map certain operators to certain estimator models during the design time of the ETL process.
Contributions. In detail, our main contribution comprises the following:

- We discuss application scenarios for cardinality estimation in ETL graphs.
- We develop a cardinality estimation framework that allows to estimate the cardinality of any arbitrary operator in any sort of ETL graph.
- We integrate our approach into the Pentaho Data Integration project.
- We define two ETL scenarios, which we use to conduct extensive experiments to test the estimation quality as well as the memory and runtime behavior.

Structure of the paper. The paper is organized as follows. Section 2 provides a review of related work and discusses application areas that would benefit from ETL cardinality estimations. We outline our ETL and estimator model in Section 3. In Section 4, we describe our evaluation setup and the model quality metric, and we discuss the experimental results. Finally, we conclude in Section 5.

2. RELATED WORK

The challenges addressed in this paper can be grouped into three categories: 1) the application scenarios that arise from cardinality estimations in ETL graphs, 2) the abstraction and formalization of ETL processes, and 3) the analytical models needed in order to perform cardinality estimation.

2.1 Applications

Knowledge of the input and output qualities of ETL operators is fundamental for the optimization of ETL processes. It allows the optimal re-ordering of ETL operators, an appropriate selection of physical operators, and the physical deployment (e.g., parallelization of operators). The issue of restructuring ETL processes on a logical level was addressed in [10, 15]. In [16], sort operators were inserted into the physical representation of an ETL process followed by the respective plan operators; this clearly increased the performance. Black-box operators were evaluated with the help of a simple cost function. A continuous estimation of the cardinalities at runtime allows online optimizations in the sense of the continuous adjustment of ETL processes to the data. This is very similar to continuous queries over data streams [1], with their on-the-fly optimization of plans. The difference can be found in the operators: in the data-stream field, they bear a much stronger resemblance to classic relational operators.

Another application area is represented by update scheduling in the real-time data warehouse field. The real-time aspect in the context of DWHs describes a new processing model where every change is automatically captured and pushed into the DWH. Thus, the data in a real-time DWH is subject to continuous changes, denoted as a trickle-feed of updates. This induces two options from the user’s point of view: 1) outdated or slightly outdated data may be used in order to get faster query results, or 2) only the most current data shall be used, i.e., all modifications are committed before the next query is executed. Given that users specify their requirements for each query, we can exploit this information and build a scheduler that controls the continuous update flow [14, 13]. In order to predict both the parts of the data warehouse that will be affected by certain updates and the degree of the update’s impact, fast predictions of the cardinalities are required.

It seems to be clear that the cardinality estimation process must be fast enough to be appropriate for the applications described here. The baseline for comparisons is the execution speed for the ETL process to be estimated itself. Thus, the estimation of the cardinalities should be significantly faster than the execution of the ETL process.

2.2 Formalizing ETL Processes

A variety of approaches for the formalization of ETL operators and ETL processes, respectively, exist: Vassiliadis et al. [17, 16], Joerges et al. [6], AJAX [4], Potter’s Wheel [9] are among the most important ones. We use these models but abstract them to a degree that allows us to employ them for our black-box approach. Additionally, we extend the formal description of ETL processes by the aspect of estimation models (see 3.2).

2.3 Cardinality Estimation

In the relational field, there are a variety of works [3, 5, 8] that are concerned with the cardinality estimation for relational operators. These may be used for the respective counterparts in the ETL field. Our rather general model allows the use of different estimators. Also, the idea of a self-tuning feedback loop for the adjustment of the optimizer estimations was discussed in the relational field [12, 2]. However, these approaches are based on histograms, and hence, they cannot be applied to arbitrary queries or ETL operators, respectively.

The authors in [7] propose a black-box approach to query cardinality estimation. For this purpose, queries are grouped into syntactically similar groups, whose parameters and operators constitute the input for the estimators. We follow a similar yet more complex approach, since we want to estimate the cardinality of many operators within one process. This leads to the additional problem of the propagation of uncertainty.

