A Query Answering System for Data with Evolution Relationships

Siarhei Bykau
University of Trento
bykau@disi.unitn.eu

Flavio Rizzolo
Statistics Canada
flavio.rizzolo@statcan.gc.ca

Yannis Velegrakis
University of Trento
velgias@disi.unitn.eu

ABSTRACT

Evolving data has attracted considerable research attention. Researchers have focused on modeling and querying of schema/instance-level structural changes, such as, insertion, deletion and modification of attributes. Databases with such a functionality are known as temporal databases. A limitation of the temporal databases is that they treat changes as independent events, while often the appearance (or elimination) of some structure in the database is the result of an evolution of some existing structure. We claim that maintaining the causal relationship between the two structures is of major importance since it allows additional reasoning to be performed and answers to be generated for queries that previously had no answers. We present the TrenDS, a system for exploiting the evolution relationships between the structures in the database. In particular, our system combines different structures that are associated through evolution relationships into virtual structures to be used during query answering. The virtual structures define "possible" database instances, in a fashion similar to the possible worlds in the probabilistic databases. TrenDS uses a query answering mechanism that allows queries to be answered over these possible databases without materializing them. Evaluation of such queries raises many technical challenges, since it requires the discovery of Steiner forests on the evolution graphs.

Categories and Subject Descriptors

H.m [**Information Systems**]: Miscellaneous

General Terms

Algorithms, Performance

Keywords

Data evolution, Probabilistic databases, Graph Algorithms

1. INTRODUCTION

Considerable amount of research effort has been spent on the development of models, techniques and tools for managing data changes. These range from data manipulation languages, and maintenance of views under changes to schema evolution and mapping adaptation. To cope with the history of data changes, temporal models have been proposed for the relational and ER models, for semi-structured data, XML and for RDF. Almost in its entirety, existing work on data changes is based on a data-oriented point of view. It aims at recording and managing changes that are taking place at the values of the data. Unfortunately, such work fails to capture the full spectrum of evolutionary phenomena. Specifically, those approaches are founded on the assumption that the nature of each real world entity represented in the database persists over time, e.g., students are added, modified, and eventually deleted, but never become professors, with a direct link between the student tuple and the professor one. As such, evolution amounts only to temporal changes of attributes/relationships. Evolution of an entity that spans different concepts [7] (e.g., student to professor, research lab to independent corporate entity) are unaccounted for. This kind of evolution finds numerous applications in many practical scenarios. For instance, in the area of Dataspaces[6], entities may split or merge, historians may model the evolution of species or the chains of human achievements, i.e., how the concept of biotechnology evolved from the agricultural technology to the modern genetics, and educators can track how courses are evolving to new modern topics or are eliminated when out-dated.

In the presence of this type of evolution, answering queries that span across multiple evolution phases is becoming a challenge. For instance, given the fact that Germany transformed from an empire into a modern country that was split to East and West and later merged into one, the query "give me all the heads-of-state of Germany between 1800 and 2000" is hard to deal with. It essentially requires hand-coding the history of Germany into several queries to be processed separately. Note that this may look similar to terminology evolution [1], i.e., using different terms to describe the same real world entity at different points in time, but it actually goes far beyond that.

In this demo, we showcase a system, called TrenDS, that supports this functionality. It achieves this in polynomial time by leveraging on the distinct properties of the evolution relationships (the detailed algorithm is presented in [3]). One of the distinct features of TrenDS is its ability to reason and generate answers that may not be explicitly recorded in the data, alongside an explanation of why such an answer is considered. The system is demonstrated on a real-case scenario that models the corporate history of companies through acquisitions, spin-offs, break-ups and merges. A motivating example for this work has been the story of the well-known AT&T Labs.

