Tiresias: A Demonstration of How-To Queries[*]

Alexandra Meliou Yisong Song Dan Suciu

Computer Science & Engineering
University of Washington
Seattle, WA, USA
{ameli,titanium,suciu}@cs.washington.edu

ABSTRACT

In this demo, we will present TIRESIAS, the first how-to query engine. How-to queries represent fundamental data analysis questions of the form: "How should the input change in order to achieve the desired output". They exemplify an important *Reverse Data Management* problem: solving constrained optimization problems over data residing in a DBMS. TIRESIAS, named after the mythical oracle of Thebes, has complex under-workings, but includes a simple interface that allows users to load datasets and interactively design optimization problems by simply selecting *actions*, *key performance indicators*, and *objectives*. The user choices are translated into a declarative query, which is then processed by TIRESIAS and translated into a Mixed Integer Program: we then use an MIP solver to find a solution. The solution is then presented to the user as an interactive data instance. The user can provide feedback by rejecting certain tuples and/or values. Then, based on the user feedback, TIRESIAS automatically *refines* the how-to query and presents a new set of results.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Constrained Optimization*; H.2.3 [**Database Management**]: Languages

General Terms

Languages, Algorithms

Keywords

Constrained Optimization, Tiresias, TiQL

1. INTRODUCTION

We will demonstrate TIRESIAS, the first system to support *how-to* queries. A *how-to* query tells the user what changes to make to a database in order to achieve a desired effect on one or several query indicators, while satisfying several global constraints. Such queries are important today in corporate planning. A Key Performance Indicator (KPI), is a standard industry term for a measure that evaluates a company's performance according to some metric [5]; for example, "percentage increase of customer base". A how-to query proposes hypothetical updates to the database in order to achieve certain effects on such indicators. How-to queries are the inverse of, and should not be confused with, what-if queries. In a *What-if* query [4, 3] a user describes some hypothetical changes in the database, and the system computes the effect on the indicators: for example, *what would the average number of suppliers per product be if we were willing to pay an extra 5% per each part?* A what-if query requires the decision maker to specify the hypothetical change, and the query will compute the effect on the indicator. *How-to* queries are the opposite: the decision maker specifies the desired effect on the indicators, and the system proposes some hypothetical updates to the database that achieve that effect.

EXAMPLE 1. *A manufacturing company that designs wireless chips maintains a list of required parts that they frequently order from various suppliers. Each order request specifies a part and a supplier that can provide the part.*

As a strategic decision, the company decides that they should diversify their list of suppliers as much as possible (i.e., order parts from as many different suppliers as possible). To achieve that, they can reassign an ordered item to a different supplier, as long as that supplier can supply the part. The question is, how can they determine the optimal arrangement?

TIRESIAS presents users with a simple interface (Fig. 3), where they can choose among three lists: a set of actions that are allowed on the EDB (existing DB instance) in order to achieve the how-to query, a set of KPIs together with constraints, and a set of possible objective functions to be optimized. The user chooses from these lists. For example, possible actions include:

- "orders may be split into multiple orders"
- "choose new suppliers for ordered items"
- "change the item quantity in orders"
- "cancel (delete) orders"

Possible KPIs include customizable options, for example:

- "Every order needs to have at least [...] items"
- "The total items in an order cannot exceed [...]"

[*]This work was partially funded by NSF IIS-0911036 and IIS-0915054

- "At least [...]% of all orders from each client should contain items from a local supplier"

Finally, some possible objective functions:

- "Inflict the minimum change in total quantities"
- "Minimize the number of ordered items that are assigned to a new supplier"
- "Minimize the number of canceled orders"

Once the user is happy with their selection they submit their query for execution. TIRESIAS generates a program in a declarative query language, called TiQL [6], designed for specifying how-to queries. In TiQL a programmer defines *hypothetical relations*, or hypothetical tables, by using non-deterministic transformations on the input database (EDB). Together, the hypothetical relations form the *hypothetical database*, or HDB. Thus, the HDB is derived from the current database using the allowed actions, satisfying the constraints on the indicators, and optimizing the objective function requirements.

