J Data Semant (2012) 1:75–97
DOI 10.1007/s13740-012-0006-9

ORIGINAL ARTICLE

Metrics for the Prediction of Evolution Impact in ETL Ecosystems: A Case Study

George Papastefanatos · Panos Vassiliadis · Alkis Simitsis · Yannis Vassiliou

Received: 19 November 2011 / Revised: 30 April 2012 / Accepted: 16 May 2012 / Published online: 4 July 2012

Abstract The Extract-Transform-Load (ETL) flows are essential for the success of a data warehouse and the business intelligence and decision support mechanisms that are attached to it. During both the ETL design phase and the entire ETL lifecycle, the ETL architect needs to design and improve an ETL design in a way that satisfies both performance and correctness guarantees and often, she has to choose among various alternative designs. In this paper, we focus on ways to predict the maintenance effort of ETL workflows and we explore techniques for assessing the quality of ETL designs under the prism of evolution. We focus on a set of graph-theoretic metrics for the prediction of evolution impact and we investigate their fit into real-world ETL scenarios. We present our experimental findings and describe the lessons we learned working on real-world cases.

Work conducted in the context of the "EICOS: foundations for personalized Cooperative Information Ecosystems" project of the "Thales" Programme. The only source of funding for this research comes from the European Social Fund (ESF)-European Union (EU) and National Resources of the Greek State under the Operational Programme "Education and Lifelong Learning (EdLL)".

G. Papastefanatos
Institute for the Management of Information Systems, Athens, Greece
e-mail: gpapas@imis.athena-innovation.gr

P. Vassiliadis
University of Ioannina, Ioannina, Greece
e-mail: pvassil@cs.uoi.gr

A. Simitsis (✉)
HP Labs, Palo Alto, CA, USA
e-mail: alkis@hp.com

Y. Vassiliou
National Technical University of Athens, Athens, Greece
e-mail: yv@cs.ntua.gr

Keywords Database evolution · Database metrics · Evolution case study

1 Introduction

Having accurate and up-to-date data warehouses is essential for Business Intelligence and Decision Support. A data warehouses design, apart from performance guarantees, should also provide correctness guarantees. Every time an evolution event occurs anywhere in the warehouse environment (e.g., a design change at the operational sources) it should be smoothly absorbed without causing any further inconvenience. For achieving this, the warehouse and its counterparts should be easily maintainable and the process of populating it should not be destructed by evolution events.

The Extract-Transform-Load (ETL) flows constitute the backbone of a typical data warehouse architecture. Most of the research for improving ETL designs has focused solely on improving performance. However, based on practical experience, maintenance makes up for up to 60 % of the resources spent in a warehouse project [34], and therefore, maintainability is an important factor for the determination of the quality of a design [19,36]. Although practitioners are well aware of this problem, still, we miss a formal and concrete answer to fundamental questions like "How good is an ETL design?" and "What makes an ETL design good or bad?". Typically, such questions are answered by a set of empirical rules based on practical observations of the past, as well as rules of thumb that have been established by expert practitioners and despite their value, they simply transfer the lessons learned the hard way in the "craft" of ETL design. Most of these rules consider only structural properties of the ETL flow or constructs internal to the underlying databases and do not take into account neither the incorporation of constructs surrounding the databases, nor the fact that a software construct, and especially an information system, evolves over time.

In practice, the problem is hard since changes in the schema of database-centric systems affect not only both its internals but also the surrounding deployed applications. Hence, the minimal interdependence of these software modules results in higher tolerance to subsequent changes and should be measured with a principled theory. Related work for evolution data-intensive applications [9], view redefinition [10,17,26], and data warehouse evolution [3,5,11,14] has provided rewriting techniques and theoretical cost models. Yet, a well-founded model, specifically tailored for the graph-based nature of ETL flows that assesses their vulnerabilities to changes is missing.

Related work also includes an approach to impact analysis and management of schema evolution, which represents the structural properties of the data warehouse schema, along with any views and queries defined over this schema, as a graph [30]. Our graph-based model captures all the parts (or, modules) of an environment, i.e., relations, views, and queries (which are practically the parts of ETL scripts that work the underlying data, or the elementary activities of a GUI-based scenario that are involved in the ETL process). Then, edges correspond to part-of or provider–consumer relationships. Given a database configuration, the impact of a schema change on the rest of the system is determined by exploiting the structure of the graph (i.e., by propagating the impact of the change via the involved edges). This clearly relates the structure of the graph, and its edges in particular, with the possibility that a component of the environment (a node in the graph) is affected by a certain evolution event. Furthermore, the evolution of the entire environment is regulated with the use of certain policies applied to the graph constructs. Example policies include either propagate/block a change or prompt the user for action. This way, administrators can regulate the evolution management, in a semi-automatic way, when changes on the database schema occur. Another research work employs a set of graph-theoretic metrics to measure evolution impact in data warehouse environments [29]. Although informally and briefly introduced in that work, these metrics are either degree-related or entropy-based metrics and compute the degree of dependence of nodes based on the structural properties of the system design from both a graph theoretic and an information theoretic perspective. However, these two works have not been adequately tested in real-world, large-scale applications.

In this paper, we built upon the aforementioned approaches with the goal of validating and experimentally assessing the proposed methods and metrics in real-world settings. We formally present these metrics and show that such metrics typically act as predictors for the vulnerability of a software module (either internal like a relation or external like a query) in a database-centric environment to future changes to the structure of the environment. Thus, we answer the aforementioned questions on the design quality of an ETL scenario from the perspective of maintenance.

Our experimental evaluation has been performed with a home-grown, publicly available, software tool, namely Hecataeus, which allows us to monitor evolution and perform evolution scenarios in database-centric environments. (For implementation details, the interested reader could read our ICDE'10 demo paper, Papastefanatos et al. [31].) The experimental analysis is based on a 6-month monitoring of seven real-world ETL scenarios processing data from statistical surveys. Our main goal was to examine different metrics over various ETL configurations and evolution events for assessing the usefulness and applicability of the proposed metrics (e.g., how well do they actually predict the impact of evolution events on a design construct). An additional desired objective was to identify which metric works best in different ETL configurations. Based on our findings, observations, and analysis, we disclose a list of lessons learned through this multi-month work.

In a nutshell, we have identified the schema size and module complexity as two important factors for the vulnerability of a system.

Schema sizes. The size of the schemas involved in an ETL design significantly affects the design vulnerability to evolution events. For example, source or intermediate tables with many attributes are more vulnerable to changes at the attribute level. Thus, a good design may involve tables with smaller schemas (e.g., we should maintain intermediate tables with a small number of attributes).

Functionality of ETL activity. The internal structure of an activity plays a significant role for the impact of evolution events on it. For example, activities with high out-degree and out-strengths tend to be more vulnerable to evolution and activities performing an attribute reduction (e.g., through either a group-by or a projection operation) are in general, less vulnerable to evolution events.

Module-level design. The module-level design of an ETL flow also affects the overall evolution impact on the flow. For example, it might be worthy to place schema reduction activities early in an ETL flow to restrain the flooding of evolution events. However, as we discuss in Sect. 5, such heuristics that significantly improve maintainability of ETL flows might contradict the normal practice for improving ETL performance.

In addition, we tested our metric suite against various ETL designs and have identified what metric provides better evolution prediction for specific ETL constructs. For modules with a single provider, the out-degree and out strength metrics (described in Sect. 3), which capture the dependencies with an adjacent module, provide better results. However, transitive degree metrics may act as predictors for the evolution of a module when it has many different providers and paths to evolving sources (e.g., queries).

Retrospectively, we can report that experimenting with real-world evolution scenarios in data-centric environments—and especially in ETL and data warehousing—is a difficult, long-termed, and time-consuming process. Such a process comprises a series of tasks, responsible for (a) recording all metadata information and workload definitions; (b) modeling and analyzing the dependencies between them; (c) collecting and categorizing all different types of evolution changes that occurred at different time periods and on different parts of the environment, and (d) recording how often each part of the environment (e.g., a query, a view) is affected by each change. Besides the non-trivial technical difficulties and effort needed for completing these tasks, in some cases, like for tasks (a) and (c), we had to deal with political and organizational issues as well. That is because more than one team of an organization is typically involved in these tasks and getting permissions, engaging people in exchanging and sharing information, and depending on other people's availability are not easy to carry tasks. This is an additional reason in favor of having a system toward the automatic or semi-automatic handling of evolution events in ETL and in general, in information management environments.

Outline The rest of this paper is structured as follows. Section 2 describes a graph-theoretic model for representing the constructs of a data warehouse environment. Section 3 presents the set of metrics. Section 4 presents our experimental findings. Section 5 provides a list of lessons learned. Finally, Sects. 6 and 7 discuss related work and conclude the paper, respectively.

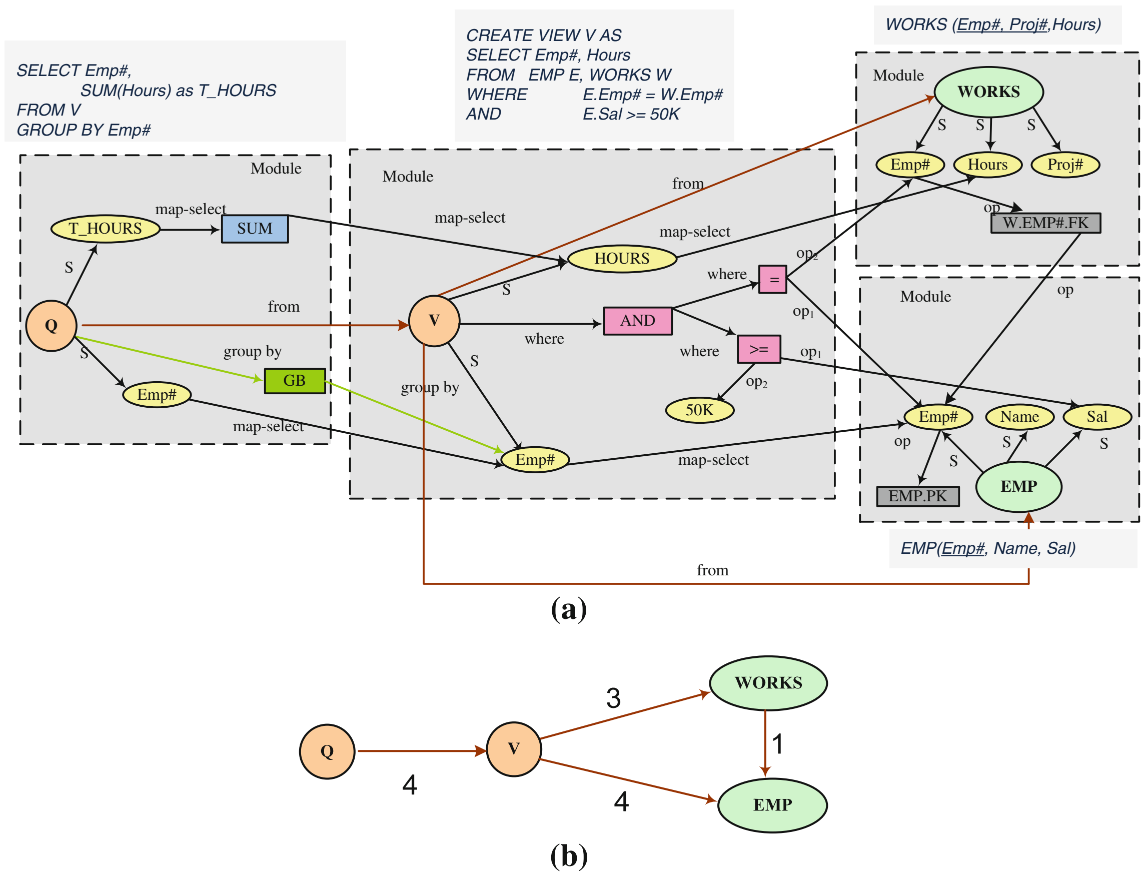
2 Modeling ETL Designs

In this section, we describe a graph-based model for ETL design. ETL designs are typically represented as graphs connecting activities and data stores. There are different styles for populating a data warehouse, like ETL, ELT, ETLT, and so on. Although these techniques have different performance characteristics, they do not differ in terms of modeling and thus, hereafter, we use the term ETL to capture all flavors of data warehouse population.

Our model uniformly covers relational tables, views, ETL activities, database constraints, and SQL queries as first class citizens. This model represents all such database constructs as a directed graph, named *evolution graph*, $G = (V, E)$. The nodes represent the entities of our model and the edges represent the relationships among these entities (mainly referring to part-of or provider-consumer relationships). The rationale for this modeling is to be able to represent *data-centric ecosystems* in a uniform way. In other words, we aim at a single, uniform way to model both database internals (like relations, views and constraints) and software modules external to the database (reports, forms, application programs, and so on). ETL flows offer a tight coupling of the database internal and external parts, along with the tight control of the application code by a small group of developers. Here, we present a brief description of our model. The interested reader may find a detailed model definition in another research paper [30].

A relation R $(\Omega_1, \Omega_2, \ldots, \Omega_n)$ in the database schema is represented as a directed graph, which comprises (a) a *relation node,* R, representing the relation schema; (b) n *attribute nodes*, $\Omega_1, \ldots, \Omega_n$, one for each of the attributes; and (c) n *schema relationships*, directing from the relation node towards the attribute nodes, indicating that the attribute belongs to the relation.

The graph representation of a Select-Project-Join-Group By (SPJG) query involves a new node representing the query, named *query node*, and *attribute nodes* corresponding to the schema of the query. The query graph is a directed graph connecting the query node with all its schema attributes, via *schema relationships*. In order to represent the relationship between the query graph and the underlying relations, we resolve the query into its essential parts: `SELECT, FROM, WHERE, GROUP BY, HAVING`, and `ORDER BY`, each of which is eventually mapped to a subgraph. The edges connected the involved attribute and operand nodes are annotated as *map-select, from, and where relationships.* Aliases in the `FROM` clause (mostly needed in self-joins for our modeling) are annotated with *alias* edges. The direction of the edges is from the query node to the attribute nodes. `WHERE` and `HAVING` clauses are modeled via a left-deep tree of logical operands to represent the selection formulae; all the involved edges are annotated as *where* and *having relationships*, respectively. Nested queries are part of this modeling, too. For the representation of aggregate queries, we employ two special purpose nodes: (a) a new node denoted as GB, to capture the set of attributes acting as the aggregators; and (b) one node per aggregate function labeled with the name of the employed aggregate function, e.g., `COUNT, SUM, MIN`. For the aggregators, we use edges directing from the query node towards the GB node that are labeled `<group-by>`, indicating *group-by relationships*. Then, the `GB` node is connected with each of the aggregators through an edge tagged also as `<group-by>`, directing from the `GB` node towards the respective attributes. These edges are additionally tagged according to the order of the aggregators; we use an identifier i to represent the ith aggregator. Moreover, for every aggregated attribute in the query schema, there exists an edge directing from this attribute towards the aggregate function node as well as an edge from the function node towards the respective relation attribute. Both edges are labeled `<map-select>` indicating the mapping of the query attribute to the corresponding relation attribute through the aggregate function node. The



```
SELECT Emp#,
        SUM(Hours) as T_HOURS
FROM V
GROUP BY Emp#
```

```
CREATE VIEW V AS
SELECT Emp#, Hours
FROM   EMP E, WORKS W
WHERE         E.Emp# = W.Emp#
AND           E.Sal >= 50K
```

```
WORKS (Emp#, Proj#,Hours)
```

```
EMP(Emp#, Name, Sal)
```

Fig. 1 **a** Graph and **b** abstract representation of an aggregate query on top of a view defined over two relations

representation of the ORDER BY clause of the query is performed similarly.

