IQR: An Interactive Query Relaxation System for the Empty-Answer Problem

Davide Mottin
University of Trento
mottin@disi.unitn.eu

Alice Marascu
IBM Research-Ireland
alice.marascu@ie.ibm.com

Senjuti Basu Roy
U of Washington Tacoma
senjutib@uw.edu

Gautam Das
UT Arlington & QCRI
gdas@uta.edu - gdas@qf.org.qa

Themis Palpanas
Paris Descartes University
themis@mi.parisdescartes.fr

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

ABSTRACT

We present IQR, a system that demonstrates optimization based interactive relaxations for queries that return an empty answer. Given an empty answer, IQR dynamically suggests one relaxation of the original query conditions at a time to the user, based on certain optimization objectives, and the user responds by either accepting or declining the relaxation, until the user arrives at a non-empty answer, or a non-empty answer is impossible to achieve with any further relaxations. The relaxation suggestions hinge on a probabilistic framework that takes into account the probability of the user accepting a suggested relaxation, as well as how much that relaxation serves towards the optimization objective. IQR accepts a wide variety of optimization objectives - user centric objectives, such as, minimizing the number of user interactions (i.e., effort) or returning relevant results, as well as seller centric objectives, such as, maximizing profit. IQR offers principled *exact* and *approximate* solutions for generating relaxations that are demonstrated using multiple, large real datasets.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Search process*

Keywords

Empty-answer problem; Query Relaxation; Optimization framework

1. INTRODUCTION

Numerous web applications allow users to search for items of interest like homes, cars, apparel, etc., by specifying their desired attribute values that are later on turned into conjunctive query predicates and the query gets executed over the underlying data source. *Information under-load* or the *empty-answer* is a frequently encountered problem for such queries, where the user over-specifies the query conditions and consequently the system finds no item in the source satisfying them all. To circumvent the *empty-answer problem*, *query relaxation* has been proposed as an effective strategy that reformulates the original query into a new query by removing or relaxing conditions and is considered as a novel alternative to the top-k ranked retrieval model [5]. Existing query relaxation strategies are mostly non-interactive, i.e., they suggest multiple relaxations to the user at a time, and are not designed to offer relaxations that guarantee the optimization of certain objectives in the returned results.

In this demonstration paper, we showcase IQR[1]- an interactive query relaxation system that not only proposes step-by-step relaxation suggestions to the users, but the suggestions are designed to optimize certain objective functions in the returned non-empty results.[2] As such, IQR is meaningful for scenarios in which the user interacts with the data source via a small device (e.g., mobile phone), or for applications where customer-agent interactions unfold step-by-step over the phone (e.g., purchase of travel insurance, reservation of a holiday house, etc.).

IQR accepts a wide range of optimization objectives: *user-centric* objectives, such as, suggesting relaxations that will return the user *most relevant* products, or the *cheapest* products, or *minimize user's navigational effort* by minimizing the number of relaxation steps, as well as, *seller-centric* objectives, such as, suggesting relaxations that will return *the most expensive* products. In order to decide what should be the next proposed relaxation, IQR first computes the *likelihood* that the user will respond positively to a proposal, as well as quantify the effectiveness of the proposal with respect to the optimization objective. Since IQR does not know the exact user intent before she makes the choice, it resorts to a *probabilistic framework* for reasoning about this question.

The IQR Approach: Given a user query that returns an empty-answer, IQR suggests one relaxation at a time. The user responds to it with a "yes"/"no". Based on the response, the next relaxation is suggested and this iterative process continues until one of the two conditions are satisfied: (a) the user has arrived at a non-empty result set; (b) non-empty result set is impossible to achieve with any further relaxations. We describe a running example next.

Running Example: Imagine that a user is interested in

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.
ACM 978-1-4503-2376-5/14/06
http://dx.doi.org/10.1145/2588555.2594512.

[1] http://youtu.be/v9CXnQyZ9jw
[2] Full description of the IQR theory can be found in [6].

	Make	Model	Price	ABS	MP3	Alarm	4WD	DSL	Manual	HiFi	ESP	Turbo
t_1	VW	Touareg	$62K	1	0	0	0	0	1	0	1	0
t_2	Askari	A10	$206K	0	1	0	0	1	1	1	1	0
t_3	Honda	Civic	$32K	1	0	0	0	0	0	0	0	0
t_4	Porsche	911	$126K	0	0	0	0	1	0	1	1	0

Figure 1: An instance of a car database.

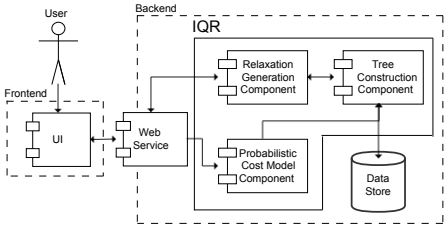


Figure 2: Architecture of the system IQR

a car that has anti-lock braking system (ABS), dusk-sensing light (DSL), and manual transmission. The data instance of Figure 1 reveals that no car satisfies these requirements.