TiQL does not compute the HDB directly. Instead, it generates a set of linear constraints that describe the HDB, and submits them to a Mixed Integer Programming solver to generate a solution. The numerical solution is then translated back into relations, and they form the HDB; the hypothetical tables are stored in the relational database.

At this point, a second interface Fig. 4 allows the users to interact with the HDB. Updated tuples are indicated correspondingly: they may be obtained through changes to the old tuples, or may have been freshly inserted. Users can select tuples, and also select values and indicate further constraints on those values. Then a user has several options. She can delete the selected tuples from the hypothetical database, she can accept the selected tuples (hence they will be propagated to the real database), or she may ask TIRESIAS to refine the hypothetical database by avoiding the selected tuples and satisfying the additional constraints.

TIRESIAS also allows user interaction through various data visualization options. The system automatically selects aggregates relevant to the optimization problem, and displays them as pie charts and/or bar graphs. Users can select data points in the visualizations (e.g., a piece of a pie chart) in order to update or create a new constraint.

Our system was named from the mythical oracle of Thebes, in ancient Greece. He was so wise, that the gods blinded him for accessing and revealing their secrets. The TIRESIAS system serves as an oracle on top of a database: it empowers users to ask for hypothetical, major changes to the database, in order to achieve a certain outcome for some key performance indicators, while satisfying global constraints.

2. TIRESIAS ARCHITECTURE

We show the architecture of TIRESIAS in Fig. 1. A detailed description of the system can be found in [6], but we also provide a high-level description here. Users can enter input directly as a TiQL query, or select various optimization options from the input screens. These options are actions (e.g. "change the supplier of ordered items"), key performance indicators (e.g. "each supplier should have at most 100 orders"), and requirements (e.g. "minimize the number of individual suppliers in all orders"). The system generates the corresponding TiQL program, which is then statically analyzed to construct helper relations (core tables), and partitioning information to be stored in the database.

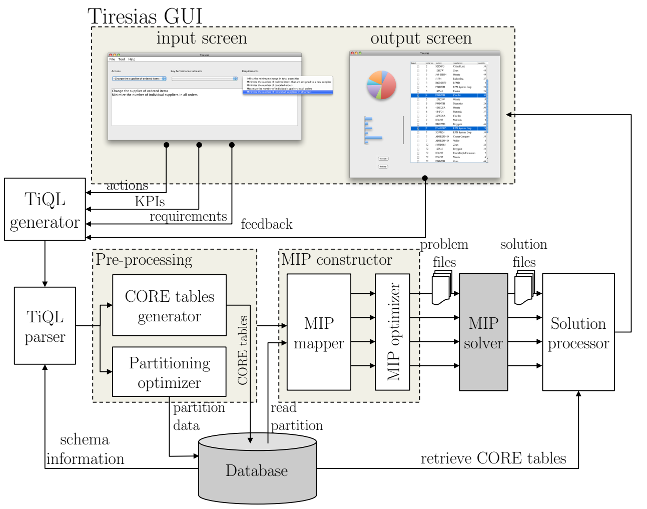


Figure 1: The Tiresias system architecture.

```
HTABLES:
  HChooseS(ok,pk,sk,sk')          KEY:(ok,pk,sk),(ok,pk,sk')
  HLineItem(ok,pk,sk,q)           KEY:(ok,pk,sk)
  HDistinctSup(sk)                KEY:(sk)
RULES:
  HChooseS(ok,pk,sk,sk')       :- PartSupp(pk,sk')
                                & LineItem(ok,pk,sk,qnt)
  HLineItem(ok,pk,sk',qnt)     :- HChooseS(ok,pk,sk,sk')
                                & LineItem(ok,pk,sk,qnt)
  HDistinctSup(sk')            :- HLineItem(ok,pk,sk',q)
MAXIMIZE(count(*))  :-  HDistinctSup(sk')
```

Figure 2: A TiQL program that implements Example 1 over a simplified TPC-H schema.