Functions used in queries are denoted as a special purpose node F having the name of the function. Each function has an input parameter list comprising attributes, constants, expressions, and nested functions, and one (or more) output parameter(s). SQL Views are considered either as queries or relations (materialized views). Finally, DML and loading statements are modeled as simple SQL queries.

Figure 1a depicts the proposed graph representation for two relations, EMP and WORKS. EMP has three attributes, Emp#, which is the primary key, Name and Sal. WORKS relation comprises Emp# (foreign key to EMP primary key), Proj# and Hours attributes. On top of these relations, there is a view performing a join operation and filtering the employees having SAL more than 50 K. Finally, the graph depicts an aggregate query defined on top of this view.

Moreover, an ETL activity (e.g., a loading, cleansing, filtering operation, etc.) is modeled as an SQL view defined over the sources of the activity; furthermore, an ETL workflow is modeled as a sequence of views corresponding to the activities of the flow.

A module is a sub-graph of the overall graph in one of the following patterns: (a) a relation with its attributes and all its constraints, (b) a view with its attributes, functions and operands, and (c) a query with all its attributes, functions and operands. Modules are disjoint with each other and connected through edges concerning foreign keys, map-select and so on. Within a module, we distinguish *top-level* and *low-level* nodes. Top level nodes are used to signify the identity of the module; for that purpose, query, relation and view nodes are used as top-level nodes. Low-level nodes comprise the rest of the module. Edges are classified into *provider* and *part-of* relationships. Provider edges are intermodule relationships, whereas part-of edges are intramodule relationships In Fig. 1, the graph comprises four modules corresponding to the query, view and the relation subgraphs.

Zoomed out graph The graph that we have presented so far has the benefit of accurately representing the interrelationships of the involved constructs at the finest level of detail (practically the attribute level when data-centric ecosystems are involved). As usually happens, this comes at a price: the graph soon becomes large and crowded with all the details of internal representation of the modules. In order to (a) concisely represent the overall graph in main memory and (b) visually depict it, whenever the scale becomes too large, it is useful to zoom out the graph into a summary of appropriate structure and size. The zoomed out graph is an abstraction of the detailed evolution graph which comprises only top level nodes and edges between them. Abstracting the graph into a modular representation at a coarser level of detail involves the following steps: (a) for each query, view or relation module, all low-level nodes and intramodule edges are suppressed and only the respective top-level node is retained, and, (b) all inter-module edges apart from *from* and *foreign key* edges are dropped. A surviving edge between two modules is annotated with a weight corresponding to the number of the edges that originally connected the two modules. We call this weight the *strength* of the edge as it assesses how tightly the involved modules are coupled. Figure 1b depicts the abstract modular representation of Fig. 1a.

Events The space of potential events comprises the Cartesian product of two subspaces; specifically, (a) the space of hypothetical actions (addition/deletion/modification), and (b) the space of the graph constructs sustaining evolution changes. We consider and collect several cases of data warehouse evolution events, such as a dimension is removed, or renamed, the structure of a dimension table is updated, (e.g., addition, removal or modification of a dimension attribute), a fact table is completely decoupled from a dimension (deletion of a FK) or decoupled from one dimension and coupled to another (update of a FK), the measures of a fact table change, or the source table of an ETL is altered. To avoid overloading the text, we refer the interested reader to ([30], section 3.1) for a detailed description of events.

An update can signify a change of data types or a renaming of a construct; our practical experience indicates that it mostly refers to the latter. We do not check for additions of fact, dimension, or source tables, because such events do not result in a direct impact on any other logical warehouse construct per se. Given these changes that can occur to a data warehouse, their basic impact is that all software modules that use these database structures must be rewritten. The impact can be both syntactic (in the sense that all views and queries using a deleted attribute will crash) and semantic (in the sense that a new attribute in a relation or a modified condition in a view might require a rewriting of all the queries that use it). Assume for example that an attribute *FullName* is split to attributes *FirstName* and *LastName* or a view condition ‘*Year* = 2007’ is altered to ‘*Year* > 2006’. The former change has syntactic impacts on all the queries using the attribute and the latter has semantic impact, since some of the queries using the view require exactly values of 2007, whereas some others will serve the purpose with any value greater than 2006.

Handling of events Given an event posed to one of the warehouse constructs (or, equivalently, to one of the nodes of the graph of the warehouse that we have introduced), the impact involves the possible rewriting of the constructs that depend upon the affected construct either directly, or transitively. In a non-automated way, the administrator has to check all of these constructs and restructure the ones he finds appropriate. This process can be semi-automated using our graph-based modeling and annotating the nodes and the edges of the graph appropriately with policies in the event of change. Assume for example, that the administrator guarantees to an application developer that a view with the sum of sales for the last year will always be given. Even if the structure of the view changes, the queries over this view should remain unaffected to the extent that its `SELECT` clause does not change. On the contrary, if a query depends upon a view with semantics ‘*Year* = 2007’ and the view is altered to ‘*Year* > 2006’, then the query must be rewritten.

The main idea in our approach involves annotating the graph constructs (relations, attributes, and conditions) sustaining evolution changes (addition, deletion, and modification) with policies that dictate the way they will regulate the change. Three kinds of policies are defined: (a) *propagate* the change, meaning that the graph must be reshaped to adjust to the new semantics incurred by the event; (b) *block* the change, meaning that we want to retain the old semantics of the graph and the hypothetical event must be vetoed or, at least, constrained, through some rewriting that preserves the old semantics; and (c) *prompt* the administrator to interactively decide what will eventually happen. Papastefanatos et al. [28] have proposed a language that greatly alleviates the designer from annotating each node separately and allows the specification of default behaviors at different levels of granularity with overriding priorities.

Given the annotation of the graph, there is also a simple mechanism that (a) determines the status of a potentially affected node on the basis of its policy, (b) depending on the node’s status, the node’s neighbors are appropriately notified for the event. Thus, the event is propagated throughout the entire graph and affected nodes are notified appropriately. The `STATUS` values characterize whether (a) a node or one of its children (for the case of top-level nodes) is going to be deleted or added (e.g., `TO-BE-DELETED, CHILD-TO-BE-ADDED`) or (b) the semantics of a view have changed, or (c) whether a node blocks the further propagation of the event (e.g., `ADDITION-BLOCKED`).

3 Metric Suite

This section presents a set of metrics based on graph theoretic properties of the evolution graph for measuring and evaluating the design quality of a database centric environment with respect to its ability to sustain changes. For our analysis, we examine the graph (a) at its most detailed level (node level) that involves all the attributes of relations, views

Table 1 Degree related metrics

Notation	Metrics for any node
$D^I(v)$	In-degree of a node v
$D^O(v)$	Out-degree of a node v
$D(v)$	Degree of a node v
$TD^I(v)$	In-transitive degree of a node v
$TD^O(v)$	Out-transitive degree of a node v
$TD(v)$	Transitive degree of a node v
$D^{Is}(v)$	In-degree of a module v
$D^{Os}(v)$	Out-degree of a module v
$D^s(v)$	Degree of a module v
$TD^{Is}(v)$	In-transitive degree of a module v
$TD^{Os}(v)$	Out-transitive degree of a module v
$TD^s(v)$	Transitive degree of a module v

Table 2 Entropy-based metrics

Notation	Metric
$H(v)$	Entropy of a node v
$H^s(v)$	Entropy of a module v

and queries, along with the internals of the queries, and, (b) at a coarse level of abstraction (module level), where only relations, views and queries are present. An earlier work, has briefly introduced these metrics [28]. Here, we formally define them and provide the intuition and a detailed definition for each metric. The whole set of proposed metrics is presented in Tables 1 and 2.

3.1 Degree-Related Metrics

The first family of metrics concerns simple properties of each node or module in the graph and specifically the degree of nodes. The main idea lies in the understanding that the in-degree, out-degree and total degree of a node v demonstrate in absolute numbers the extent to which (a) other nodes depend upon v, (b) v depends on other nodes, and (c) v is interacting with other nodes in the graph, respectively.

Specifically, let $G(V, E)$ be the evolution graph of a database centric environment and $v \in V$ a node of the graph; then

Definition 1 Degree of Node: The *In-degree*, D^I(v), *Out-degree*, D^O(v) and *Degree*, D(v) of the node v are the total number of incoming, outgoing and adjacent edges to v. That is

$$D^I(v) = |e_{\text{in}}|, \text{ for all edges } e_{\text{in}} \in E \text{ of the form } (y_i, v), y_i, v \in V$$

$$D^O(v) = |e_{\text{out}}|, \text{ for all edges } e_{\text{out}} \in E \text{ of the form } (v, y_i), y_i, v \in V$$

$$D(v) = D^I(v) + D^O(v)$$

Transitive Degrees. The simple degree metrics of a node v are good measures for finding the number of nodes that directly depend on v, or on which v directly depends on, but they cannot detect the transitive dependencies between nodes. This typically occurs whenever a query accesses a view, which is of course defined over one or more views and relations. The metrics related to simple degrees cannot capture the fact that a change in a relation can eventually propagate to a large number of dependent modules *transitively*. Take, for example, the case of an ETL flow where a source relation may feed only a single activity; however, a change in this relation can transitively propagate and affect the entire workflow. In the context of our graph model, we say that a node v_1 is transitive dependent on another node v_2 if there is a path from v_1 towards v_2. Therefore, we employ the following definition for the transitive degrees of a node v with respect to the rest of the graph:

Definition 2 Transitive Degree of Node: The *In-Transitive, TD*$^{\text{I}}$(v), *Out-Transitive*, *TD*$^{\text{O}}$(v), and *Transitive degree, TD*(v) of a node $v \in V$ with respect to all nodes $y_i \in V$ are given by the following formulae:

$$TD^I(v) = \sum_{y_i \in V} |paths(y_i, v)|, \ y_i \in V$$

$$TD^O(v) = \sum_{y_i \in V} |paths(v, y_i)|, \ y_i \in V$$

$$TD(v) = TD^I(v) + TD^O(v)$$

Module degree. The aforementioned metrics are able to capture the significance of individual nodes of the graph at a fine-grained level. However, it is quite possible that administrators and designers are interested to see the graph properties at the module level. A first possible reason for this requirement is the graph's size: the administrators/designers might be willing to pay a small price in accuracy in favor of faster computation. Also, the metric properties of the modules (seen as black boxes) per se could be of interest to the administrators and the developers. To address this requirement, one can measure the degrees of *the zoomed-out* graph. As already mentioned in chapter 2, zooming-out operation on the graph provides an abstract view of the modules of the graph, which comprises only top-level nodes, i.e., relations R, views VS and queries Q and edges between them. All edges are annotated with a strength corresponding to the number of edges previously connecting these modules. Thus, we define the module degree for a node of the zoomed out graph as

Definition 3 Degree of Module (Strength): The *In-Module*, $D^{Is}(v)$, *Out-Module*, $D^{Os}(v)$ and *Module Degree*, $D^s(v)$of a node v are given by the following formulae:

$$D^{Is}(v) = \sum_{y_i \in V} strength(y_i, v), \ y_i, v \in \{R, Q, VS\}$$

$$D^{Os}(v) = \sum_{y_i \in V} strength(v, y_i),\ y_i, v \in \{R, Q, VS\}$$

$$D^s(v) = D^{Is}(v) + D^{Os}(v)$$

Module transitive degree. Similarly to above, we may extend the transitive degrees to the zoomed-out graph according to the following definition:

Definition 4 Transitive Degree of Module (Strength): The *In-Module, $TD^{Is}(v)$, Out- Module, $TD^{Os}(v)$*, and *Module Transitive degree, $TD^s(v)$* of a node v $\in$*{R,Q,VS}* with respect to all nodes y_i $\in$*{R,Q,VS}* are given by the following formulae:

$$TD^{Is}(v) = \sum_{y_i \in V} \sum_{e_p \in paths(y_i, v)} strength(e_p),\ y_i, v \in \{R, Q, VS\}$$

$$TD^{Os}(v) = \sum_{y_i \in V} \sum_{e_p \in paths(v, y_i)} strength(e_p),\ y_i, v \in \{R, Q, VS\}$$

$$TD^{\mathrm{S}}(v) = TD^{\mathrm{Is}}(v) + TD^{\mathrm{Os}}(v)$$

3.2 Entropy-Based Metrics

The last family of metrics presented is related to the information theoretic notion of entropy. Entropy is viewed as an arcane subject related somehow to uncertainty and information [32]. Given a set of events $A = [A_1, \ldots, A_n]$ with probability distribution $P = \{p_1, \ldots, p_q\}$, respectively, entropy is defined as the average information obtained from a single sample from A:

$$H(A) = -\sum_{i=1}^{n} p_i \log_2 p_i.$$

Entropy is strongly related to the information that is "hidden" in a probabilistic model. For instance, in a uniform probabilistic model, all events are equally likely to occur and therefore the entropy of the model is maximum.

In our evolution context, the notion of entropy is used to evaluate the extent to which a node is *likely* to be affected by a *random* evolution event on the graph. Intuitively, this likelihood is strongly related to the number of other nodes in the graph on which this node depends (i.e., connected to) either directly or transitively. Nodes connected via multiple paths to many parts are more likely to be affected if a random event occurs on the graph. Thus, entropy measures either the a priori uncertainty of the impact of an event on a part of the graph or equivalently the a posteriori amount of information we get from the knowledge that a part of the graph has been affected by an event. The more unpredictable the impact of a

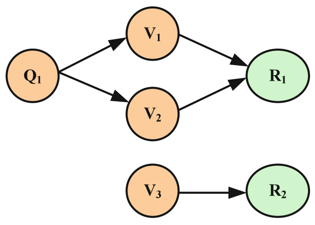


Fig. 2 A graph with a query, three views, and two relations

schema change on a part (either a module or a specific node) of the graph is, the higher the entropy is that characterizes this impact. For example, consider the graph of Fig. 2, where a query Q_1 is defined on top of two views, V_1 and V_2, which both access a single relation R_1; a third view, V_3, is defined on top of a second relation, R_2. Q_1 depends on three out of five modules in the graph, i.e., V_1, V_2 and R_1, and thus, it has a high *potential* that it will be affected by a random event on the graph, in contrast with V_3, which is connected only to R_2 and affected by changes occurring only to this relation.