User-centric Optimization: Consider the optimization objective is to return the *most relevant result*. Imagine that the most relevant tuples to the user query are the VW and the Askari (both satisfy two out of the three query constraints). However, if IQR knows that the most users prefer cars with ABS (i.e., the user has a very high likelihood of responding "no" to the suggestion), then it would instead suggest her to relax DSL to return the VW to her.

Seller-centric Optimization: Conversely, if the seller wants to sell the *most expensive* car (i.e., Askari) to the user, IQR would instead propose cars with no ABS.

IQR is designed to take all of these issues into account while proposing relaxation suggestions based on a certain optimization objective. The effectiveness of a relaxation suggestion is quantified by its *expected cost* by capturing *how well* it serves towards the optimization objective and *how likely* the user will accept it. Given the initial query, the underlying solution then relies on constructing a query relaxation tree [6] that captures the space of possible relaxations as well as user responses and computing the *expected cost* (probabilistic) for each of them. After that, this tree needs to be traversed in a top down fashion to suggest a relaxation that optimizes the expected cost.

A big challenge is raised by the fact that the relaxation suggestions must be generated in real-time, given that the size of the query relaxation tree is exponential to the number of query constraints. IQR offers novel algorithms to ensure scalability by proposing efficient exact solutions, as well as principled approximate solutions.

2. THE IQR SYSTEM

IQR is a web-application (the architecture and flow of information of the system is shown in Figure 2), which can be invoked from the computers or from the mobile devices. The *Query Builder* tab allows the user to select a data source and build a conjunctive query from a drop-down list. The *Search* tab enables searching. If the initial query returns an empty-answer, then the interface allows the user to perform interactive step-by-step relaxation by appropriately selecting the optimization objective. The user can also select the relaxation strategy (i.e., exact or approximate) from the interface. Finally, the *Results* tab shows the returned results along with additional statistics. The back-end consists of the following three components: (a) *Relaxation tree construction*, (b) *Probabilistic cost model*, and (c) *Relaxations generation*. The tree construction component interacts with the database. To enable faster database retrieval, IQR first converts the database to a Boolean type and then designs bitmap indexes for fast retrieval. The probabilistic cost model is abstracted as a framework which could be easily instantiated for different objective functions, or different implementations for computing probabilities. We describe these components next.

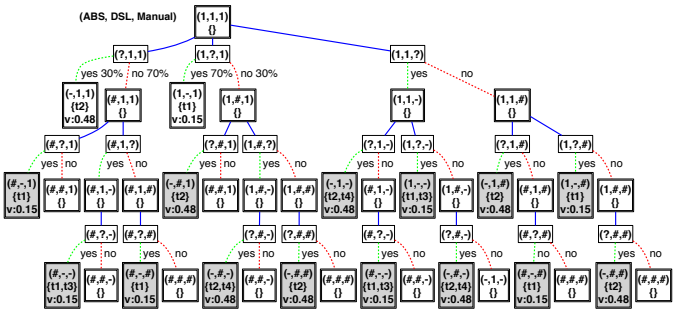


Figure 3: Query Relaxation tree of the running example.

2.1 Relaxation Tree Construction

To encode the different relaxation suggestions and user choices that may occur for a given empty-query Q, we employ a special tree structure called the *query relaxation tree* (Figure 3 represents the running example query). The tree contains two types of nodes: the *relaxation* nodes (marked with double-line rectangles in Figure 3) and the *choice* nodes (marked with single-line rectangles in Figure 3). Note that the children of relaxation nodes are choice nodes, and the children of choice nodes are relaxation nodes.

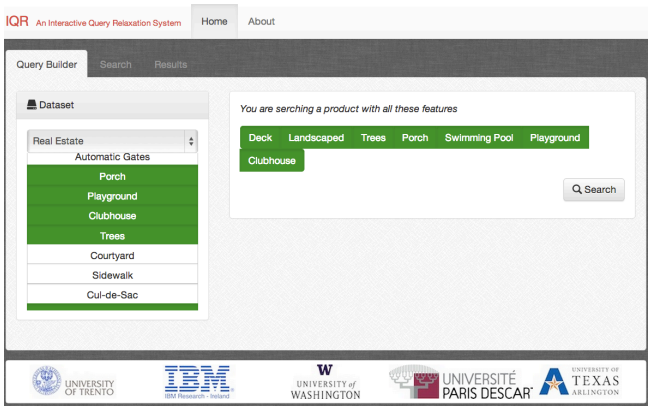
A *relaxation node* represents a relaxed query Q'. The root node represents the original query Q. A relaxation node does not have any children when the respective query returns a non-empty answer, or returns an empty-answer but cannot be relaxed further.