2. MOTIVATING EXAMPLE

Consider AT&T, a company that over the years has gone through a large number of break-ups, merges and acquisitions. Its famous Bell Labs where many great innovations took place, had a similar

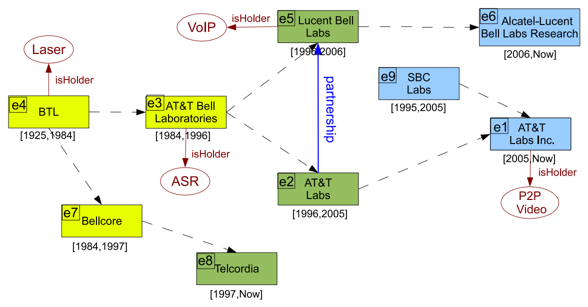


Figure 1: The history of the AT&T Labs.

fate. It was founded in 1925 under the name Bell Telecommunication Laboratories (BTL). In 1984, it was split into Bellcore (to become Telcordia in 1997) and AT&T Bell Laboratories. The latter existed until 1996 when it was split into Bell Labs, that was bought by Lucent, and to AT&T Labs. The Lucent Bell Labs became Alcatel-Lucent Bell Labs Research in 2006 due to the take-over of Lucent by Alcatel. Furthermore, due to the take-over of AT&T by SBC in 2005, the AT&T Labs were merged with the SBC Labs to form the new AT&T Inc. Labs. Despite being research labs of different legal entities, Lucent and AT&T Labs have actually maintained a special partnership relationship. All the different labs have produced a large number of inventions, as the respective patents can demonstrate. Examples of such inventions are the VoIP (Voice over Internet Protocol), the ASR (Automatic Speech Recognition), the P2P (Peer-to-Peer) Video and the laser. A graphical illustration of the above can be found in Figure 1 where the labs are modeled by rectangles and the patents by ovals.

Assume that a temporal database has been used to model this information as illustrated in Figure 1, and consider a user who is interested in finding the lab that invented the laser and the ASR patent. It is true that these two patents have been filed by two different labs, the AT&T Bell Labs and the AT&T Labs Inc. Thus, the query will return no results. However, it can be noticed that the latter entity is an evolution of the former. It may be the case that the user does not have the full knowledge of the way the labs have changed in in her own mind, the two labs are still considered the same. We argue that instead of expecting from the user to know all the details of the evolution granularity and the way the data has been stored, which means that the user's conceptual model should match the one of the database, we'd like the system to try to match the user's conceptual model. This means that the system should have the evolution relationships represented explicitly in the dataset and take them into account when evaluating a query. In particular, we want the system to treat the AT&T Bell Labs, the AT&T Labs Inc, and the AT&T Labs as one unified (virtual) entity. That unified entity is the inventor of both the laser and the ASR, and should be the main element of the response to the user's query.

The query response is based on the assumption that the user did not intend to distinguish between the three aforementioned labs. Since this is an assumption, it should be associated with some degree of confidence. Such a degree can be based, for instance, on the number of labs that had to be merged in order to produce the answer. A response that involves 2 evolution-related entities should have higher confidence than one involving 4.

3. TECHNICAL CHALLENGES

[Modeling Evolution] As a data model we have adopted an *entity-based* (or *concept-based*) model that is gaining popularity and has been the fundamental model in dataspaces. The dataspace is a well-known and highly popular data model in heterogeneous systems [6]. Its fundamental structure is the *entity* which is used to model a real world object. An entity is a data structure consisting of a unique identifier and a set of attributes. Each attribute has a name and a value. The value of an attribute can be an atomic value or an entity identifier. A *database* is a collection of entities. For a query language we adopt a datalog style language, the query language for web forms, in which a query consists of a head and a body. The body is a conjunction of atoms. An atom is an expression of the form $e(n_1{:}v_1,\ n_2{:}v_2,\ \ldots,\ n_k{:}v_k)$ or an arithmetic condition such as $=$, $\leq$, etc. The head is always a non-arithmetic atom. Given a database, the body of the query is said to be true if all its atoms are true. A non-arithmetic atom $e(n_1{:}v_1,\ n_2{:}v_2,\ \ldots,\ n_k{:}v_k)$ is true if there is an entity with an identifier e and attributes $\langle n_i, v_i\rangle$ for every $i{=}1..k$. When the body of a query is true, the head is also said to be true. A value v_i in a query may be either a constant or a variable. A *true assignment* is an assignment of the variables in a query to constant values that make the body of the query true. The answer to a query is a set of entities constructed according to the head specifications, for every true assignments of the variables in the body. Note that keyword query answering techniques are typically translate the set of keywords into our style of queries [2], but this is not the focus of our work.