3. CARDINALITY ESTIMATION MODELS FOR ETL PROCESSES

In this section, we first want to formally describe ETL operators and ETL processes; based on this, we then develop a framework that supports cardinality estimation in ETL processes. For this purpose, we propose a mapping of individual ETL operators to sets of estimation models. The structure of these sets depends on the complexity of the operator to be estimated as well as on its input and output data flows. Finally, we also discuss different model granularities and optimizations based on them.

3.1 ETL Operators and Processes

For the cardinality analysis of ETL processes, we want to restrict our considerations to the fundamental process parameters. An operator or a so-called activity is a tuple $a_j = (I_j, O_j)$, where $I_j$ and $O_j$ represent the set of input and output data flows (see Figure 1). A directed edge between two activities $a_j$ and $a_k$ denotes that $a_k$ receives data from $a_j$. The order of the activities within the graph is given by their temporal order. It holds that $a_j \leq a_k$ if a direct
data flow exists between the two activities, \( \exists o \in O_i \) and \( \exists i \in I_k \) with \( o = i \). If there is such a data flow between two activities, we denote \( a_j \) as the provider and \( a_k \) as the consumer. Aside from the activities, there is also the set of data sources DS that provide the data to be processed by the ETL process. For our further analysis, we abstract the data source to data providers for the ETL activities; these providers must have a specific output cardinality, i.e., a certain number of tuples that must be processed by the ETL process.

We consider an ETL process as a data-flow description that can be written in the form of a directed acyclic graph (DAG), with activities and data sources as vertices and data flows as edges. This simple description of the data flow constitutes the basis for the cardinality estimations.

### 3.2 Cardinality Estimation Models

The semantics of ETL operators may vary strongly, which must be reflected in the cardinality estimation models to be used. The cardinalities of some operators can be estimated very well with simple regression models, but other operators require more complex models. There is a large variety of possible models for cardinality estimation. In addition to different classes, such as training or statistics-based approaches, we find diverse algorithms within each such class. These approaches differ in their complexity, speed, and in their estimation’s quality. When we consider ETL operators as black boxes, there is no a-priori answer to the question for the best suited estimator model. Depending on the application, the objectives may alternate between either very fast or very exact model estimations. For this reason, we link each operator \( a_j \) to a set of different estimator models \( M_j \) (see Figure 1). By maintaining a set of models instead of a single model, our approach is very flexible. Depending on the complexity of the respective ETL operator, the model set may consist of either a simple regression model or a variety of complex models that all contribute to the cardinality estimation (via appropriate weightings). At runtime, the best models \( m \) from a model set will be selected. This selection should be realized with the help of a feedback loop; we will look at this in more detail later on.

### 3.3 In- and Outbound Cardinality

Another aspect to be considered is the number of output data flows \( |O_j| \) of an operator \( a_j \). Most algorithms are only able to estimate one model parameter from a number of input parameters (n-1 models). Similarly, estimation ap-

![Figure 1: General operator and estimator model](https://example.com/figure1.png)

![Figure 2: Comparison of a raw and a fine-grained modelling approaches for relational operators, which may be based on histograms, for example, build on the principle of mapping several input parameters to one output parameter (see [3, 5, 8]). Even though a variety of operators can be described with n-1 models, these are insufficient to cover all ETL operators. One example for such an activity is a filter operator that splits up the tuple stream according to a filter predicate. Due to the above mentioned limitations, it seems logical to build an n-m model from multiple n-1 models. Instead of having one new model that is more complex to describe, we thereby receive a set of simple models that are independent of each other. In formal terms, this means: If an operator \( a_j \) has more than one output data flow, one model set \( M_j^k \) will be created for each of those (see Figure 1). The input data flows are the same for each of the n-1 models.

### 3.4 Model Granularity

There are two fundamental approaches for the cardinality estimation of ETL processes. On the one hand, it is possible to consider the whole ETL process in one model. On the other hand, each single operator of the ETL process may be individually mapped to a model (see Figure 2).