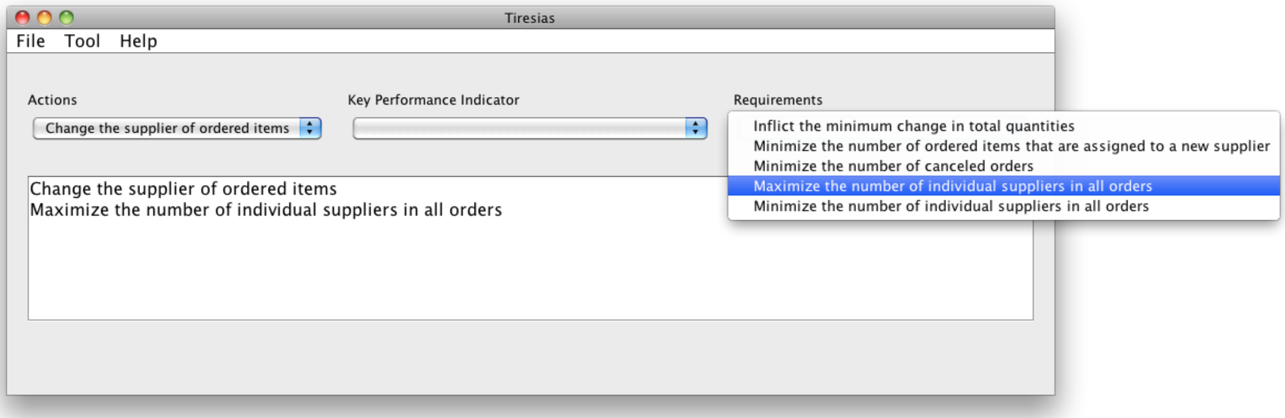
The MIP[1] constructor retrieves the partitioning information and queries the core tables to construct MIP problems. The MIP mapper follows sophisticated translation rules to transform the declarative input to linear constraints. After applying several size optimizations, the MIP problem is send to a black-box MIP solver. The current implementation of TIRESIAS interfaces with Postgres as the underlying DBMS, and the GLPK [2] solver as its black-box MIP solver.

The solution processor parses the solution and generates a new hypothetical instance which is send to the output screen for the user to evaluate. The user can accept the solution as is, or denote their disagreement with certain tuples, values, or aggregates. The collected feedback is used to refine the how-to query which is resubmitted to the system.

3. TiQL

The central piece in TIRESIAS is a novel declarative language for expressing how-to queries. End users do not have to write directly in TiQL: this is left to the database administrator. Instead, the TIRESIAS interface uses a program file consisting of several TiQL rules to present the list of actions, constraints, and objective functions.

[1] *Mixed Integer Programming*


Tiresias
File Tool Help
Actions
Change the supplier of ordered items
Key Performance Indicator
Requirements
Inflict the minimum change in total quantities
Minimize the number of ordered items that are assigned to a new supplier
Minimize the number of canceled orders
Maximize the number of individual suppliers in all orders
Minimize the number of individual suppliers in all orders

Change the supplier of ordered items
Maximize the number of individual suppliers in all orders

Figure 3: The Tiresias demo input screen. The users can design a how-to query by selecting from multiple actions, KPIs, and requirements. Once they are satisfied with their selection, they submit the query for evaluation.

Following datalog terminology, we call a database an EDB, for *extensional database*. In TIRESIAS, users write how-to queries in a declarative language called TiQL. The central concept in TiQL is that of a *hypothetical table*; the collection of all hypothetical tables forms the *Hypothetical Database*, or HDB. The TiQL program defines a set of HDB predicates, which are derived non-deterministically from the EDB, as well as a set of constraints that the hypothetical tables must satisfy. The HDB predicates are not uniquely defined. Instead, the user merely expresses what actions on the database she allows in order to satisfy the *how-to* query. In addition, there are several constraints in a TiQL program, which encode business constraints, or desired outcomes for the KPI's that the user is trying to improve.

Figure 2 shows the representation of Example 1 in TiQL. The main HDB predicate is `HLineItem`; the other two HDB predicates (`HChooseS` and `HDistinctSup`) are auxiliary, and are used for clarity.