The following definitions introduce the metrics of the entropy of a node at the detailed and the zoomed out graph:

Definition 5 Entropy of Node: Assume a node v in our graph $G(V, E)$. We define the probability that $v \in V$ is affected by an arbitrary evolution event e over a node $y_k \in V$ as the number of paths from v towards y_k divided by the total paths from v towards all nodes in the graph, i.e.,

$$P(v|y_k) = \frac{|paths(v, y_k)|}{\sum_{y_i \in V} |paths(v, y_i)|}, \text{ for all nodes } y_i \in V.$$

Then, the information we gain when a node v is affected by an event that occurred on node y_k is $I(P(v|y_k)) = \log_2 \frac{1}{P(v|y_k)}$ and the entropy of node v with respect to the rest of the graph is then

$$H(v) = -\sum_{y_i \in V} P(v|y_i) \log_2 P(v|y_i), \text{ for all nodes } y_i \in V.$$

Observe that high entropy values correspond to these parts of the graph, that are dependent on many providers either directly or transitively, *capturing in a "smoother" way than the local or the transitive degrees the dependencies in the graph.*

Definition 6 Entropy of Module: Moreover, we can apply the previously used technique to the zoomed out-graph $G^s(V^s, E^s)$, by defining the probability of a node $v \in V^s$ to be affected by an evolution event over a node $y_k \in V^s$ as

$$P^s(v|y_k) = \frac{\sum_{e_p \in paths(v, y_k)} strength(e_p)}{\sum_{y_i \in V^s} \sum_{e_p \in paths(v, y_i)} strength(e_p)}, \text{ for all nodes } y_i \in V^s.$$

with $e_p \in E^s$ being the edges of all the paths of the zoomed out graph stemming from v towards y_k. Similarly, the entropy of node $v \in V^s$ is

$$H^s(v) = -\sum_{y_i \in V^s} P^s(v|y_i) \log_2 P^s(v|y_i) \quad \text{for all nodes } y_i \in V^s.$$

A summary of the proposed set of metrics is provided in Tables 1 and 2.

3.3 Rationale for Our Approach and Broader Perspective

Before describing how the aforementioned metrics have been applied to a real case study, we discuss the rationale for the choice of this metric suite.

We base our approach on the modeling power of the graph proposed. Our graph-based model is simple, comprehensive, rigorously defined, and it has a uniform treatment of all involved constructs. Nodes correspond to both detailed entities like attributes or values and to higher level entities like relations or queries (this can further be abstracted to databases, scripts, libraries, and so on). Edges denote *any* form of relationship, with two kinds of relationship being the most prominent ones: (a) part-of (between high-level modules and their constituents) and (b) provider-consumer in terms of data provision. Both kinds of edges capture the notion of *dependency*: an edge $e(v, u)$ from node v to node u, signifies that node v is potentially affected whenever u is affected too.

We believe that dependency is the cornerstone of maintenance in data-centric ecosystems. To this end, we devised two families of metrics based on mathematical fundamentals to quantify the dependency of a node. The first family of metrics has to do with graph-theoretic properties. In the context of our studies, we focused on the properties of *individual nodes*, and, in fact, we opted to constrain ourselves to simple metrics like degrees and we explore the main kinds of degrees. At the same time, we also explore two different modes of locality. First, we are interested in the local degree, as a simple measure of direct dependence between nodes. Second, we operate in a workflow-like environment where data are "copied" from one module to another for further processing and thus, we explore the idea of assessing transitive degrees as measures of the overall dependency of a node to other nodes. The second family of metrics serves an information-theoretic rationale: what if a random evolution event appears in the graph? How likely is a node v to be affected by it due to its dependency to other affected nodes? Thus, we follow a mathematically founded, information theoretic approach to capture the vulnerability of a node to a random evolution event.

In both families, we do distinguish between coarse-grained metrics at the module level versus detailed metrics at the full extent of the graph. As already mentioned, the coarse-grained summary of the graph was intended to alleviate the user from the information overload of all the miniscule details of module internals at the full extent of the graph. However, due to the size of the graph, it is possible that large ecosystems will have to be assessed at a coarser detail level for performance reasons; then, the open question is how well do coarse-level metrics approximate the detailed ones.

From a subjective viewpoint, without excluding the possibility of more sophisticated metrics, we support the idea of *simple* metrics. In this paper, we are exploring the problem from an empirical perspective and are primarily interested to see what simple metrics appear to work well in the case study we have encountered. We deem *simplicity* as an inherent good quality that makes concepts easily understandable and explainable to the involved stakeholders (let alone efficiently computed). But even under the prism of simplicity, we do not claim that our metric suite exhausts all possibilities, either with respect to their mathematical foundation or with respect to practical intuition (or any other rationale that can be used to build a set of metrics); on the contrary, we do anticipate that other metrics can possibly be devised to assess the vulnerability of a node to change.

4 A Real-World Case Study and Experimental Validation

The metrics presented in Sect. 3 express the degree of dependence or importance of a node in an objective and quantifiable way. Yet, this is a characterization of the structural properties of a node within the graph. How accurately can we use the metrics to *predict* the vulnerability of a node to change? Is it fair to say that the *more dependent* a node is, the *higher the probability* that it is affected by evolution changes? To address this issue, we have evaluated the proposed metrics with real-world ETL scenarios. The goal of our analysis is to evaluate the extent to which our metrics are good indicators for the prediction of the effect that evolution events have. A clear desideratum in this context is the determination of the most suitable metric for this prediction according to the different types of evolved constructs.

4.1 Experimental Setting

In our experiments, we have studied a data warehouse scenario, which involves real-world evolution scenarios of ETL workflows that were monitored for a period of six months. The environment involves a set of seven real ETL workflows extracted from a Greek public sector's data warehouse maintaining information for farming and agricultural statistics. The ETL flows extract information out of a set of seven source tables, namely S1 to S7 and 3 lookup tables, namely L1 to L3, and load it to seven target tables, namely T1 to T7, stored in the data warehouse. The seven scenarios comprise

Table 3 ETL scenarios configuratrion

ETL	# Activities	Sources	Tmp Tables	Targets
ETL 1	16	L1,L2,L3,S1,S4	T1_Tmp, T2_Tmp, T3_Tmp	T1, T2, T3
ETL 2	6	L1,S2	T1_Tmp, T3_Tmp	T3
ETL 3	6	L1,S3	T1_Tmp, T3_Tmp	T3
ETL 4	15	L1,S4	T1_Tmp, T3_Tmp, T4_Tmp	T3, T4
ETL 5	5	S5	T1_Tmp, T5_Tmp	T5
ETL 6	5	S6	T1_Tmp, T6_Tmp	T6
ETL 7	5	S7	T1_Tmp, T7_Tmp	T7
Total	58			

Table 4 Number of attributes in ETL source tables

Table	S1	S2	S3	S4	S5	S6	S7	L1	L2	L3
# Attributes	59	160	82	111	13	7	5	7	19	7

Table 5 Distribution of events at the ETL tables

Source	Change type	Occurrence	Affected ETL
L1	Add Attribute	1	ETL 1, 2, 3, 4
L1	Add Constraint	1	ETL 1, 2, 3, 4
L2	Add Attribute	3	ETL 1
L3	Add Attribute	1	ETL 1
S1	Add Attribute	14	ETL 1
S1	Drop Attribute	2	ETL 1
S1	Modify Attribute	3	ETL 1
S1	Rename Attribute	3	ETL 1
S1	Rename Table	1	ETL 1
S2	Add Attribute	15	ETL 2
S2	Drop Attribute	4	ETL 2
S2	Rename Attribute	121	ETL 2
S2	Rename Table	1	ETL 2
S3	Rename Attribute	80	ETL 3
S3	Rename Table	1	ETL 3
S4	Add Attribute	58	ETL 1, 4
S4	Drop Attribute	26	ETL 1, 4
S4	Modify Attribute	1	ETL 1, 4
S4	Rename Attribute	27	ETL 1, 4
S4	Rename Table	1	ETL 1, 4
S5	Modify Attribute	2	ETL 5
S5	Rename Table	1	ETL 6
S6	Rename Table	1	ETL 6
S7	Rename Attribute	5	ETL 7
S7	Rename Table	1	ETL 7
T1	Drop Attribute	1	ETL 1
T1	Modify Attribute	1	ETL 1
T1_tmp	Drop Attribute	1	ETL 1-7
T1_tmp	Modify Attribute	1	ETL 1-7
T2	Add Attribute	15	ETL 1
T2	Modify Attribute	2	ETL 1
T2_tmp	Add Attribute	15	ETL 1
T2_tmp	Modify Attribute	2	ETL 1
T5	Modify Attribute	2	ETL 5
T5_tmp	Modify Attribute	2	ETL 5
Total		416	

a total number of 58 activities extracting, filtering and loading data into the target tables. They, also, make use of seven temporary tables (each target table has a temporary replica) for keeping data in the data staging area, as shown in Table 3. The warehouse maintains statistical information collected from surveys, held once per year via questionnaires. The survey data are primarily stored in the S1–S7 source tables and are subsequently processed so that they can be integrated in the organization's warehouse and queried. Each survey varies its schema according to the different number and types of questions comprising the survey's questionnaire; however, there are several common elements in all surveys. Table S1 holds information about the metadata of the survey (e.g., the year held, the sample size, etc.), which are rarely altered or renamed and mostly complemented or specialized yearly by adding new attributes in the survey's metadata. All other source tables (S2–S7) retain the answers to the questionnaires, where most alterations occur every year. The size of the schema of each table, in terms of number of attributes, is shown in Table 4.

Our choice for experimenting with these scenarios was based on their evolution behavior that satisfied the following criteria: The first criterion was the ease of collecting real evolution events. As statistical surveys are held once a year, most source tables are suffering the majority of evolution events during a short peak period when surveys are designed and modeled in the database. This fact enabled us to collect and analyze most evolution events at once. The second criterion was the number and diversity of events. The scenarios examined exhibit a large number of evolution events covering a broad variety of alterations on the source tables (see Table 5). Finally, the third criterion was the diversity in the designs of the ETL flows. The chosen ETL scenarios enabled us to evaluate our metrics in simple (e.g., ETL5, ETL6, ETL7) as well as more complex (e.g., ETL1) flows.

All ETL scenarios were source coded as PL\SQL stored procedures in the data warehouse. First, we extracted embedded SQL code (e.g., cursor definitions, DML statements, SQL queries) from activity stored procedures. Table definitions (i.e., DDL statements) were extracted from the source and data warehouse dictionaries. Each activity was represented in our graph model as a view defined over the previous activities, and table definitions were represented as relation graphs.

We have used a homegrown software artifact, called Hecataeus[1] for the graph representation, the application of changes on the source tables, the evaluation of the metrics on the graph, and the identification of the impact of evolution events. Hecataeus is an open-source software tool for enabling impact prediction, what-if analysis, and regulation of relational database schema evolution. Under the hood, it supports the proposed graph-based modeling technique and represents database schemas and database constructs, like queries and views, as graphs. Our tool enables the user to create hypothetical evolution events and examine their impact over the overall graph before these are actually enforced on it. It also allows the definition of rules (i.e., policies) on nodes of the graph (in the form of annotations) that regulate the propagation of the evolution impact on specific parts. Most importantly, Hecataeus includes a metric suite for evaluating the impact of evolution events and detecting crucial and vulnerable parts of the system.

Our study is based on the evolution of the source tables and their accompanying ETL flows, which has happened in the *context of maintenance due to the change of requirements at the real world*. As already mentioned, source S1 stores the constant data of the surveys and did not change a lot. The rest of the source tables (S2–S7), on the other hand, sustained maintenance. The recorded changes in these tables mainly involve restructuring, additions, and renaming of the questions comprising each survey, which are furthermore captured as changes in the source attributes names and types. The set of evolution events includes renaming of relations and attributes, deletion of attributes, modification of their domain, and last, addition of primary key constraints. We have recorded a total number of 416 evolution events and the number of events per table is shown in Table 5. Observe that the majority of evolution changes concerns attribute renaming and attribute additions. These findings were due to the evolution context of the examined warehouse sources.

The last column in Table 4 shows the flows as affected by each change. L1 table is used in 4 ETL flows (1–4), S4 in 2 flows, namely ETL1 and ETL4, whereas all other source tables are used only to one flow. The most evolved table is S2 with a total of 141 changes and S4 follows with 113 changes. S2, however, supplies only one flow (ETL 2), whereas S4 supplies both ETL1 and ETL4. In addition, schema changes were applied in T_1, T_2, and T_5 target tables and their respective temporary tables as a result of the changes in the ETL sources. All evolution changes were applied in the form of annotations on the nodes of the graph.

Summarizing, the configuration of our experiments involved representing the ETL workflows in our graph model as well as the recorded evolution events on the nodes of the source, lookup and temporary tables. We then applied each event sequentially on the graph assuming that no rules constrain the propagation of the change towards the nodes of the graph. We, finally, monitored the impact of the change towards the rest of the graph by recording the times that a node has been affected by each change.

4.2 Experimental Validation

We have first evaluated the graph metrics on the seven ETL graphs and then applied the evolution events of Table 4 sequentially on these ETL graphs. We monitored each node of the graphs on how many times it was affected by an event. This measurement constitutes the baseline measurement that simulates what would actually happen in practice. This baseline measurement is compared with all measured metrics. In the rest, we discuss our findings organized according to each ETL flow.