**One Model per ETL Process.** In an extreme case, all input data flows are input values for one large model. The output cardinality (or, in the more general case, the set of output cardinalities) represents the model measure to be computed. The advantage of such an approach is that only one model has to be designed and maintained. The most significant disadvantage is, that if individual ETL operators that deliver quite unpredictable results are mapped within large models, the whole estimation may become worse. Furthermore, none of the cardinalities within the ETL process are accessible, which makes this approach not applicable for the optimization of individual operators (see Section 2.1).

**One Model per ETL Operator.** The opposite case to the mapping above is the one model per ETL operator approach. It comes with the advantage that most ETL operators are comparatively simple, which allows the usage of simple and often linear models. For specific operators or operator classes, we can use particularly suited estimators. In comparison to the one model per ETL process approach, the intermediate
cardinalities of individual operators are available as well. A disadvantage, however, is certainly found in the considerably higher memory and maintenance efforts for the models, especially for large ETL processes with a high number and variety of operators.

It is not as simple to give comparative statements on the cardinality estimation error for both approaches. The very complex model from the one model per ETL process approach can be very sensitive when faced with parameter modifications. On the other hand, the one model per ETL operator approach comes with the drawback that small errors at the beginning of the ETL process may be transitively forwarded and increased.

A major difference between both approaches is found in the adaptivity concerning changes of the ETL process, i.e., the insertion, deletion or update of ETL operators. For the one model per ETL process approach, the model has to be discarded and trained anew after every modification. In contrast, the one model per ETL operator approach only requires the invalidation and re-generation of the model for the affected ETL operator.

Hybrid ETL Operator Model. In order to combine the advantages of both approaches, we choose a hybrid ETL operator model. In its basic configuration, each operator has its own model, which can be combined for optimization purposes. This is beneficial, especially for operators that affect the cardinality either in very predictable fashion or not at all, e.g., sort or projection. For this type of operators, it would be redundant to maintain individual models and they would be estimated together with adjacent operators. Due to the resulting reduction in the number of models, the memory and computation consumption decreases. Complex operators that tend to deliver bad estimation results remain isolated, which ensures that very good partial results can still be used further on.

The requirements that need to be fulfilled (e.g., model compatibility) in order to merge models will be discussed in the next section.

### 3.5 Merge of Estimation Models

The described hybrid approach for the model generation over sets of operators requires that several models can be merged into larger models. In a brief discussion, we want to focus on the semantics for merging two models. The pre-requisites under which it is possible to merge two estimator models strongly depend on the used estimator models. However, the fundamental idea of merging two models $a_i$ and $a_{i+1}$ is based on the concept of mapping the ETL operators of both models within one common model that covers both operators. Since the merging of two models results in a new model, this one will have to be trained as well in order to return good estimations. This process 1) may be part of the merging step or 2) can happen at a later point in time during the runtime of the ETL operators to be estimated. In case of the former, the training data, i.e., the input and output cardinalities of previous executions, must still be available. For this purpose, the respective cardinalities need to be stored. Therefore, we recommend a sliding-window approach, which only holds the most current cardinalities. The training data will also have to be merged but under consideration of the reduced number of model parameters. In case of the latter, new training data have to be collected at runtime and then serve to train the newly created model. When doing so, we have to keep in mind that the estimations of the new model will only become reliable enough after a sufficiently long training phase.

---

**Figure 3: ETL cardinality monitor and control**
3.6 ETL Cardinality Monitor and Control Process

So far, we have considered the individual aspects of cardinality estimation in ETL processes in isolation; we now want to summarize them within one architecture. Figure 3 provides a schematic illustration of this architecture. On the left, you find the ETL process to be monitored, and on the right, you see the component that contains the ETL monitor and the controller. This component returns the estimated cardinalities to the respective applications and controls the cardinality estimation’s quality. This is done by comparing the estimated and the actual cardinalities for the individual models of an operator. Since the models have not yet been trained when the ETL process starts, the estimates will only be propagated to the application once the error is statistically bound (ε is given by the application). The error will also be checked continuously at later stages, and in case of too heavy deviations, a refreshing of the models will be triggered. When validating the error, we have to keep in mind that the continuous propagation of uncertainty also increases the error, i.e., estimated and actual cardinalities will drift further apart for operators that appear later in the ETL process.