The semantics of a TiQL program is the collection of all possible worlds on the hypothetical relations that satisfy the constraints and achieve the desired changes to the KPI's. The main part of the system is a translator from TiQL into linear (LP) or *Mixed Integer Program*[2] (MIP). The tuples and/or the attributes of the hypothetical relation become integer or real variables in a LP/MIP, and the datalog rules in TiQL become linear constraints. Most importantly, the system performs several optimizations on the TiQL and the resulting MIP program before submitting it to the solver: these optimizations are necessary in order to scale to large databases, otherwise today's solvers would not be able to handle the resulting MIP problems. Finally, the results from the solver are translated back into relational format and presented to the user. In addition to its own compiler and optimizer, TIRESIAS uses a relational database system for accessing the data (we used Postgres), and an LP/MIP solver for solving the optimization problem (we used the GLPK solver [2]).

TIRESIAS makes use of declarative modeling languages to communicate with the MIP solver. Our current implementation uses MathProg, which is a subset of AMPL [1]. AMPL adds a level of declarativeness to the definition of constrained optimization problems through abstractions that can generalize the definition of variables and constraints. For example, a long list of constraints "$x_1 < 5$, $x_2 < 5$, $\ldots x_n < 5$" is abstracted to "`{i in S}:x[i]<5, S :=[1,2,..., n]`". Languages like AMPL only offer weak DBMS interfaces, and they don't provide the same level of declarativeness as TiQL, since variables and constraints still need to be explicitly defined.

[2]A MIP is a LP where *some* of the variables are restricted to integer values.

4. DEMONSTRATION OUTLINE

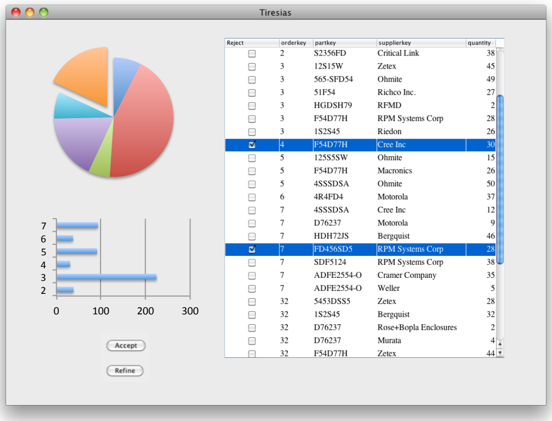
We will demonstrate the construction and execution of how-to queries in TIRESIAS, by guiding users through the manufacturing company scenario. Our goal is to show that TIRESIAS provides functionality that so far has not been present in database systems, and that how-to queries facilitate the definition of optimization problems over large datasets. Moreover, we will demonstrate the scalability of our system through examples over large datasets.

4.1 Optimization Problems in TIRESIAS

We will demonstrate the following use cases:

Defining a basic How-to Query. Our first demonstration will impersonate a manager of the manufacturing company of Example 1, who uses the TIRESIAS graphical user interface (Fig. 3) to submit a basic how-to query. As specified in Example 1, the manager would like to diversify the list of suppliers in the company's order table. The user will load the corresponding dataset from the File menu, and then will have to select from various possible options in order to build the how-to query. She first needs to determine the action to take: that is to change the supplier for some ordered items. Then she will select from a drop-down list the requirement that the list of individual suppliers in the orders is maximized. When the user has all the appropriate selections, she will click "Submit query" in the File menu.