Conversely, a *choice node* models an interaction with the user and always has two children: one that corresponds to a positive ("yes") response, and one that corresponds to a negative ("no") response. A choice node can never be a leaf. Thus, any root-to-leaf path in the tree starts with a relaxation node, ends with a relaxation node, and consists of an alternating sequence of relaxation and choice nodes.

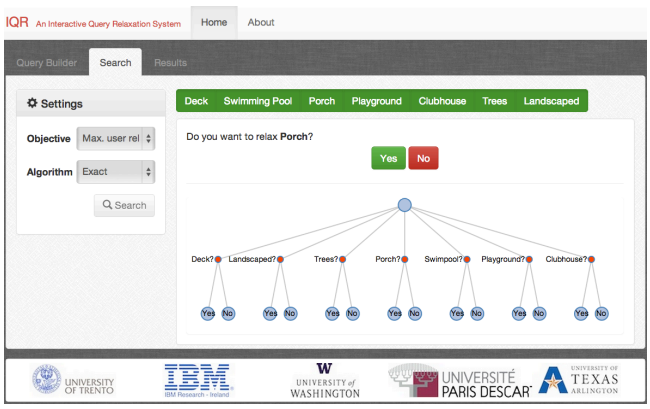
$\mathfrak{Constrs}(Q)$ denotes the set of query constraints some of which may be fundamental; these are referred to as *hard constraints* and all the others as *soft constraints*. We use the "#" symbol to indicate a hard constraint in a query, and "?" to indicate a question to the user for the relaxation of the respective constraint. With every "no" response associated with a suggested relaxation, it becomes a *hard constraint*.

2.2 Probabilistic Cost Model

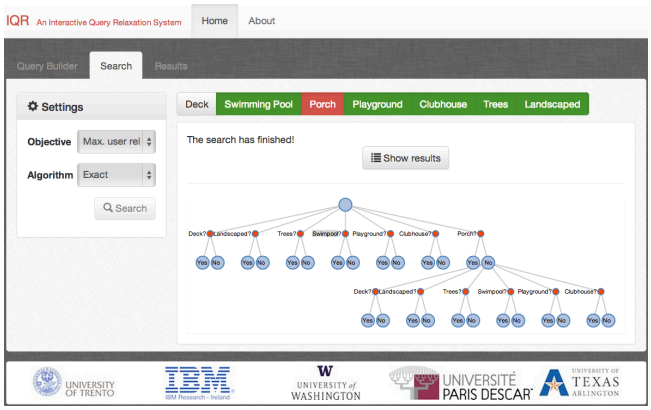
Once the tree is constructed, IQR needs to probabilistically compute the "cost" associated with each possible relaxation suggestion. Intuitively, the probability that a user responds positively to a proposed relaxation depends on the (a) *prior* function, $prior(Q, Q', t)$ - which measures the user belief that a certain tuple t satisfying the relaxed query Q' exists in the database, and the (b) *preference* function, $pref(t, Q)$



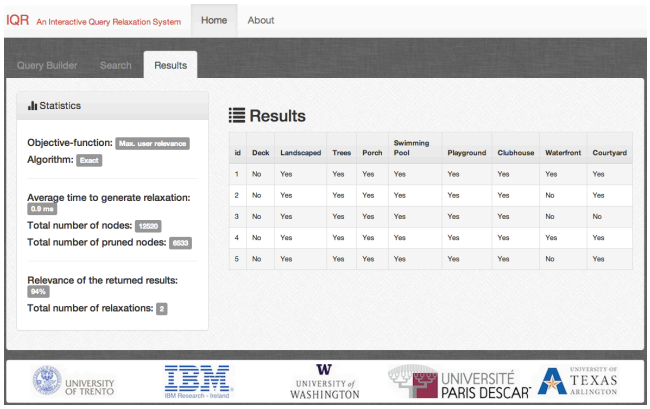
(a) Query building and search.



(b) Exact algorithm for max. relevance suggests "Porch" first.



(c) Interaction ends with "no" to "Porch","yes" to "Deck".



(d) Final page enlists results and additional statistics.

Figure 4: The interactive query relaxation system IQR.

- which estimates the probability that a user will like a tuple t (even though it only partially satisfy the initial query).

Even though any prior function or preference function (i.e, ranking/scoring) [3, 1] could be adapted in IQR, in our implementation, we use the Iterative Proportional Fitting (IPF) [4] technique on the instance data to compute the prior and the Normalized Inverse Document Frequency [1] to calculate the preference.[6] contains further details.