There are several possibilities to improve the estimation quality: 1) a model refresh by discarding old training data and adding new ones, 2) the addition of new models that are more appropriate for the operator to be estimated, and 3) the adjustment of the model weightings. The improvement of the estimation quality is closely tied to the optimization of the estimation process with regard to runtime and memory complexity. If models return bad estimations, they can be pruned, which will speed up the estimation process. When merging models, different objectives have to be compared with one another: the loss of estimation quality versus the runtime improvement. A cost model can be used here, which would weigh the different strategies, e.g., merge of Mj with Mj+1 or Mj−1.

4. EVALUATION

Our presented framework for cardinality estimation in ETL graphs offers a variety of starting points for evaluations. Due to the limited space here, we want to focus on the general applicability of our approach using two example scenarios. In detail, we conducted an experimental study to evaluate 1) the cardinality estimation’s quality in different ETL graphs and 2) the runtime and space overhead for the ETL cardin-ality estimation and for specific estimator models.

4.1 Experimental Setup

For the implementation of our ETL cardinality estimation, we used the open-source ETL platform Pentaho Data Integration1 (formerly Kettle) and extended it with our approach. The required estimators are given by the WEKA Project2, which provides a collection of various classification, clustering and regression algorithms with consistent interfaces. Further information on WEKA and the algorithms implemented there can be found at [18]. In the following, we will present the example scenarios designed for our evaluation as well as the used estimator models and the evaluation metrics.

4.1.1 ETL Example Graphs

The basis for our experimental evaluation is given by two transformations that include several interesting ETL operators. Figure 4 shows a graphical representation of these transformations via a screenshot of Pentaho GUI, which was used to generate these transformations. Each of the small squares represents one ETL operator; the identifiers for the individual steps are given below. The gray boxes around the ETL operators are our extension and symbolize estimator models. All operators that are grouped together within one contiguous gray box are estimated with one common model. We will explain the operators and their respective meaning in the following.

As input relations for our ETL transformations, we use the tables of the TCP-H schema. We can fill it with various data for the purpose of training the estimator models and for subsequent analyses. Initially, we look at the ETL graph that processes the table NATION_REGION (a join of the two original tables) (see Figure 4 below). In the first step, this table is read and then processed by the Row Denormaliser On NR_TYPE operator. This operator separates the columns N_NAME, N_COMMENT and N_REGIONKEY of the input schema and splits it into the tables REGION and NA-TION. The respective estimator model for this operator is denoted as Model 1. Since it internally contains two n-1 models, we further distinguish between Models 1a and 1b. This operator is particularly interesting for analyses because it does not have any counterpart in the relational world. The other two steps serve to reconstruct the REGION relation. For its reconstruction, we have to remove all duplicates that might have been created by the join during the preprocessing. The respective model for these operators is quite simple and therefore not part of our evaluation.

The second ETL process treats data from the table CUSTOMER_SUPPLIER, which originates from a UNION between the CUSTOMER and the SUPPLIER table. In order to differentiate the respective tuples, we added the attribute CS_TYPE. The operator Filter Rows On CS_TYPE=1 splits the table into customer and supplier data. Since this operator has two output data flows as well, we require two n-1 models, 2a and 2b. We assume that the linear dependency between the input relation’s cardinality and the set of written tuples can be mapped efficiently. We do not consider further the tuples of the SUPPLIER table, instead we focus on the CUSTOMER data.

The next operators, Java Script Delete Phone Numbers and Filter Rows On CS_Phone is not null, feature the semantics that randomly selected phone numbers will be replaced by NULL and the respective tuples will be removed (Model 3). Poor estimations are an inevitable consequence of this operator group’s indetermination. No model will be able to define dependencies between the input and output cardinalities.

In our last operator, we use the attribute CS_Orderkey and perform a lookup to determine the number of orders for a given customer (Stream Lookup Or-derkeys). In case we do not find any orders, the data set will be removed from the stream (Filter Rows On CS_Ordercount is not null). The respective model for these two operators is denoted as Model 4.