TIRESIAS will display within a few seconds the hypothetical table result in the output screen (Fig. 4). The user can



Reject	orderkey	partkey	supplierkey	quantity
☐	2	S2356FD	Critical Link	38
☐	3	12S15W	Zetex	45
☐	3	565-SFD54	Ohmite	49
☐	3	51F54	Richco Inc.	27
☐	3	HGDSH79	RFMD	2
☐	3	F54D77H	RPM Systems Corp	28
☐	3	1S2S45	Riedon	26
✓	4	F54D77H	Cree Inc	30
☐	5	125S5SW	Ohmite	15
☐	5	F54D77H	Macronics	26
☐	5	4SSSDSA	Ohmite	50
☐	6	4R4FD4	Motorola	37
☐	7	4SSSDSA	Cree Inc	12
☐	7	D76237	Motorola	9
☐	7	HDH72JS	Bergquist	46
✓	7	FD456SD5	RPM Systems Corp	28
☐	7	SDF5124	RPM Systems Corp	38
☐	7	ADFE2554-O	Cramer Company	35
☐	7	ADFE2554-O	Weller	5
☐	32	5453DSS5	Zetex	28
☐	32	1S2S45	Bergquist	32
☐	32	D76237	Rose+Bopla Enclosures	2
☐	32	D76237	Murata	4
☐	32	F54D77H	Zetex	44

Figure 4: The Tiresias output screen allows the user to interact with the hypothetical database, accept the answers, or mark tuples or aggregate values as problematic and resubmit a *refinement* to the previous query.

scroll through the data instance, order based on a column, and interact with different visualizations.

Compare problems with different specifications. In continuation of the previous scenario, the user notices in the results that the order table has changed much more than she expected. She realizes that in selecting the most diverse supplier arrangement, she should give preference when possible to the supplier originally assigned to each ordered item.

From the input screen, she adds the additional requirement that the number of changes in the dataset is minimized. Execution of this query will bring up a new output screen, where the user can draw comparisons with the earlier results.

Solution Feedback. The user is satisfied with the hypothetical instance produced by the second query, but there are still a few particular values that she would like to correct: for certain important parts she would like to enforce that no substitutions be made. However, we can't simply revert those particular tuples to their original values and keep the rest, as the solution may not be optimal anymore. However, instead of restarting the query, the user can select the tuples to be rejected, and click "Refine". Alternatively, the user may provide feedback through the graph visualizations over data aggregates. She can select a portion of a pie chart, and insert an additional constraint for the corresponding aggregate, or edit an existing one. TIRESIAS processes the user feedback and automatically generates the additional needed constraints before running the optimization again.

Throughout the demo scenario, we will showcase the TiQL program generated from the user selections in the GUI, as well as the mixed integer program generated by the MIP constructor. This will show that TIRESIAS provides a powerful abstraction for very complex optimization problems.

5. DISCUSSION

TIRESIAS is the first system that integrates databases with constrained optimization solvers, which has very important benefits. The data does not need to be extracted from the database, where it naturally resides. In fact, users can now *declaratively* specify optimization problems as powerful how-to queries, which are *significantly* more compact than the MIP problems they represent. An additional pivotal benefit is that TIRESIAS can go a step ahead of what modern solver tools can accomplish: TIRESIAS uses data-specific information (e.g. functional dependencies), and analyzes the declarative TiQL statements to greatly optimize the generated optimization problem, by eliminating unnecessary constraints and variables, and even by splitting the problem into several independent subproblems. This is a crucial step, as even state of the art solvers have limitations on the problem sizes that they can handle.

Our current implementation can in fact handle complex optimization problems over more than 1M tuples. As part of the demonstration we will also present these performance statistics over a variety of queries and data sizes.

6. REFERENCES

[1] AMPL: A modeling language for mathematical programming. http://www.ampl.com.

[2] GNU Linear Programing Kit. http://www.gnu.org/s/glpk.

[3] L. Antova, C. Koch, and D. Olteanu. From complete to incomplete information and back. In *SIGMOD Conference*, pages 713–724, 2007.

[4] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas. Mcdb: a monte carlo approach to managing uncertain data. In *SIGMOD Conference*, pages 687–700, 2008.

[5] K. Lyytinen, P. Loucopoulos, and J. Mylopoulos. *Design Requirements Engineering: A Ten-Year Perspective: Design Requirements Workshop, Cleveland, OH, USA, June 3-6, 2007, Revised and Invited Papers*. Lecture Notes in Business Information Processing. Springer, 2009.

[6] A. Meliou and D. Suciu. Tiresias: The database oracle for how-to queries. In *SIGMOD Conference*, 2012.