ETL1. The first workflow (Fig. 3) comprises two source tables (S1, S4), three lookup tables (L1–L3), three target tables (T1–T3) along with their temporary tables and 16 activities. Both source tables contain a large number of attributes, namely 59 for S1 and 111 for S4 (there are no foreign keys defined), lookup tables are small in size, target tables T1 and T2 are two dimension tables with 74 and 38 attributes, respectively, whereas T3 is a fact table with 16 attributes. S1 data are extracted and loaded in the two dimension tables through the upper branch of the flow and to the fact table via the lower branch. S4 contains measure data that are loaded in the fact table. Regarding the functionality of the activities, the activities 1–5 perform extraction and filtering of data from the two source tables. Then, activity 9 joins the two sources and projects all attributes of S1 but only a small number of attributes of S4 (most data coming from S4 table are loaded via the ETL4 scenario). Activities 10–12 of the upper branch update the data with lookup values and activities Q2 and Q3 project and load data to T1, T2 temporary tables. In the lower branch, Q4 activity updates the data with values from L3 and loads it to T3 temporary table. Finally, activities 6–8 perform the final loading to the target tables of the data warehouse. Based on the functionality of each activity, we distinguish the *filtering* activities performing a select or a transforming operation on their source, (e.g., ETL1_ACT1, ETL1_ACT2, ETL1_ACT5, etc.), *joining* activities combining data from more than one sources (e.g., ETL1_ACT9, ETL1_ACT11,

[1] For more details about Hecataeus, we refer the interested reader to our ICDE'10 demonstration paper [31] and the project website: www.cs.uoi.gr/~pvassil/projects/hecataeus.

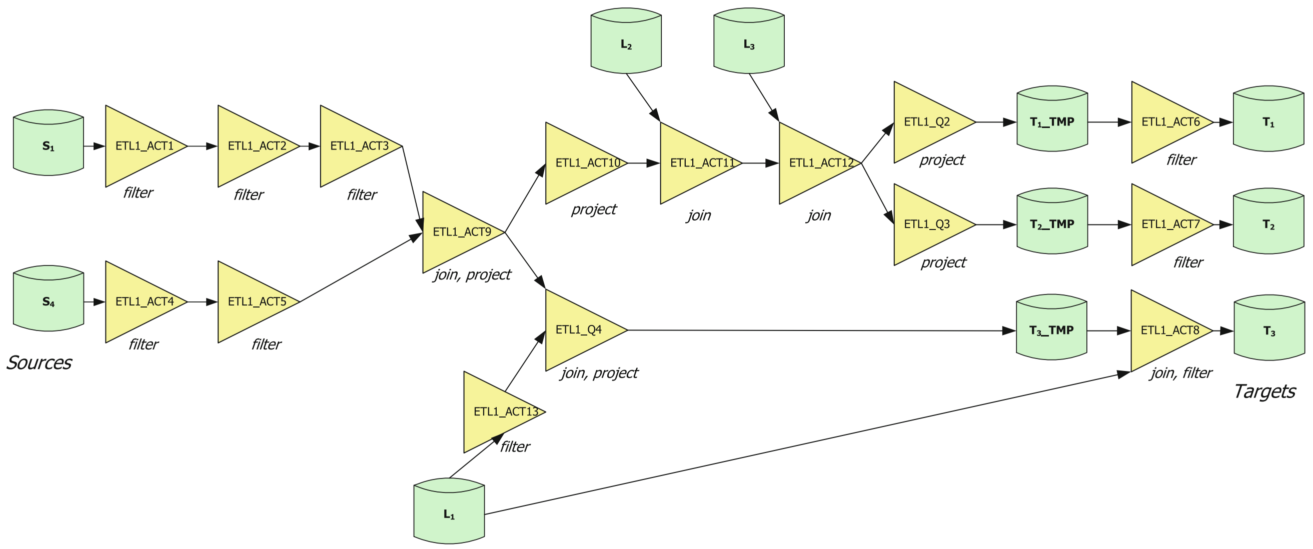


Fig. 3 Workflow of the first ETL scenario, ETL1

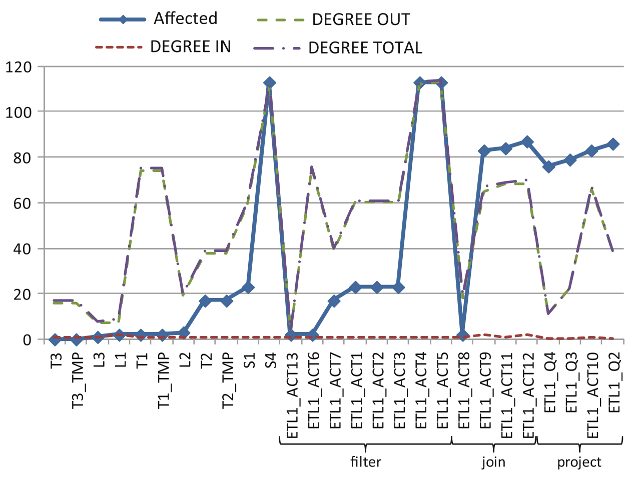


Fig. 4 Results for degree metrics for ETL1

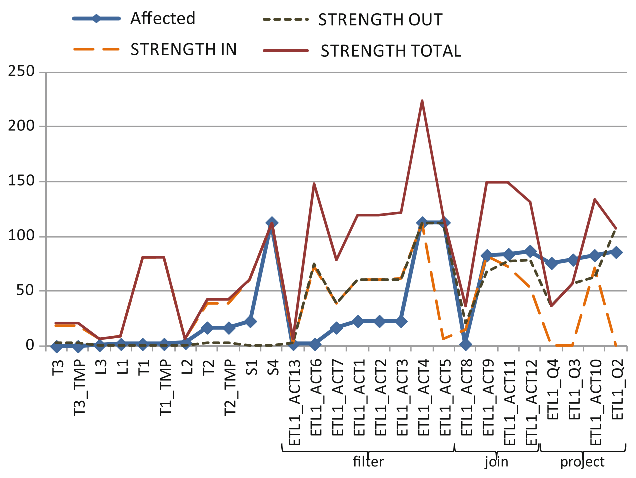


Fig. 5 Results for strength metrics for ETL1

ETL1_Q4, etc.) and finally activities that *project* a subset of the available attributes of their sources (e.g., ETL1_Q4, ETL1_ACT10, ETL1_ACT12, ETL1_Q2, ETL1_Q3, etc.).

In Figs. 4, 5, 6 and 7, we present the results for the first of the 7 ETL scenarios, grouped by the different types of metrics. In Fig. 4, we present the simple degree metrics, in Fig. 5 the transitive degree along with the entropy metrics (entropy and transitive metrics have been scaled up and down, respectively, for comparison reasons), and in Fig. 6 the strength metrics and last in Fig. 7 the transitive strength metrics. The goal is to show the overall trend of the examined metrics (i.e., we are not interested in the absolute numbers) with respect to the type of module and the times it is *affected* by all events that occurred at its source. In all figures, the tables are positioned on the left side followed by the activities. In Figs. 4 and 5 activities are first arranged by their type and then by the affected series, whereas in Figs. 6 and 7 by their type and then by their topological order in the workflow.

The *affected* series for the tables (in all figures) corresponds to the number of changes that occurred at their schemas as presented in Table 4. The most "evolved" table in ETL1 is S4, followed by S1, whereas T2 dimension table (along with the relevant temporary table) exhibits the highest number of changes among the data warehouse tables. This is due to the fact that most source schema changes, occurred at S1, have been exclusively propagated to the T2 dimension table, without altering T1 dimension table. Filtering activities are affected by all changes occurring at their source table. For example, ETL1_ACT1, ETL1_ACT2, and ETL1_ACT3 activities exhibit the same affected number with S1; ETL1_ACT4, ETL1_ACT5 with S4, etc. As we mentioned earlier, ETL_ACT9 projects all S1 attributes, but

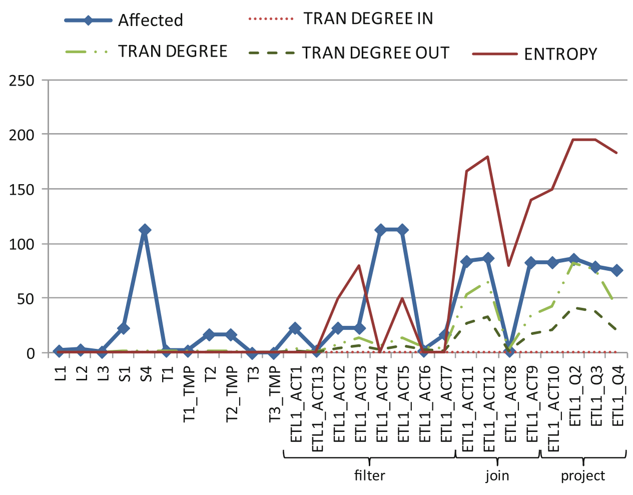


Fig. 6 Results for transitive degrees and entropy metrics for ETL1

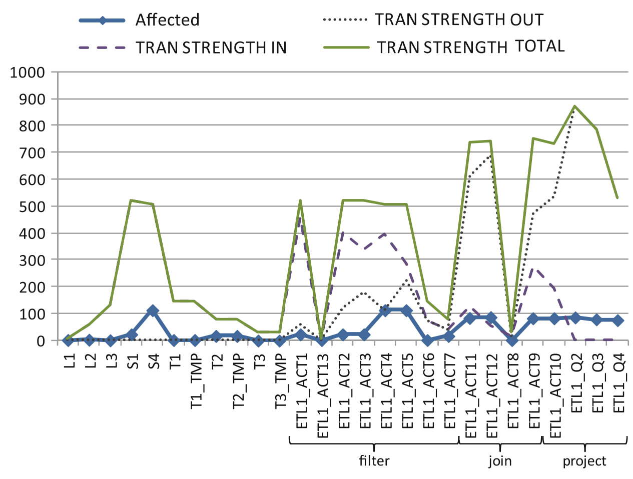


Fig. 7 Results for transitive strength metrics for ETL1

only a small number of S4 attributes. Thus, all other project and join activities, positioned after ETL_ACT9, are mostly affected by S1 evolution changes and S4 attribute additions. Next, we discuss our findings for each family of metrics.

The *in-degree* metric is significantly low and invariant to the number of events affected all modules. The *out-degree*, and consequently the total degree, follows proportionally the number of events that occurred on the source and lookup tables of the workflow, as well as on the T2 target table (the table that “absorbed” most source schema changes). The out-degree metric captures the size of the table schema, which seems to be a crucial factor for the evolution of the tables. The out-degree for all types of activities follows a similar trend. The out-degree for activities mostly captures the number of attributes that are projected by each activity and as a result, project activities exhibit low out-degree values. However, most activities have been affected by evolution proportionally to their out-degrees, except for some peaks such as ETL1_ACT6. ETL1_ACT6 activity depends on the T1_Temp table, which, however, has not sustained any schema changes.

The *strength* metrics for the tables (shown in Fig. 5) follows an opposite trend from the simple degrees. The *strength-out* is invariant to the affected series, whereas the *strength-in* and *total strength* follows the affected series for each table (with the exception of T1 and T1_temp). This can be explained by the fact that the strength-in metric for a table captures attribute dependencies between a module and this table. Figure 5 shows that the more “used” is a table, the higher the probability is to evolve. Filtering activities show a similar trend for both in and out strengths, except for ETL1_ACT5. The latter with all other activities show that the out-strength values follow more smoothly the affected series. Again, the peak for ETL1_ACT6 is due to its dependence from T1_Temp table.

The *transitive degrees and entropy* metrics, shown in Fig. 6, do not provide useful results for tables, because all values are insignificant to the affected series. This is not surprising as the entropy metric and transitive out-degree for a module captures the number of other modules on which this module depends transitively and tables have few or no dependencies on other modules. Regarding the activities, transitive out degree metrics and entropy follow smoothly the trend of the affected series, except for ETL1_ACT4 dive. ETL1_ACT4 exhibits a low value for the transitive degree metric, as it depends exclusively on S4, which, however, is affected by a large number of evolution events.

Similar to the simple strengths, the *transitive in* and *total strengths* follows the trend of affected series for tables. On the other hand, *transitive out* and *total strengths* seem to be more precise for activities, especially for join and filtering ones.

ETL2 and ETL3. The next two flows, ETL2 and ETL3, are shown in Figs. 8 and 10, respectively. Both flows behave similarly and load data from S2 and S3 to the T3 fact table; L1 and T1_temp tables are used as lookup tables. S2 contains 160 attributes and S3, 83 attributes. Both flows have no branches, whereas the activities mainly filter data from their sources and update lookup values. We observed that the number of events on these tables follows proportionally the tables’ size and out-degree metric is again validated as a candidate predictor for the behavior of the evolution of the tables. The examined metrics on the activities of these two flows show similar results with ETL1. Out-Degree and out-strength are the most accurate predictors, but also transitive degree metrics (entropy and total transitive strength) follow the “affected” series. In Fig. 9a–d, we present the results for all the examined metrics for ETL2, and in Fig. 11a–d, the corresponding results for ETL3.