1http://kettle.pentaho.org/
2http://www.cs.waikato.ac.nz/ml/weka/
Figure 4: ETL example scenario

4.1.2 Estimator Models

For our evaluation, we use three different WEKA models, which have already delivered good estimations according to our experience. The employed estimators (all of which are found in the package weka.classifiers.functions) will be listed and briefly described in the following itemization:

- **Linear Regression**: this estimator model maps affine-linear input values to an output value.
- **Multilayer Perceptron**: this one is based on neuronal networks and backpropagation. On various layers, the algorithm attempts to split the result space with hyperplanes.
- **SMOreg**: this estimator implements the algorithm for sequential minimal optimization [11]. It is trained with a support-vector regression model.

Before these models can be used, they require a certain amount of training. Therefore, we used the TCP-H data generator as basis and developed a component called TD-Generator. This component generates training data with different scale factors from a Zipfian distribution, with varying Zipf value, in order to have a realistic scenario. In all subsequent experiments, we initially trained the models with 500 training data sets.

4.1.3 Model Quality Metrics

We use three quality criteria to evaluate the precision of our models: The first one is the root mean-squared error

\[
RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (p_i - a_i)^2}.
\]

Here, \(a_i\) stands for the actual cardinalities, and \(p_i\) denotes the estimated cardinalities. Variable \(n\) represents the number of data records. The second criterion is the root relative squared error, which is calculated as follows:

\[
RRSE = \sqrt{\frac{\sum_{i=1}^{n} (p_i - a_i)^2}{\sum_{i=1}^{n} (\bar{a} - a_i)^2}}, \quad \text{with} \quad \bar{a} = \frac{1}{n} \sum_{i=1}^{n} a_i.
\]

To a certain degree, this measure indicates the improvement of the respective model compared to the simplest possible model, i.e., the calculation of the mean value for all previous results. In any case, the interpretation of this measure must take the underlying data and models into account. The third criterion is the correlation coefficient, with the calculation formula:

\[
\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y},
\]

where \(\text{cov}\) denotes the covariance and \(\sigma_X\) and \(\sigma_Y\) stand for the standard deviations. The correlation coefficient takes a value between -1 and +1, and it represents the statistic correlation between the actual values and the estimated ones. A value of 1 for the correlation coefficient denotes perfect correlation, whereas a value close to 0 signifies that there is no correlation at all.

4.2 Quality of Cardinality Estimations

In our first experiment, we want to analyze the estimation’s quality for the estimator models introduced in Section 4.1.1. For this purpose, we initially look at the error values \(RMSE\) and \(RRSE\). Figures 5a and 5b list them for the six models, 1a to 4, and the three estimators, LinearRegression, MultilayerPerceptron and SMOreg. As can be seen, the models for the various ETL operators produce quite different error values. In contrast, the estimation’s quality for the individual estimators is quite similar, and the Linear Regression model even performs best.

Model 1 (both 1a and 1b) turns out to be very reliable in terms of estimated cardinalities. The mean squared error is only approx. 1 tuple when using the best model, which should suffice in most cases. If we simultaneously consider the other two variables as well, we will find confirmation of the model’s quality. In both cases, the correlation coefficient is close to 1 (see Table 1), which emphasizes the dependency between the cardinalities. With ca. 20%, the values for the root relative squared error are rather high, but the reason lies in the data structure. The cardinality fluctuates only slightly and the number of written tuples is often close to the median of all possible values. For this reason, the calculation of the median also constitutes a decent model. Hence, Model 1 is a confirmation of our approach to use machine
learning algorithms for estimations. It shows that we can also estimate ETL operators that do not have any relational counterparts and that therefore have not received significant attention in the research field of cardinality estimation so far.

Model 2 is as reliable concerning all three evaluation parameters. A correlation coefficient of 1 and low error values confirm our assumption. The linear character of the operators is sufficiently mapped by all three classifiers. Only the estimator model MultilayerPerceptron deviates with its prediction in Model 2b by 39 tuples on average. However, if we consider the cardinality of the whole data set, which is 7,500 for the highest scaling, we can determine that an error of 39 tuples is truly negligible.