EXAMPLE 3.1. *The query:*
$\$x(isHolder{:}\$y)$:-$\$x(name{:}'AT\&TLabsInc.', isHolder{:}\$y)$
looks for entities called "AT&T Labs Inc." that are holders of a patent. For every entity found in the database satisfying these conditions, an entity is created in the answer set that has an attribute isHolder with a value that is equal to the patent of the entity found in the database. ■

To model the evolution relationship we consider a special association that we elevate into a first-class citizen in the database. We call this association an *evolution relationship*. Intuitively, an evolution relationship from one entity to another is an association indicating that the real world object modeled by the later is the result of some form of evolution of the object modeled by the former. In Figure 1, the dotted lines between the entities illustrate evolution relationships. A database with evolution relationships is an *evolution database*.

Given an evolution database, one can construct a directed acyclic graph by considering as nodes the entities and as edges its evolution relationships. We refer to this graph as the *evolution graph* of the database.

[Query Answering Semantics in Evolution Databases] Our proposal is to merge entities representing different evolution phases of the same real world object into one single representation when this is going to lead to new answers to the user query. This kind of merging is called *coalescence*. The full details and formal definitions of the concept of coalescence and query answering can be found in our full papers [7, 3, 4].

Intuitively, coalescence is defined only on entities that are connected through a series of evolution relationships; the coalescence of those entities is a new entity that replaces them and has as attributes the union of their attributes (including associations).

Given an evolution database and a set of evolution relationships one can perform a series of consecutive coalescence operations, each one coalescing the two entities that an evolution relationship associates. The result of such coalescences is a new databases referred as a *possible world*. Our notion of a possible world is similar to the notion of possible worlds in probabilistic databases [5].

According to the definition of a possible world, an evolution database can be seen as a shorthand of a set of databases, each one representing a possible world. Thus, the answer of a query on an evolution database can be seen as a shorthand for the union of the answers of the evaluation of the query on every possible world.

\$x	\$y	Possible World	Answer	Cost
e1	P2P Video	∅	e1(isHolder:“P2P Video")	0
Cl(e1,e2)	**P2P Video**	**e1,e2**	**Not generated**	**1**
Cl(e1,e2,e3)	**P2P Video**	**e1,e2,e3**	**Not generated**	**2**
Cl(e1,e2,e3)	ASR	e1,e2,e3	Cl(e1,e2,e3)(isHolder:“ASR")	2
Cl(e1,e2,e3,e4)	Laser	e1,e2,e3,e4	Cl(e1,e2,e3,e4)(isHolder:“Laser")	3
…	…	…	…	…

Table 1: A fraction of variable assignments for Example 3.2.

The number of possible worlds is exponential to the number of evolution relationships, which means that generating them at run-time is practically infeasible, and pre-computing them is prohibitively space-consuming. Our solution to this is to detect at run-time only the possible worlds that are likely to generate answers, and generate only the part of these worlds that are related to the query.

Furthermore, for a given query, there may be multiple possible worlds that generate the same results. To eliminate this redundancy we require every coalescence taken into consideration for the generation of a possible world to be well-justified. In particular, our principle is that no possible world or variable assignment will be considered, unless it generates some new results in the answer set. Furthermore, among the different possible worlds that generate the same results in the answer set, only the one that requires the smaller number of coalescences will be considered.