ETL4. Figure 12 shows the configuration of ETL4 and the respective results are shown in Fig. 13a–d. The ETL4 flow has one source, namely S4, comprising a fairly large number of attributes, 111. The two data warehouse tables, T3 and

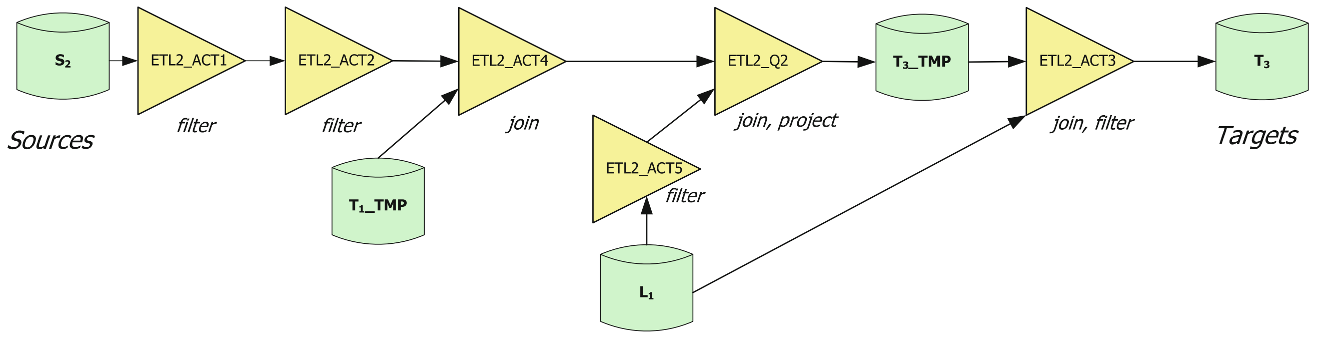


Fig. 8 ETL2 workflow

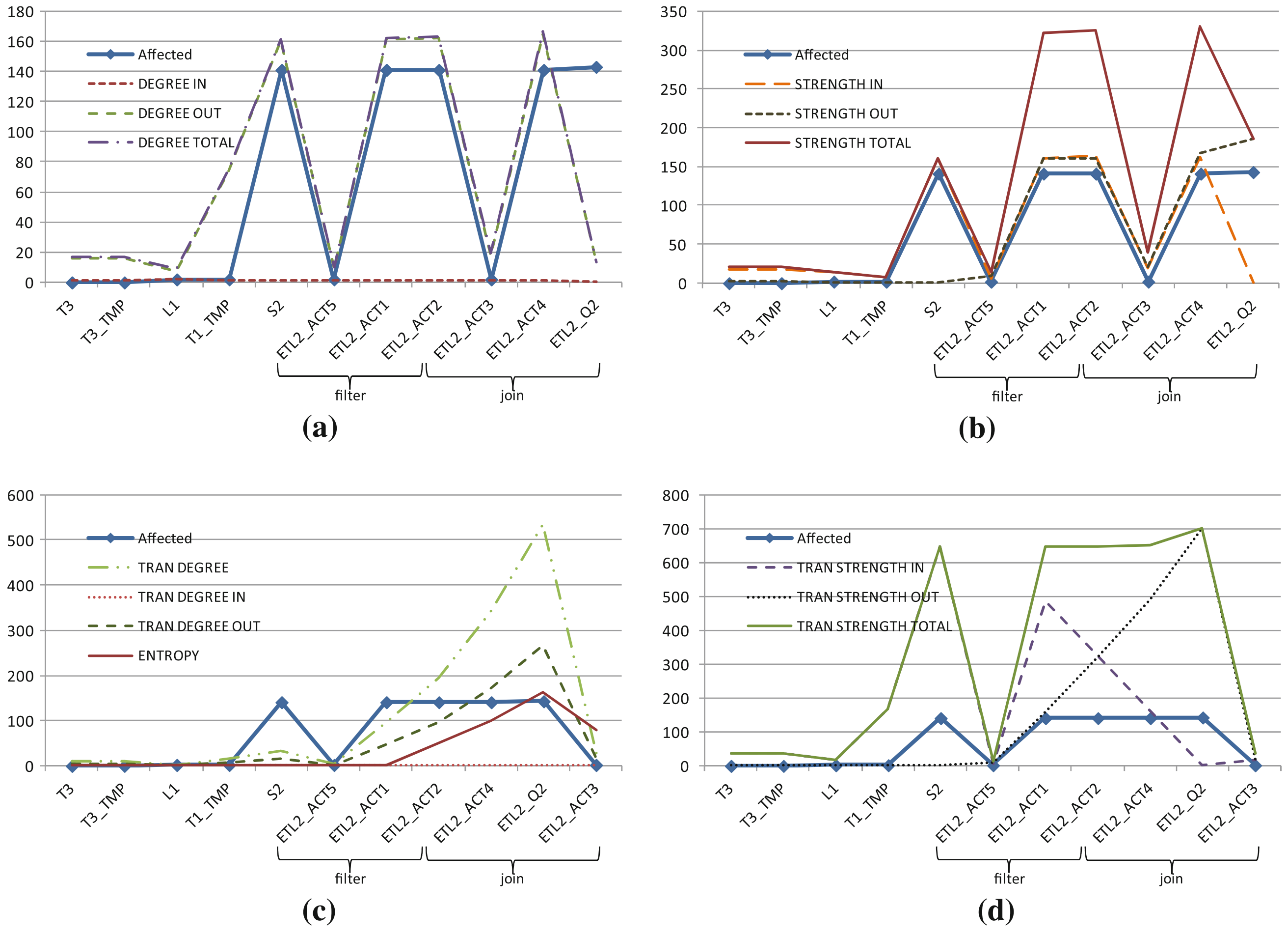


Fig. 9 Results for examined metrics for ETL2

T4, are both fact tables and are populated by a series of 9 activities, i.e., ETL4_Q2-ETL4_Q10. Each activity projects a different subset of source attributes and maps them to measure attributes in the data warehouse. Finally, L1 and T1 are used as lookup tables and ETL4_ACT3 and ETL4_ACT4 are used for transferring data from the temporary tables to the data warehouse tables.

The results for ETL4 are illustrated in Fig. 13a–d. Again, the out degree can be used as a predictor for the events that affect a table. Out-degree, total strength, and transitive total strength metrics are quite precise for the activities of this scenario. In contrast with the previous scenarios, scaled entropy and transitive out-degree are also proven to be good estimators for this setting. This can be explained by the fact that ETL4 is a short workflow, with only a few steps of processing and few transitive dependencies. Therefore, transitive degree metrics exhibit the same trend with the simple degree or strength metrics for all activities.

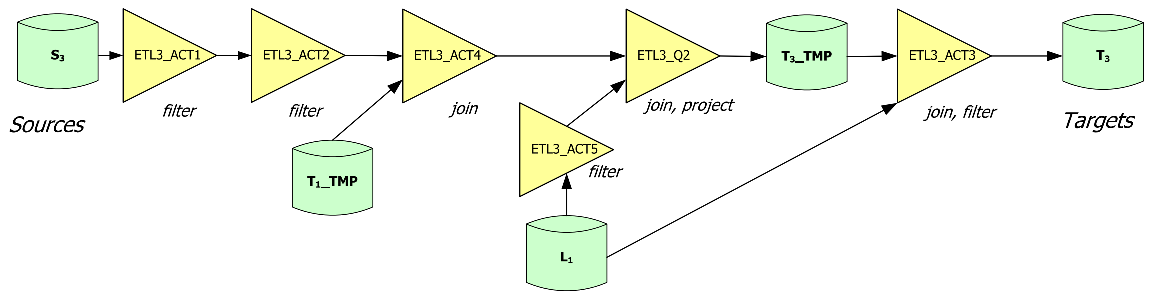


Fig. 10 ETL3 workflow

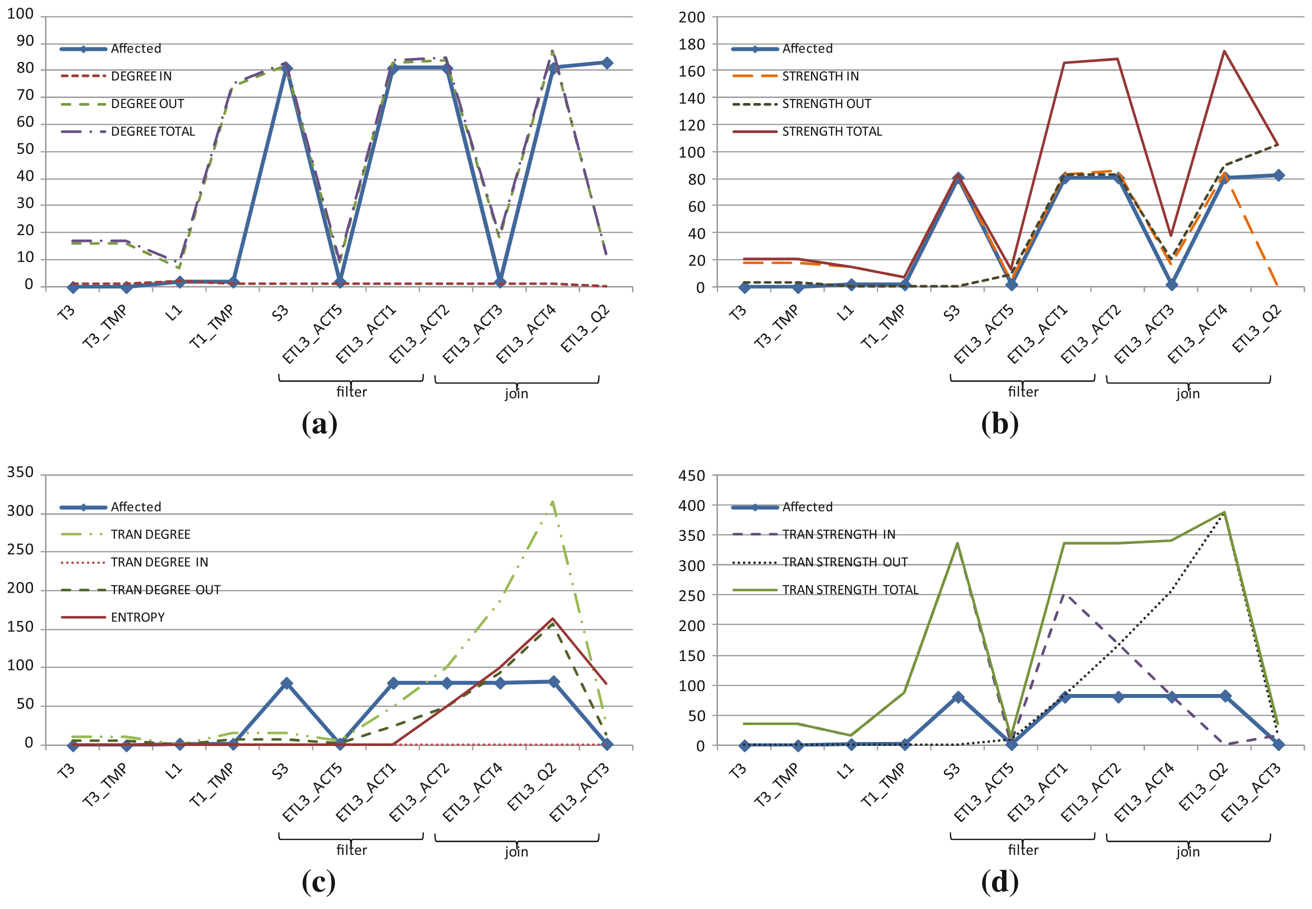


Fig. 11 Results for examined metrics for ETL3

ETL5, ETL6, and ETL7. The final three flows, ETL5, ETL6, and ETL7, are depicted in Fig. 14a, b, c, respectively. These three are similar to each other, loading data from three different source tables, namely S5 (with 13 attributes in its schema), S6 (with 7 attributes in its schema), S7 (with 4 attributes in its schema) to three target tables, T5, T6, and T7. T1_TMP table is used as a lookup table.

The results for these ETL flows are shown in Figs. 15, 16 and 17. The out-degree of relations follows proportionally the number of occurred events on them, except for S7, which unusually exhibits a high number of events with respect to its size. Activities in all three flows show similar behavior, where out degrees and strengths seem to provide more accurate results for the possible events on them (even though the sample of occurred events is low for these flows and affected series has very low values).

5 Lessons Learned

For several months, we have observed and experimented with genuine evolution events in real-world, ETL workflows. Our

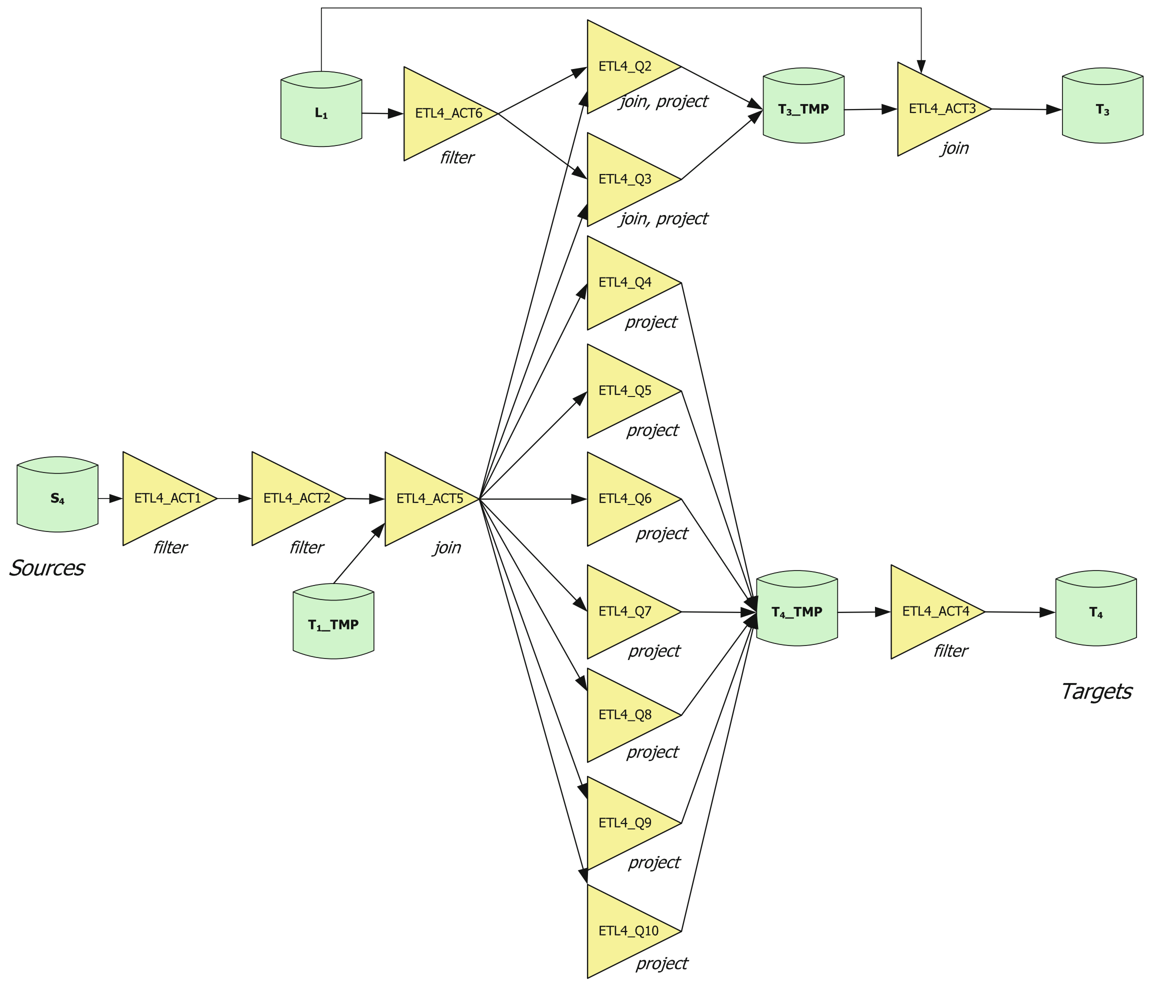


Fig. 12 ETL4 workflow

experimental findings are reported in Sect. 4. In the past, we did a similar exercise for a set of artificially created evolution scenarios [29]. Based on both activities, we assessed the metrics for predicting the behavior of a software system to evolution operations described in Sect. 3 and came up with some interesting insights regarding the value of our metrics. Next, we discuss our observations and suggest design strategies based on the results obtained by the use of these metrics.