Model 3 is the only model listed here that consistently returns high error values. This observation matches our previously made hypothesis that the random removal of tuples and the resulting high fluctuation in the cardinality cannot be mapped satisfactorily. We calculated an average deviation of more than 1,000 tuples, which is not acceptable in most applications. The model type and the correlation coefficients clarify, though, that any scaling of the data will also scale this error to the same extent.

In contrast to this, Model 4 proves to be very reliable again; RMSE is 6.7% and RRSE is 0.64%, the correlation coefficient is 1 (all values are for the linear model). The quality of the model for the lookup operator depends on the data basis of the lookup. If this data is characterized by larger fluctuations or changing distributions, we would have to expect a less qualitative model.

### 4.3 Runtime and Space Utilization

To analyze the runtime, we use the example ETL scenarios introduced in Section 4.1.1. For our purposes here, we executed the ETL processes as well as the respective cardinality estimation models and compared their runtimes using different data volumes from 5 MB to 3 GB. During our experiments, each operator was associated to three concrete estimator models (see Section 4.1.2). Figure 6 shows the results of our conducted measurements. The x-axis of the plot represents the data set’s size in Megabyte. The y-axis gives the associated execution time for the ETL process as well as the time for the estimation (in seconds). As we can see, the estimations are, on average, executed 10-15 times faster than the process. However, the differences for small data volumes are only marginal, e.g., for 10 MB, we find 1.53 seconds for the process execution and 1.21 seconds for the estimator execution. In practice, this difference would be much more significant though, in favor of the estimator.

There are several reasons: On the one hand, the runtime advantage increases for larger ETL processes as well as for more complex individual operators. While the operators considered as black boxes may be of arbitrary complexity, we always have a-priori knowledge of an estimator’s complexity, e.g., O(n) for the linear regression. Furthermore, there are also blocking operators, which have to wait for other data or which may have to rely on human interaction. Since there will never be any blockings during cardinality estimation, the estimation takes much less time. An ETL process’ runtime is likely to get even worse in comparison to the estimation when write operations come into play, especially in case of lock contention.

In contrast to the runtime analyses, the analysis of the storage requirements can be done analytically. Two factors are important here: the used estimation model on the one hand and the amount of buffered training data on the other hand. Regression functions have negligible space requirements. For more complex models, such as a neuronal network, the space requirements depend on the desired degree of accuracy and length of training time. In any case, the size of the models will not exceed the three-digit kB range. If the ability to quickly train new models is desired, for example, after a merge, the training data - i.e., the input and output cardinalities - have to be stored additionally. However, these are only a few dozen or a few hundred integer values, and hence, their storage does not pose any problems.

### 5. CONCLUSION

In this paper, we presented a black-box approach for the cardinality estimation of arbitrary ETL processes. We introduced applications, e.g., the ETL process optimization as well as the update scheduling in real-time data warehouses,
that would profit from cardinality estimation. Our approach is of generic nature, which means the estimation can be implemented with various models. Depending on the application, we can use generic estimators or - if knowledge on the operators is available - specialized algorithms. In order to examine the feasibility of our approach, we developed a prototypical implementation based on the Pentaho Data Integration project and the WEKA project. The evaluation with the help of two example ETL processes confirmed the quality of our approach. Both ETL operators with relational counterparts and operators without relational equivalents could be estimated with satisfying accuracy. However, there are also operators that make estimations very difficult or even impossible (e.g., random deletes). These exceptions have to be detected and handled appropriately.

In our previous evaluation, we did not consider the propagation of uncertainty. Unfortunately, this can turn into a problem, particularly for complex ETL processes. Thus, we want to address this issue in further research activities, quantify the error, and define possible solutions. Furthermore, we want to employ learning algorithms in order to combine ETL operators in operator groups. Operators within the same group are marked by similar behavior with regard to their impact on the data cardinality and can thus be estimated efficiently with the same estimator models. New ETL operators can then be assigned to suitable estimator models by using a group’s similarity measure.

6. REFERENCES