It is natural to assume that not all possible worlds are equally likely to describe the database the user has in mind when she was formulating the query. We assume that the more a possible world differs from the original evolution database, the less likely it is to represent what the user had in mind. This is also in line with the minimality and well-justification principle described previously. We reflect this as a reduced confidence to the answers generated by the specific possible world and quantify it as a cost assigned to each answer. One way to measure that confidence is to count how many evolution relationships have to be coalesced for the possible world to be constructed. The evolution relationships may also be assigned a cost reflecting the confidence to the fact that its second entity is actually an evolution of the first.

EXAMPLE 3.2. *Table 1 illustrates a set of true variable assignments for the query of Example 3.1 on the database of Figure 1 alongside the possible world on which each assignment is illustrated (third column). The fourth column contains the result generated in the answer set from the specific assignment and the last column its respective cost. $Cl()$ is a coalescence of entities specified in the brackets. Note that the second and the third variable assignment (highlighted in bold), are redundant since they are subsumed by the first.* ∎

The existence of a cost for the different solutions, allows us to rank the query results and even implement a top-k query answering.

[Finding the best coalescence] For a given set of entities connected through evolution relationships, there may be multiple ways on how these relationships can be used to coalesce the entities into a single one. Finding these ways and deciding the best may result in an exponential cost in the size of the query and evolution graph, e.g. using the recursive SQL. However, we show that the problem boils down to finding the Steiner forest of these entities on the evolution graph and we propose a novel algorithm which is polynomial in the size of evolution graph (see [3] for details).

3.1 Evolution Query Evaluation

Our query evaluation algorithm consists of the following six steps:

[Step 1: Query Normalization] We decompose every non-arithmetic atom in the body of the query that has more than one condition into a series of single-condition atoms.

[Step 2: Individual Variable Assignments Generation] For each non-arithmetic atom in the decomposed query, a list is constructed that contains assignments of the variables in the respective atom to constants that make the atom true.

[Step 3: Candidate Assignment Generation] The elements of the lists generated in the previous step are combined together to form complete variable assignments, i.e., assignments that involve every variable in the body of the query. In particular, the cartesian product of the lists is created. Each element in the cartesian product is a tuple of assignments. By construction, each such tuple will contain at least one assignment for every variable that appears in the body of the query. If there are two assignments of the same attribute bound variable to different values, the whole tuple is rejected. Any repetitive assignments that appear within each non-rejected tuple is removed to reduce redundancy. The result is a set of variable assignments, one from each of the tuples that have remained.

[Step 4: Arithmetic Atom Satisfaction Verification] Each assignment generated in the previous step for which there is at least one arithmetic atom not evaluating to true, is eliminated from the list.

[Step 5: Candidate Coalescence Identification] Within each of the remaining assignments we identify entity-bound variables that have been assigned to more than one value. Normally this kind of assignment evaluates always to false. However, we treat them as suggestions for coalescences, so that the assignment will become a *true assignment*. In order for the assignments of variable to evaluate to true, we need to be able to coalesce the entities. To do so, these entities have to belong to the same connected component in the evolution graph of the database. If this is not the case, the assignment is ignored.

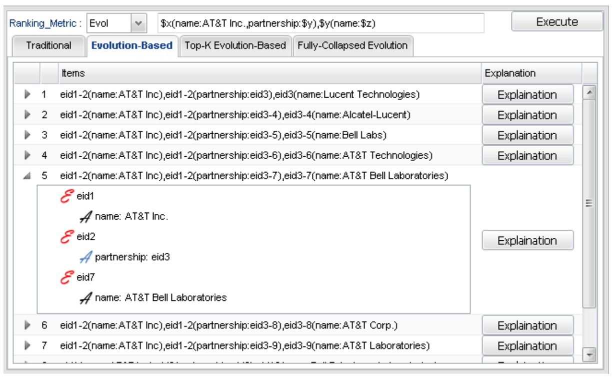
[Step 6: Coalescence Realization & Cost Computation] For every set of entities to be coalesced (obtained at step 5) we have to compute the Steiner forest, and based on that forest, generate a cost for the respective answers. To do that, we have designed and implemented an optimal Steiner Forest algorithm which is polynomial to the size of the evolution graph and exponential to the size of user's query. Note that approximate solutions of the Steiner forest problem cannot be applied because we cannot order the answers if we have approximate costs. The details of the algorithm alongside its performance evaluation can be found elsewhere [3].