Observations Our first observation, which is in accordance with intuition too, is that an important factor for the potential evolution of the whole or a part of a system is eventually its *schema size*. Especially in a workflow setting like the ETL environment, source or intermediate tables comprising many attributes in their schema are more likely to be altered and hence, more likely to affect the workflow they feed. Therefore, a particularly handy metric for the evaluation of the evolution potential of a workflow source table is practically the number of attributes it has (expressed by the *out-degree* metric in our setting). In terms of design, although it is often hard to change a source table, at least, the designer may choose to use intermediate tables with smaller schema sizes. For example, instead of just saving a snapshot of a table, she should try storing only its most valuable projection or instead of using a combination of production keys along with their origins, she should replace keys-origin pairs with surrogate keys.

Of course, in practice, looking just at a module size is not a panacea. There are counterexamples too, like in the case of the source table S7 in ETL7, where the number of evolution events is disproportional to the table size. Due to these cases, common practices like simple examination, through a set of standard queries, of the DB catalog tables do not suffice to get the most interesting metrics. Such cases are only identified with rigorous experimentation and for that, we need a well-structured set of metrics (like the ones presented in Sect. 3) to avoid dealing with exponentially perplexing situations as the project complexity increases.

Based on the results reported in Sect. 4, we observed that the most accurate and suitable metrics for all module types are the *out-degree* and *out-strength* metrics. On the one hand,

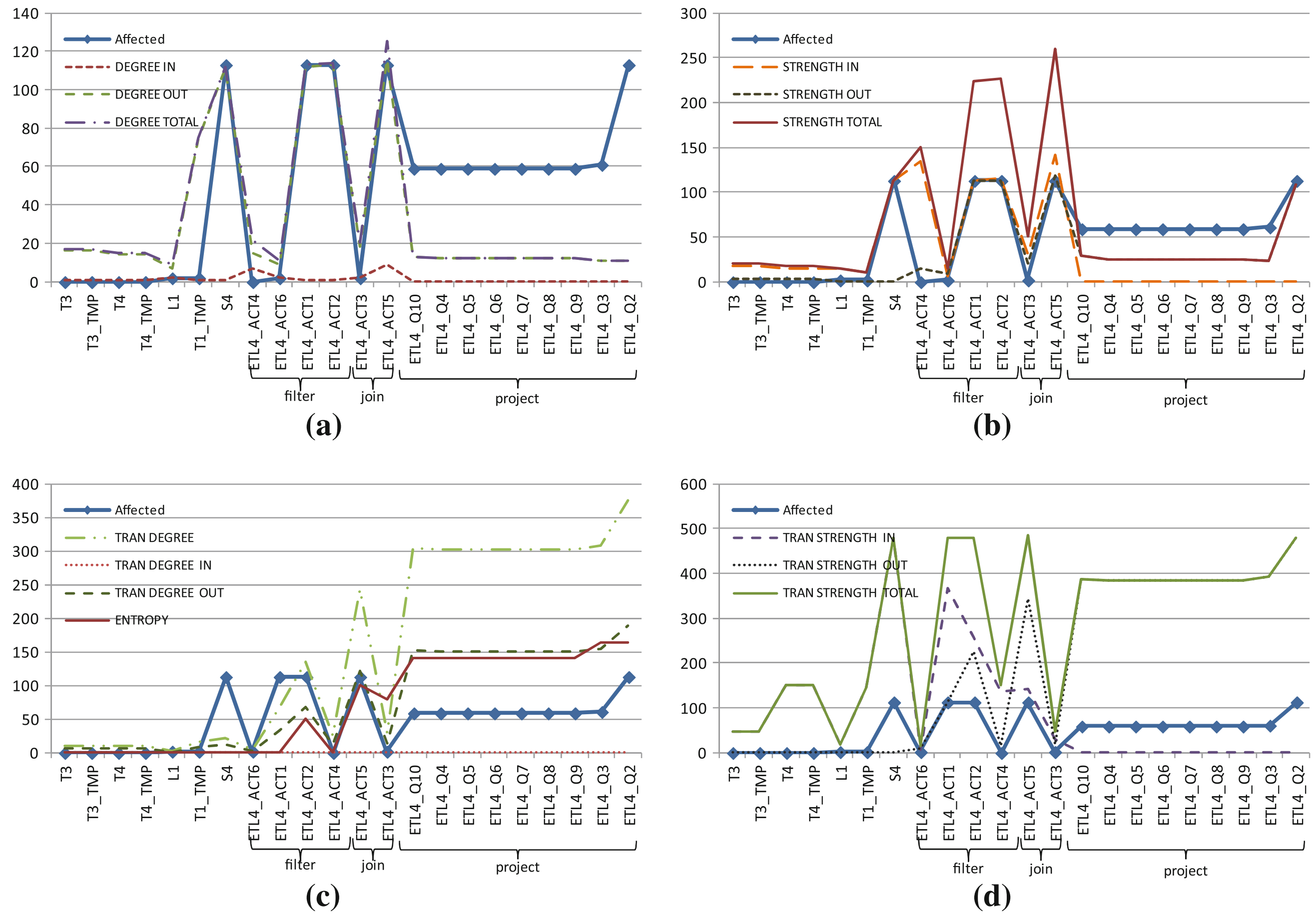


Fig. 13 Results for examined metrics for ETL4

when a module has only one provider, then it is safer to take into account the out-degree/out strength metrics, which capture the dependencies with an adjacent module. On the other hand, transitive degree metrics may act as predictors for the evolution of a module when it has many different providers and paths to evolving sources like for example Q2 and Q3 queries in ETL4 (see Fig. 12c, d) or ETL1_ACT10 - ETL1_ACT12 in ETL1 (see Figs. 6, 7).

Therefore, another observation is that the internal structure of each activity plays a significant role for the impact of evolution events on it. Activities with high *out-degree* and *out-strengths* tend to be more vulnerable to evolution. For example, such activities may project or use in conditions, a large number of attributes from their sources (either previous activities or tables). The out-degree captures the projected attributes by an activity, whereas the out-strength captures the total number of dependencies between an activity and its sources. Activities with joins between many sources tend to be more affected than activities sourced by only one provider, but still, the most decisive factor seems to be the activity size. Thus, activities that perform an attribute reduction on the workflow through either a group-by operation or a projection of a small number of attributes are in general, less vulnerable to evolution events and propagate the impact of evolution further away on the workflow (e.g., Q4 in ETL1 or Q2–Q10 in ETL4). In contrast, activities that perform join and selection operations on many sources and result in attribute preservation or generation on the workflow have a higher potential to be affected by evolution events (e.g., observe the activities ETL1_ACT10–ETL1_ACT12 in Fig. 4 or the activity ETL4_ACT5 in Fig. 13a).

Out transitive degree metrics capture the dependencies of a module with its various non-adjacent sources. These metrics exhibit more valuable results for activities, which act as "hubs" of various different paths from sources in complex workflows. For cases where the out-degree metrics *do not* provide a clear view of the evolution potential of two or more modules, the out-transitive degree and entropy metrics may offer a more adequate prediction (as for example ETL4_Q3 and ETL4_Q2 in Fig. 7a, d).

Hence, the module-level design of an ETL workflow is another crucial factor for the overall impact of evolution on the whole workflow. Thus, in terms of design, since attri-

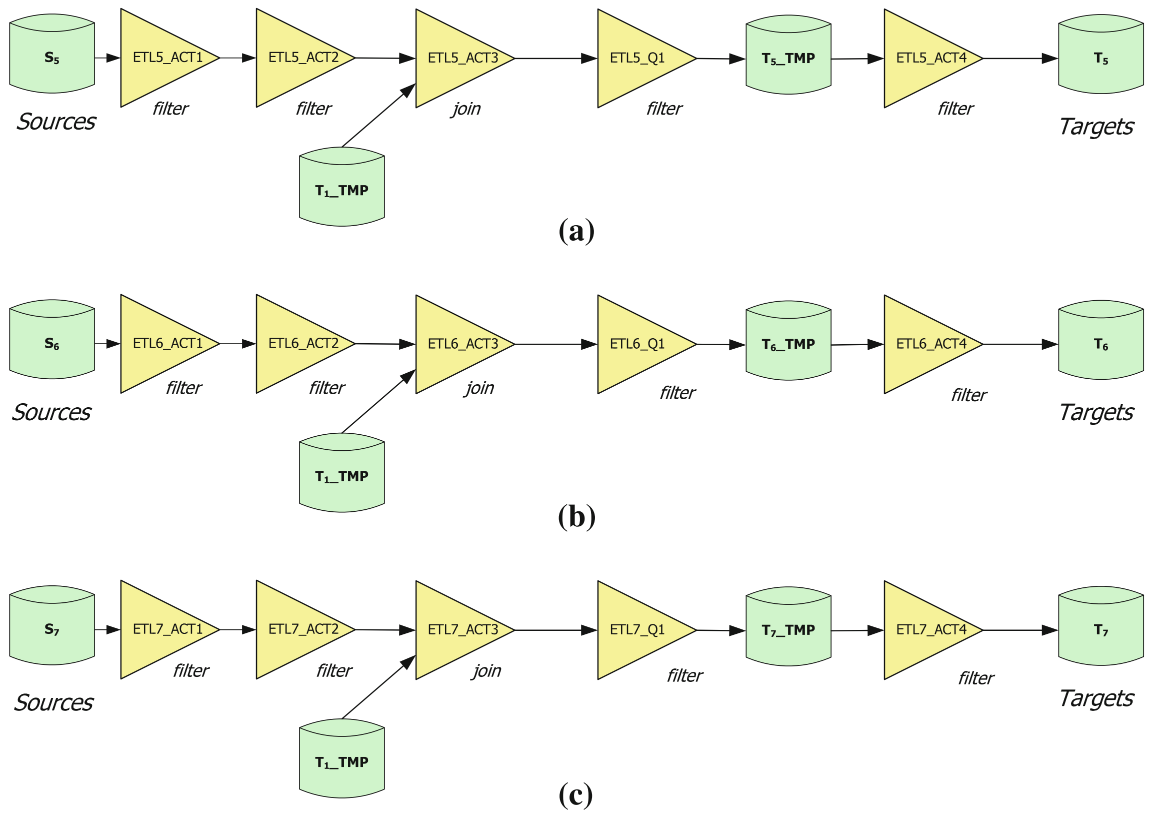


Fig. 14 **a** ETL5, **b** ETL6, **c** ETL7 workflows

bute reduction activities (e.g., projections, group by queries) are less likely to be affected by evolution actions than other activities that retain or increase the number of attributes in the workflow (many projections with joins), the ETL designer should attempt placing the attribute reduction activities in the early stages of the workflow in order to restrain the flooding of evolution events. In a way, that is in accordance with typical performance optimization strategies, where the most selective operations should be pushed toward the start of the flow.

Heuristics Based on these observations, we identify the most suitable metrics for each type of construct and provide possible optimization heuristics for reducing the maintenance effort. Table 6 reflects our observations. When persistent data stores are involved, the generic guideline is to retain their schema as small as possible. Since the schema size affects a lot the propagation of evolution events, it is advisable to reduce schema sizes across the ETL flow, so activities that help in that direction should be considered first. In addition, based on our discoveries related to what metric is suitable for each construct, e.g., transitive degree metrics are good predictors for modules with many providers and the out-degree and out strength metrics could be used for modules with a single provider.

Discussion Finally, we discuss how our methods and results may be used elsewhere and how such design choices may affect other ETL optimization objectives.

Generalization of results. Our analysis is based on a specific case study and the extent of how much this is representative is hard to show. However, in a previous work, we had presented a benchmark for ETL designs, where we presented a set of frequently used ETL template designs like butterfly, tree, fork, primary flow, and so on [35,36]. Interestingly, the designs in our case study resemble either those template designs or a combination of those. In particular, ETL1 is a complex *butterfly*-like design, ETL2 and ETL3 are *tree* designs, ETL4 is a combination of *fork* and *tree* designs, and ETL5, ETL6, and ETL7 are *primary flow* designs. Hence, we believe that the results obtained in this study may serve as general hints in other ETL projects as well.

Optimization trade-offs. In general, the aforementioned guidelines that favor maintainability of ETL flows contradict the normal practice for improving ETL performance. For example, when we have source data stores with large schema sizes, from an evolution handling perspective, it makes sense to split the schema into smaller chunks. How to efficiently do this for not hurting performance much (e.g., for avoiding join operations later on) is an open and challenging research

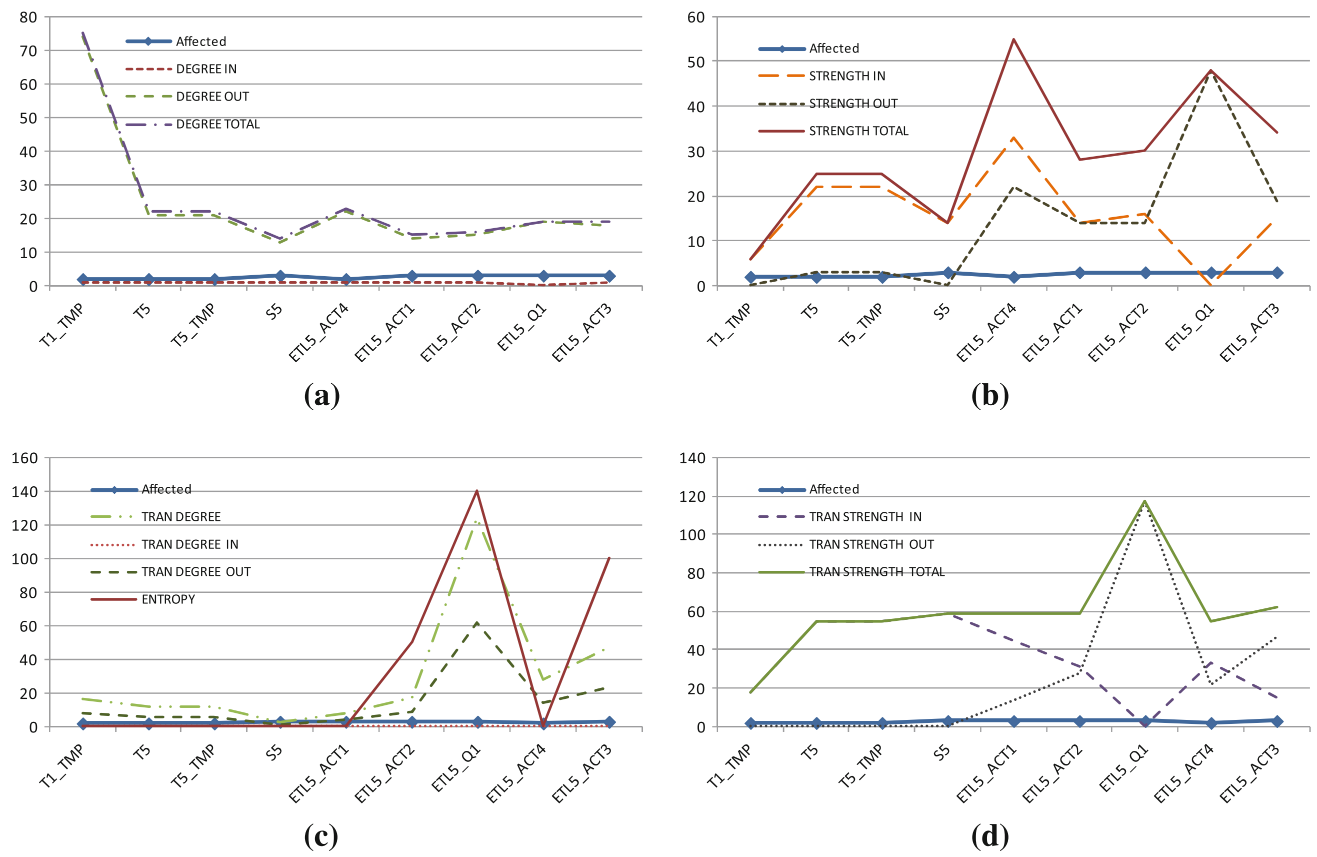


Fig. 15 Results for examined metrics for ETL5

question. As another example trade-off, although an attribute reduction operation like aggregation seems to better be placed at the beginning of a flow in terms of maintainability, in general, it should not be placed early in the flow when performance is considered. For performance, it makes sense to have early in the flow tuple reduction operators and heavier operations like aggregations should be placed next. Clearly, the ETL architect has to deal with an interesting trade-off between maintainability and performance, two very important quality factors for the design of an ETL system. However, this topic is out of the scope of this paper. Preliminary efforts towards such a multi-objective optimization of ETL flows has been presented elsewhere (e.g., [35–37]).

Usage in ETL engines Ideally, database administrators and ETL designers can employ these metrics for detecting, evaluating, and most importantly, experimenting with the design properties of ETL flows with respect to evolution. Based on such an analysis, the designer may decide to modify an ETL design or choose among more than one design for improving the maintainability of her system. In addition, the metric suite that we propose may be incorporated to an existing ETL tool for facilitating the ETL design. Since the most popular ETL tools already represent an ETL flow as a graph, measuring and predicting the evolution impact with metrics as those proposed in this work, is a realistic goal. Alternatively, the metric suite may be used as a basis of an external module—like our home-grown tool, Hecataeus—that could connect to an ETL tool. Then, based on such measures, the ETL designer could be notified about possible actions.

6 Related Work

Various approaches exist in the area of database metrics. Most of them attempt to define a set of database metrics and map them to abstract quality factors, such as maintainability, good database design, and so on. According to the model in which they are applied, we can categorize these efforts into conceptual metrics referring to the conceptual design of the database (i.e. ER diagram), relational metrics referring to the logical design of the database (i.e. relational data diagram), multidimensional metrics evaluating the design of data warehouses, information-theoretic approaches, etc.

Conceptual metrics are useful for evaluating quality issues for a database in the early stage of the design. To summarize the motivation for conceptual-level metrics, a “good” design

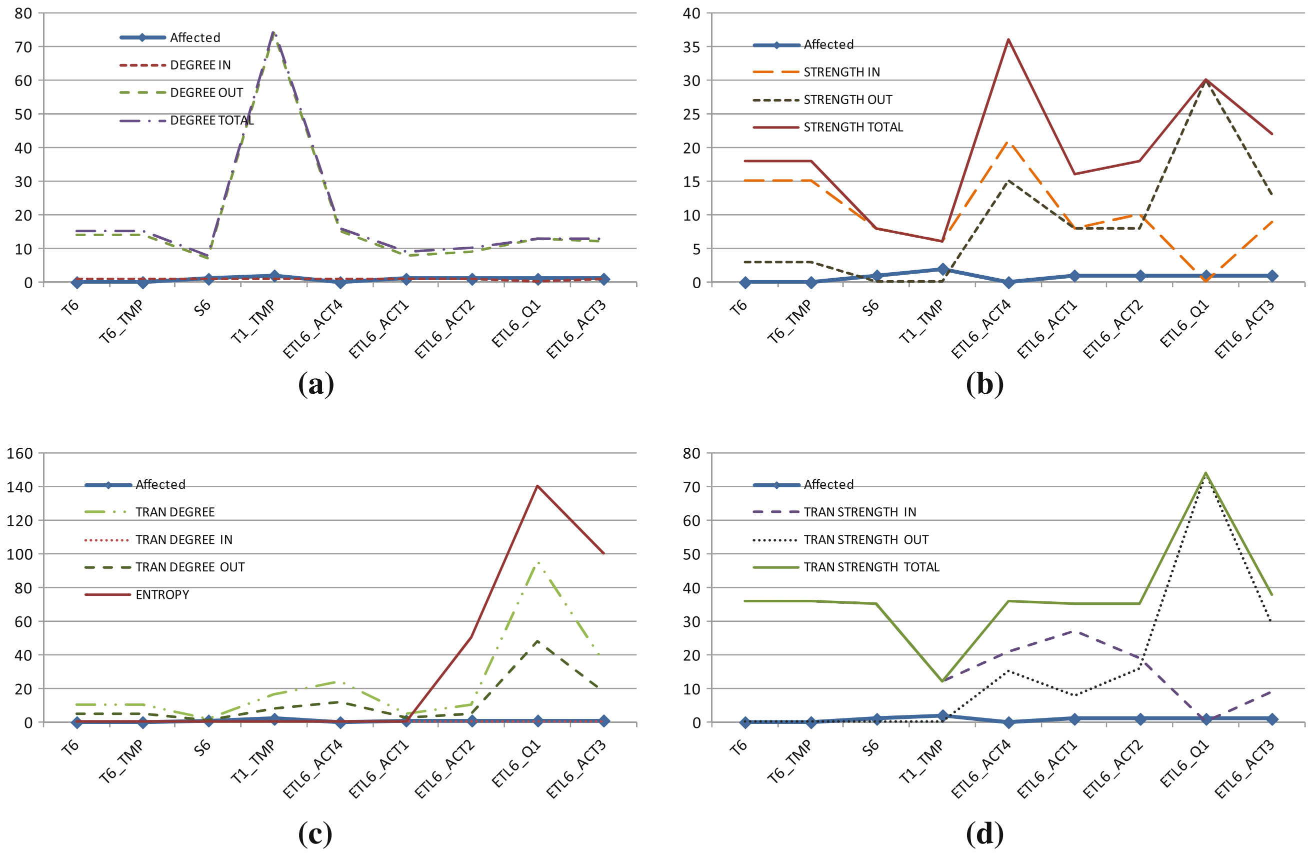


Fig. 16 Results for examined metrics for ETL6

Table 6 Metrics and heuristics for different types of ETL construct

ETL construct	Most suitable metric	Heuristic
Source tables	$D^O(v)$	Retain small schema size
Intermediate & target tables	$D^O(v)$	Retain small schema size in intermediate tables
Filtering activities	$D^O(v)$, $D^{Os}(v)$	Retain small number of conditions
Join activities	$D^O(v)$, $D^{Os}(v)$, $TD^O(v)$, $TD^{Os}(v)$, $H^s(v)$	Move to early stages of the workflow
Project activities	$D^O(v)$, $D^{Os}(v)$, $TD^O(v)$, $TD^{Os}(v)$, $H^s(v)$	Move attribute reduction activities to early stages of the workflow and attribute increase activities to later stages

at the conceptual level of a database may assure that fewer inconsistencies will emerge (mainly in terms of fundamental violations, e.g., primary and foreign keys) and furthermore fewer changes are needed during the lifetime of the information system, in general. In one of the early works, Gray et al. [16] propose two objective and open-ended metrics, namely ER metric and Area metric, to evaluate the quality of an ER diagram. ER Metric is a measure of the complexity of an ERD, based on the number of relationships between entities and Area metric is a measure of the compliance of an ERD with the corresponding ERD in 3rd Normal Form. Kesh [21] develops a method for assessing the quality of an ERD, based on both ontological and behavioral components. Ontological components are distinguished into structure and content metrics. Structure metrics are suitability, soundness, consistency, and conciseness, whereas content metrics are completeness, cohesiveness, and validity. Behavioral components are considered to be usability (from the user's point of view), usability (from the designer's point of view), maintainability, accuracy, and performance. Moreover, Moody [25] proposes a data model quality evaluation framework, which can be applied to a wide range of organizations. The proposed framework comprises a set of eight quality factors (completeness, integrity, flexibility, understandability, correctness, simplicity, integration, and implementability) which can be considered as properties of a data model with positive and negative interactions with each other. They are, in turn, evaluated by a set of 25 *quality metrics*. The quality

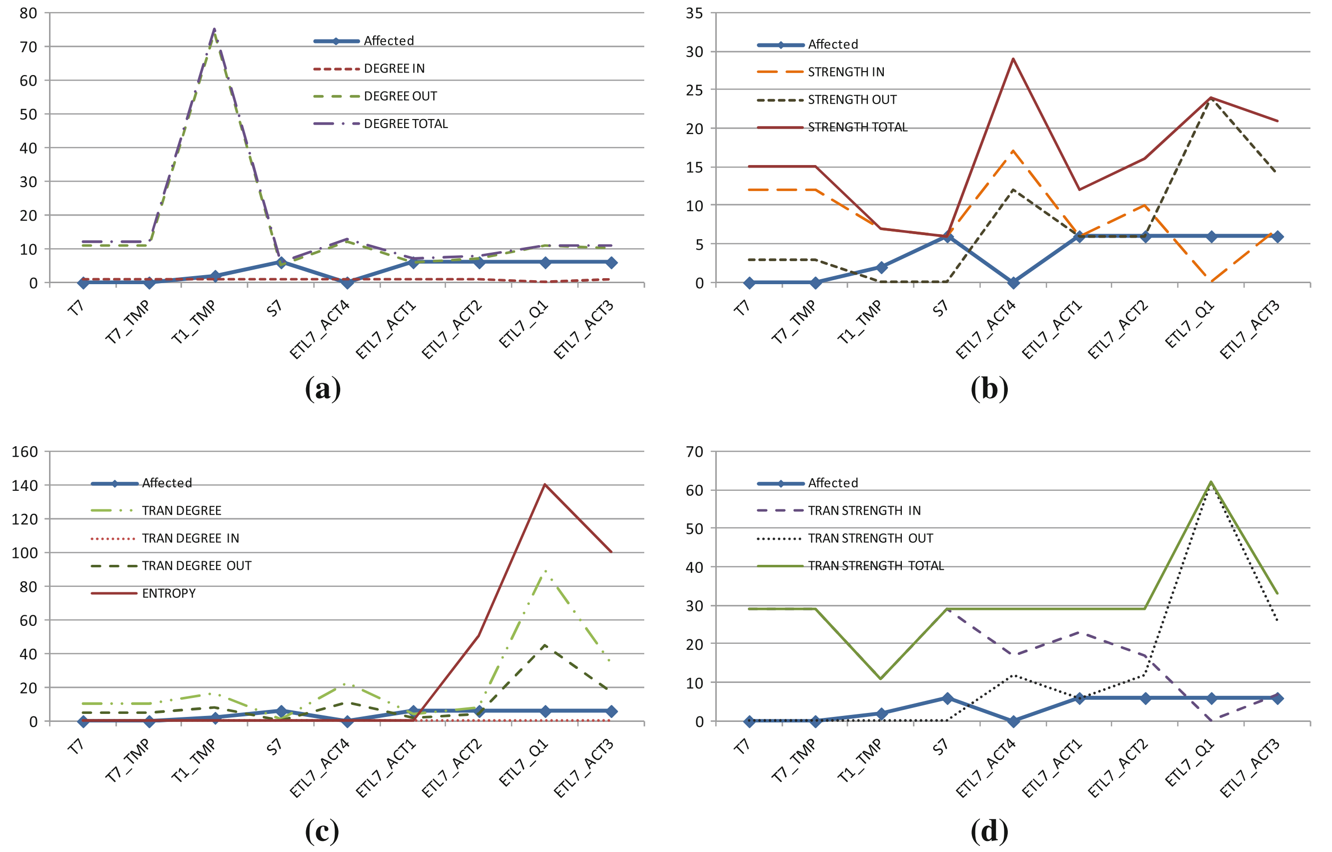


Fig. 17 Results for examined metrics for ETL7

factors may contribute to the overall quality of the system according to *weights*, which determine the importance of each factor in a problem situation. Genero et al. [13] focus on measuring the maintainability of ER diagrams through evaluating their structural complexity. They introduce a set of open-ended metrics and classify them into three main categories: entity metrics (i.e., number of entities within an ERD), attribute metrics (i.e., number of attributes within an ERD, number of composite attributes, etc.), and relationship metrics (i.e. number of M:N relationships, etc.). Wedemeijer [41] proposes a metric set for evaluating the stability capabilities of conceptual data model. The author sets up a framework for stability of conceptual schemas and proceeds to develop a set of metrics from it. The metrics are based on measurements of conceptual features, such as the number of conceptual constructs affected by a change, the complexity of a conceptual schema, the abstraction of a conceptual schema, etc. Last, Berenguer et al. [4] present a set of quality indicators and metrics for conceptual models of data warehouses. They employ UML diagrams for modeling multidimensional databases and in this context they define metrics for capturing diagram's properties such as number of packages in a diagram, number of relationships between two packages, etc. Although they provide a methodology for theoretically validating the proposed metric set, they do not present an empirical validation.

Relational database metrics are used as measures for the quality of a database at the logical level. Relational metrics are used to measure internal characteristics and structures of a database, such as tables, foreign keys, and so on. Normalization theory can give the guidelines for designing a database, but still cannot address other quality issues, such as the maintainability—or evolution—of a database. In [7,8,33], the authors propose a set of metrics for relational databases that focuses on assessing maintainability issues in a database, such as analyzability, testability, stability, and changeability. These are the number of relational tables (NT) in the database, the number of foreign keys (NFK), total number of attributes (NA), and the depth of referential tree (DRT), which is the maximum distance from a table towards another table through referential integrity constraints). Analyzability is proportionally correlated to NT, NA, DRT and NFK, changeability to NT, testability to NT, NA, and NFK, whereas stability is correlated to NT in an inverse relationship. Last, in [27], the authors propose a set of quality metrics, defined at four granularity levels (database, relation, attribute, and value) that measure referential completeness and consistency.