4. THE TRENDS SYSTEM

We have implemented the above theory into a system called TrenDS, which is a stand-alone Java program on top of a data repository. Since the module is based on an entity-based data model, an RDF repository has been a natural choice. The evolution relationships are implemented as normal attributes in the repository, but TrenDS elevates them into first class citizens.

TrenDS contains a query execution engine module which is responsible for parsing the query and decomposed it into the right queries that need to be sent to the underlying storage system. The module is greatly assisted by a *steiner forest* component. It is the component responsible for finding the best way that different entities satisfying part of the query conditions are related through evolution relationships. The results of the queries are collected in the query execution module that combines them together in order to form the final answer.

The query execution engine operates in four modes. The first is the *traditional* mode in which no evolution relationships are considered and the answer set consists of only those data elements



$x(name:AT&T Inc.,partnership:$y),$y(name:$z)

	Items	Explanation
1	eid1-2(name:AT&T Inc),eid1-2(partnership:eid3),eid3(name:Lucent Technologies)	Explaination
2	eid1-2(name:AT&T Inc),eid1-2(partnership:eid3-4),eid3-4(name:Alcatel-Lucent)	Explaination
3	eid1-2(name:AT&T Inc),eid1-2(partnership:eid3-5),eid3-5(name:Bell Labs)	Explaination
4	eid1-2(name:AT&T Inc),eid1-2(partnership:eid3-6),eid3-6(name:AT&T Technologies)	Explaination
5	eid1-2(name:AT&T Inc),eid1-2(partnership:eid3-7),eid3-7(name:AT&T Bell Laboratories)	Explaination
6	eid1-2(name:AT&T Inc),eid1-2(partnership:eid3-8),eid3-8(name:AT&T Corp.)	Explaination
7	eid1-2(name:AT&T Inc),eid1-2(partnership:eid3-9),eid3-9(name:AT&T Laboratories)	Explaination

```
eid1
  name: AT&T Inc.
eid2
  partnership: eid3
eid7
  name: AT&T Bell Laboratories
```

Figure 2: TrenDS Graphical User Interface

that satisfy the query conditions. The second mode is the *evolution* mode in which the answer set is enhanced with answers satisfying the query conditions but do not exist in the data repository. These are data elements created through merges of existing entities in the repository that are related through evolution relationships. The third mode is the *top-k evolution* which is the same like the *evolution* mode apart from the fact that only the top-k results are returned. The sorting of the results is based on the Steiner forest that was used to associate the entities in the repository and merge them. Finally, the fourth mode, called *fully-collapsed*, executes the query against a repository where all entities connected through evolution relationships have been collapsed into one.

A screen-shot of the TrenDS is illustrated in Figure 2. It consists of a text field on which the user can write a query. On the left of the query, a pull down menu allows the user to select the method of computing the cost of result items. The main part of the screen is the list of the results found as an answer to the query. Theoretically, every entry in the results describe the binding of the query variables into values of the repository. This is the case in which no evolution information is considered. However, this is not the case if the evolution is taken into consideration. The presented entities may be virtual, i.e., created on-the-fly through a merge of entities associated with evolution relationships. As an example, consider the 5th entry in Figure 2. There is no entity `eid1-2`, neither entity `eid3-7`. The former has been constructed by the merging of the entities `eid1` and `eid2`, while the second by the merging of `eid3` and `eid7`. If the user wants to see the binding of variables to actual entities in the repository, she can do so by clicking on the entry which will open a small panel as shown in the figure. Note that TrenDS has variables bound to more than one data value, which can look strange at first sight. However, this is the feature of our approach and the actual reason that the merging of the entities on which the variable is bound is forced. In the specific example, variable $x is bound to entity `eid1` and `eid2`, while variable $y is bound to `eid3` and `eid7`. This leads to the merging (collapse) of the two former, creating the virtual entity `eid1-2` and the two latter creating entity `eid3-7`. If the user wants additionally to understand how the Steiner forest between these entities looks like, she can click on the `Explanation` button on the right of the result entry which will open a graphical illustration.