Data warehouse metrics. Quality in the context of data warehouses (DW) has been studied in [13,38]. The authors propose mathematical techniques for measuring or optimizing certain aspects of DW quality and adapt the Goal-Question-Metric approach from software quality management to a metadata management environment to link these special techniques to a generic conceptual framework of DW quality. Similar to aforementioned approaches DW quality is classified into several quality dimensions according to the stakeholders (e.g., data warehouse administrator, the programmer, and the decision maker) that are typically interested in them; each of these dimensions is further mapped to sample types of measurement (metrics), which help to establish the quality of a particular DW component with respect to a particular quality dimension. Various types of measurements are introduced to evaluate these dimensions. For example, the administrator is interested in the completeness dimension which concerns the preservation of all the crucial knowledge in the data warehouse schema (model), which is further quantified by the number of missing entities in the DW schema with respect to the conceptual model. Another approach to DW quality metrics is presented in [7,8]. They elaborate three kinds of metrics: table metrics regarding only table characteristics of the database (e.g. number of attributes, number of foreign keys), star metrics regarding only multidimensional characteristics of the database (e.g. number of dimension tables), and last, schema metrics regarding characteristics of the whole database schema (e.g. number of fact tables, number of overall dimension tables).

Data Warehouses and their Evolution. Concerning the evolution of data warehouses, we refer the reader to an excellent survey of [42]. A distinct line of work concerns multiversion data warehousing: see for example, Wrembel and Morzy [43] that handle the evolution of data warehouses via multiple versions and [2] that handles the problem via nested transactions, or Golfarelli et al. [14] for cross-version querying. Another excellent survey by [15] on temporal data warehousing contains a summary of the related work along these lines. A survey by [40] discusses the area of ETL and the related work.

Various *information-theoretic metrics* exist in software engineering for evaluating the quality of software design [1, 18,22]. In the data management field, an information theoretic approach to evaluating the design quality of data warehouses is presented in [23], where the relation between entropy and redundancy in the context of data warehouses is studied. They show that the redundancy in the snowflake join of the primary key of the fact table is zero, i.e. it is minimal. They define a new normal form, namely SSNF—Snowflake Schema Normal Form, justifying it in terms of entropy-based equations.

Most of the aforementioned approaches consider design metrics that correlate structural properties of the database schema to abstract quality factors. However, they confine themselves to constructs internal to the database without taking into account the incorporation of constructs surrounding the database. To the best of our knowledge this is the first set of design metrics that are explicitly targeted towards the assessment of evolution ability of the design of a data-centric ecosystem as a whole to evolutionary processes.

Formally specified frameworks. Several *software quality metrics* have been introduced in the software engineering community. Software measurement is a well-established research area that has been explored under many different programming paradigms (e.g., procedural, object oriented, service oriented, etc.) and for various stages of the lifecycle of software development (i.e., requirements analysis, design, coding, testing and maintenance). A detailed presentation of software metrics, software quality factors, and measurement approaches is out of the scope of this paper and can be found in [12]; still, we mention here the concepts of module cohesion and coupling that are mostly used for assessing the maintainability of software [24] along with complexity, as well (referred as the 3 ‘c’-s in [34]).

Briand et al. [6] employed measurement theory to provide a set of five generic categories of measures for software artifacts. In a previous work, we made a first attempt to relate these families of measures to ETL flows [39]. In that work, we used a different model for module representation (based on LDL) and formally proved that the measures proposed respect the properties of the framework by [6]. However, representing ETL activities with LDL does not scale well in terms of operations that can be supported. In addition, a typical, modern ETL flow involves operations implemented in different environments and runs on different engines (e.g., operations in Java, PL/SQL, SQL, Perl, Awk, etc. that may run in different engines like DBMS, ETL, Map-Reduce, and so on). However, independently of the internal representation, our graph-based model for data-centric systems, such as ETL flows, can be viewed as a modular system, with queries, views and relations being its building blocks encapsulating data and business logic. We generalize thus the discussion, to highlight how our method fits within a more formally specified framework like the one by [6]. First, we start by referring to the involved measures, which are

- *Size*, referring to the number of entities that constitute the software artifact; we assess the size of a (sub)graph by the number of its nodes.
- *Length*, referring to the longest path of relationships among these entities, which we assessed by the maximum transitive dependency of a module’s node.
- *Complexity*, referring to the amount of inter-relationships of a component, which we assessed measured by the number of internal edges plus the 50 % of the strength of the module.

- *Coupling*, capturing the amount of interrelationships between the different modules of a system, which we assessed by the strength of a module.
- *Cohesion*, measuring the extent to which each module performs exactly one job, by evaluating how closely related are its components, which we assessed as the fraction of the input/output nodes of a module related to some internal function of the module.

Some (but not all) of these metrics are straightforwardly related to the metrics used in this study. The transitive degree is an attempt to test the sensitivity of nodes depending on the length of their provider path. The coupling of modules within the flow is probably the most straightforward measure relating to the strength of the module.

Cohesion refers to how much interrelated are the constituents of a module. If a module performs more than one "job" and its constituents are divided in two groups of nodes doing different things, then a module is not cohesive. Still, related to data-centric software, cohesion is a weakly definable notion. For example, how can one *intuitively* convince on a model for the cohesion of a relation? In a more subtle line, even if we adopt the idea that each selection and each group-by constitutes a different "job" how convincing is it to assume that a query combining several selections and/or a group by is not cohesive?

A serious observation (that goes well beyond the scope of a case study) is that the Briand et al. meta-measures were originally thought towards traditional imperative/object-oriented software with loops and control structures rather than data-centric software. In [39], the authors provide measures for the Briand et al. [6] classification, but they do not fit important measures into the framework, like the maximum path over the graph or the degree of an individual attribute of a relation. Complexity is a good example where the respective notion in traditional software (McCabe's cyclomatic complexity) is not adequately mapped to a measure for data intensive software.

Hence, overall, we find that our most valuable metric, out-degree, which is going down to the details of individual attributes, does not fit well with the Briand et al. framework. At the same time, although module coupling is smoothly covered by strength, cohesion and complexity must be re-evaluated when we think of data-centric software.

7 Conclusions

In this paper, we have presented a real-world case study of data warehouse evolution for exploring the behavior of a set of metrics that (a) monitor the vulnerability of warehouse modules to future changes and (b) assess the quality of various ETL designs with respect to their maintainability. We have described first our graph-theoretic model for capturing the evolution impact in the ETL ecosystem, and then, we presented a detailed description of our metric suite. Finally, we have reported on our exhaustive, 6-month experimentation with real-world evolution scenarios affecting seven ETL workflows.

We have identified the schema size and module complexity as two important factors for the vulnerability of a system. We have observed that out-degrees help as predictors for the source tables; the out-degree and out-strength are very good predictors for the evolution of views; out transitive degree and entropy may be applied for queries in addition to the aforementioned metrics. Based on our experiments, we have compiled a list of lessons learned regarding the evolution behavior of an ETL environment with respect to the schema of the source tables, its constituent activities, and its overall design. We believe that these metrics and the lessons learned in this paper can be practically useful for database administrators and designers for detecting vulnerable parts and evaluating the design properties of data-centric ecosystems, like ETL workflows, with respect to evolution.

Coming back to our starting point, have we answered the fundamental questions like *How good is an ETL design?* and *What makes an ETL design good or bad?*. We have demonstrated only ways to predict vulnerability to change and discussed some interrelationship with other aspects, like for example, performance. A complete answer to the above questions and an attempt to combine different aspects of the environments design (performance, vulnerability to change, understandability, etc.) in a comprehensive framework are prominent directions for future research. Another direction for future work concerns models that are not founded on a graph-based model. Although our approach is founded on a simple and intuitive graph representation of modules and their dependencies, it is quite possible that other approaches that avoid the translation of code to graphs can be pursued. How this can be done and what is the effect to the metrics used are open problems.

Acknowledgments We would like to thank the anonymous reviewers of a first version of this paper for their constructive comments that improved the clarity and completeness of the paper.

References

1. Allen EB (2002) Measuring graph abstractions of software: an information-theory approach. In: Proceedings of the 8th international symposium on software metrics (METRICS'02)
2. Bebel B, Królikowski, Z, Wrembel R (2006) Managing evolution of data warehouses by means of nested transactions (ADVIS'06)
3. Bellahsene Z (2002) Schema evolution in data warehouses. Knowl Inf Syst 4(2):283–304

4. Berenguer G, et al (2005) A set of quality indica-tors and their corresponding metrics for conceptual models of data warehouses. In: 7th International conference on data warehousing and knowledge discovery (DaWaK'05)
5. Blaschka M, Sapia C, Höfling G (1999) On schema evolution in multidimensional databases. In: 1st International conference on data warehousing and knowledge discovery (DaWaK'99)
6. Briand LC, Morasca S, Basili VR (1996) Property-based software engineering measurement. IEEE Trans Softw Eng 22(1):68–85
7. Calero C, Piattini M, Genero M (2001) Empirical validation of referential integrity metrics. Inf Softw Technol 43(15):949–957
8. Calero C, Piattini M, Pascual C, Serrano M (2001) Towards data warehouse quality metrics. In: Proceedings of the 3rd international workshop on design and management of data warehouses (DMDW'01)
9. Cleve A, Brogneaux A, Hainaut J (2010) A conceptual approach to database applications evolution. In: Proceedings of the 29th international conference on conceptual modeling (ER'10)
10. Fan H, Poulovassilis A (2004) Schema evolution in data warehousing environments—a schema transformation-based approach. In: Proceedings of the 23rd international conference on conceptual modeling (ER'04)
11. Favre C, Bentayeb F, Boussaid O (2007) Evolution of data warehouses' optimization: a workload perspective. In: 9th International conference on data warehousing and knowledge discovery (DaWaK'07)
12. Fenton NE, Pfleeger SL (1998) Software metrics: a rigorous and practical approach, revised 2nd edn. PWS Publishing Co.
13. Genero M, Piattini M, Calero C, Serrano M (2000) Measures to get better quality databases. In: Proceedings of the 2nd international conference on enterprise information systems (ICEIS'00)
14. Golfarelli M, Lechtenbörger J, Rizzi S, Vossen G (2006) Schema versioning in data warehouses: enabling cross-version querying via schema augmentation. Data Knowl Eng 59(2):435–459
15. Golfarelli M, Rizzi S (2009) A survey on temporal data warehousing. In: Database technologies: concepts, methodologies, tools, and applications, pp 221–237
16. Gray R, Carey B, McGlynn N, Pengelly A (1991) Design metrics for database systems. BT Technol J 9(4):69–79
17. Gupta A, Mumick IS, Rao J, Ross KA (2001) Adapting materialized views after redefinitions: techniques and a performance study. Inf Syst 26(5):323–362
18. Harrison W (1992) An entropy-based measure of software complexity. IEEE Trans Softw Eng 18(11):1025–1034
19. Inmon WH (2000) The data warehouse budget. White paper
20. Jarke M, Jeusfeld MA, Quix C, Vassiliadis P (1999) Architecture and quality in data warehouses: an extended repository approach. Inf Syst 24(3):229–253
21. Kesh S (1995) Evaluating the quality of entity relationship models. Inf Softw Technol 37(12):681–689
22. Kim K, Shin Y, Wu C (1995) Complexity measures for object-oriented program based on the entropy. In: Proceedings of the 2nd Asia-Pacific software engineering conference (APSEC '95)
23. Levene M, Loizou G (2003) Why is the snowflake schema a good data warehouse design?. Inf Syst 28(3):225–240
24. Lorenz M, Kidd J (1994) Object-oriented software metrics. Prentice Hall, Englewood Cliffs
25. Moody DL (1998) Metrics for evaluating the quality of entity relationship models. In: Proceedings of the 17th international conference on conceptual modeling (ER'98)
26. Nica A, Lee AJ, Rundensteiner EA (1998) The CSV algorithm for view synchronization in evolvable large-scale information systems. In: Proceedings of the 6th international conference on extending database technology (EDBT'98)
27. Ordonez C, García-García J (2008) Referential integrity quality metrics. Decis Support Syst 44(2):495–508
28. Papastefanatos G, Vassiliadis P, Simitsis A, Vassiliou Y (2008) Design metrics for data warehouse evolution. In: Proceedings of the 27th international conference on conceptual modeling (ER'08)
29. Papastefanatos G, et al (2008) Language extensions for the automation of database schema evolution. In: Proceedings of the 14th international conference on enterprise information systems (ICEIS'08)
30. Papastefanatos G, Vassiliadis P, Simitsis A, Vassiliou Y (2009) Policy-regulated management of ETL evolution. J Data Semantics 13:147–177
31. Papastefanatos G, Vassiliadis P, Simitsis A, Vassiliou Y (2010) HECATAEUS. Regulating schema evolution. In: Proceedings of the 26th IEEE international conference on data engineering (ICDE'10)
32. Papoulis A (1990) Probability & statistics. Prentice Hall, Englewood Cliffs
33. Piattini M, Genero M, Calero C (2001) Table oriented metrics for relational databases. Softw Quality J 9(2):79–97
34. Pressman RS, Ince D (2000) Software engineering (a practitioner's approach), 5th edn. European Adaptation. McGraw Hill
35. Simitsis A, Vassiliadis P, Dayal U, Karagiannis A, Tziovara V (2009) Benchmarking ETL workflows. In: Proceedings of the TPC technology conference (TPCTC'09)
36. Simitsis A, Wilkinson K, Castellanos M, Dayal U (2009) QoX-driven ETL design: reducing the cost of ETL consulting engagements. In: Proceedings of the 35th SIGMOD international conference on management of data (SIGMOD'09)
37. Simitsis A, Wilkinson K, Dayal U, Castellanos M (2010) Optimizing ETL workflows for fault-tolerance. In: Proceedings of the 26th IEEE international conference on data engineering (ICDE'10)
38. Vassiliadis P, Bouzeghoub M, Quix C (2000) Towards quality-oriented data warehouse usage and evolution. Inf Syst 25(2):89–115
39. Vassiliadis P, Simitsis A, Terrovitis M, Skiadopoulos S (2005) Blueprints and measures for ETL workflows. In: Proceedings of 24th international conference on conceptual modeling (ER 2005), 24–28 Oct 2005, Klagenfurt, Austria
40. Vassiliadis P (2009) A survey of extract–transform–load technology. Int J Data Warehousing Mining 5(3):1–27
41. Wedemeijer L (2000) Defining metrics for conceptual schema evolution. In: Proceedings of the 9th international workshop on foundations of models and languages for data and objects (FMLDO'00)
42. Wrembel R (2009) A survey of managing the evolution of data warehouses. Int J Data Warehousing Mining 5(2):24–56
43. Wrembel R, Morzy T (2006) Managing and querying versions of multiversion data warehouse (EDBT'06)