5. DEMO HIGHLIGHTS AND LESSONS

The demonstration will be based on a dataset extracted from the United States Patent and Trademark Office[1]. The trademarks are a kind of intellectual property. When there is a change of ownership, the trademark will have to re-register accordingly. Tracking the re-registration history of trademarks alongside a number of other factors can reveal important information about the history of the various companies. The dataset extracted from the UPTSO contained approximately 16,000 US companies, 200,000 attributes. In this dataset we have discovered 573 evolution graphs of various sizes between 5 and 373.

[1]http://www.uspto.gov

Evolution Discovery. Although it is not the main goal and feature of TrenDS, the participants will have the ability to explore the data set of US companies. By entering the name of a company (e.g. Oracle, Microsoft, Google), they will be able to see its evolution graph, i.e., from what companies it has been created, if any, what companies it has incorporated, what spin-offs were created by it, and to what companies it was broken-up, if any. For these evolution states, the reregistration of the trademarks will be displayed to justify the inference about its evolution. The goal of this demonstration is twofold. First to show to the user how evolution relationships can be inferred, and second, to familiarize the user with the data so that she will be able to proceed into the following demonstration scenarios.

Query Evaluation with Evolution Semantics. The main part of the demonstration will be the use of evolution in query answering. A number of queries will be executed and their results will be presented. The users will have the ability to see the results of their queries as they would have been returned by a traditional search engine. Then, by selecting the appropriate tab the results when evolution is taken into consideration will be displayed. The additional results will be highlighted and explained. For explanation, we use the Steiner forest that illustrates to the user in a graphical way the evolution merges that have led to the specific answer. Furthermore, the user will also be able to choose between seeing the top-k results of the fully collapsed evolution graph, and compare the time needed for generating each one.

Ranking Based on Different Metrics. Merging entities that are (from the evolution perspective) far to each other is not as important as merging those that are close. The users of the demo will have the ability to experiment with the different ranking functions which are based on the Steiner forest connecting the entities to be merged, and witness how the results are automatically affected.

The overall demo will allow SIGMOD participants to understand the concept of evolution in this new perspective, realize the different parameters that are affecting it, learn the algorithms that are producing answers under this assumption, and amuse themselves with the interesting results that can be found in the US patent dataset and the way that American companies have evolved over time.

References

[1] K. Berberich, S. J. Bedathur, M. Sozio, and G. Weikum. Bridging the Terminology Gap in Web Archive Search. In *WebDB*, 2009.
[2] S. Bergamaschi, F. Guerra, S. Rota, and Y. Velegrakis. A Hidden Markov Model Approach to Keyword-based Search over Relational Databases. In *ER*, 2011.
[3] S. Bykau, J. Mylopoulos, F. Rizzolo, and Y. Velegrakis. Supporting queries spanning across phases of evolving artifacts using steiner forests. In *CIKM*, pages 1649–1658, 2011.
[4] S. Bykau, J. Mylopoulos, F. Rizzolo, and Y. Velegrakis. On modeling and querying concept evolution. *J. Data Semantics*, pages 31–55, 2012.
[5] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB J.*, pages 523–544, 2007.
[6] M. Franklin, A. Halevy, and D. Maier. A first tutorial on dataspaces. *VLDB*, pages 1516–1517, 2008.
[7] F. Rizzolo, Y. Velegrakis, J. Mylopoulos, and S. Bykau. Modeling Concept Evolution: A Historical Perspective. In *ER*, pages 331–345, 2009.