Therefore, the probability to reject a relaxation Q' is:

$$relPref_{no}(Q, Q') = \sum_{t \in Q'} (1 - pref(t, Q')) * prior(t, Q, Q')$$

which represents the probability of not liking any of the tuples associated to the relaxation. Thus, the probability of accepting the relaxation is the probability that the user likes at least one tuple, namely,

$$relPref_{yes}(Q, Q') = 1 - relPref_{no}(Q, Q')$$

A *probabilistic cost* value is associated to every node of the relaxation tree that quantifies how suitable a certain relaxation is towards a specific optimization objective.

The *expected cost* of a choice node n can be expressed as:

$$Cost(n) = relPref_{yes}(Q, Q') * (C_1 + Cost(n_{yes})) + relPref_{no}(Q, Q') * (C_1 + Cost(n_{no})) \quad (1)$$

where the n_{yes} and n_{no} are the two children (relaxation) nodes of n, Q is the query corresponding to the parent of n, and Q' corresponds to the suggested relaxation of Q at node n and the variable C_1 is a constant, that is used to quantify any additional cost incurred for answering the current relaxation proposal.

Since the cost of a node depends on the below subtree, to produce the optimal solution, we select that relaxation node which optimizes (maximizes or minimizes) the cost, such as:

$$Cost(n) = optimize_{c \in S} Cost(n_c) \quad (2)$$

where S is the set of soft constraints in $\mathfrak{Constrs}(Q)$, and n_c is the choice child node of n that corresponds to an attempt to relax the soft constraint c. These equations capture a generic cost and have to be appropriately instantiated for a specific application as described in Section 2.2.1.

Finally, the *cost* of a leaf node depends on the "value" of the tuples in that leaf which quantifies the effectiveness of those tuples towards the specific objective, which is potentially different from the value of the *preference* function.

2.2.1 *Application Specific Cost Computation*

Next we describe how to instantiate the above cost model under different application scenarios by appropriately modifying the *preference, value, and the cost* computation.

The preference for a tuple (1) could be independent of the query Q and static; or, (2) it could be query dependent, but only depends on the initial query and does not change after that; or, (3) it could depend on the latest relaxed query the user has accepted - this is a very dynamic scenario where after each step of the interactive session the preference can change. These different preference computation approaches are referred to as Static, Semi-Dynamic, and Dynamic respectively. An astute reader may notice that Static is applicable to optimization objectives that are agnostic to query constraints (such as minimize cost, maximize profit), Semi-

Dynamic is for applications that require the preference to be proportional to the original query Q (e.g., maximizing relevance), and Dynamic is designed for applications where the preference changes with every accepted relaxation (e.g., minimizing user effort). The *value* calculation of a tuple t is similar, except that it needs to quantify the contribution of t towards a specific objective.

Similarly, when the objective is *maximize* (*minimize*), the cost of a relaxation node is the *maximum* (*minimum*) cost of its children. The cost of a choice node is calculated using Equation 1 by setting $C_1 = 0$ to all optimization objectives except effort minimization ($C_1 = 1$ in this case), since the latter case incurs a cost of 1 with every additional relaxation suggestions. We refer to [6] for further details.

2.3 Relaxations Generation

IQR proposes an *exact algorithm* and an *approximate algorithm* to efficiently generate the relaxation suggestions.

2.3.1 Exact Query Relaxation Algorithm

To avoid computing the entire query relaxation tree, IQR constructs the tree partially only up to a certain level $L < |\mathfrak{Constrs}(Q)|$, and try to assign the cost of all the nodes in level L and higher by calculating the *upper and lower bound* of cost. From the ranges of the costs that the computation provides, it identifies those branches that cannot lead to optimal cost. For example, when the specific optimization minimizes (maximizes) cost, these are the branches starting with a node that has as a *lower bound (upper bound) for its cost that is higher (lower) than the upper bound (lower bound) of the cost of another sibling node*. By pruning these branches the required computations are significantly reduced. This way, the computation proceeds level by level until all query $|\mathfrak{Constrs}(Q)|$ constraints are relaxed.

The challenge here is to *tightly* estimate the upper bound and lower bound of cost of the tree nodes for different optimization objectives without compromising the correctness. As discussed in [6], IQR indeed offers this efficient alternative under a variety of optimization objectives.

2.3.2 Approximate Query Relaxation Algorithm

Although the strategy discussed in Section 2.3.1 generates optimum-cost relaxations and builds the relaxation tree on demand, it may still have to construct the entire tree first in the worst case, even before suggesting any relaxation to the user. Applications that demand fast response time but allow slight imprecision (such as, online air-ticket or rental-car reservation systems) may not be able to tolerate such latency. IQR proposes a *novel approximate solution* that takes L (# levels) as an input parameter and constructs the tree only upto level L leading to a *significantly smaller* tree.

This strategy computes first the exact structure of the relaxation tree up to level L. Next, it approximates the cost of each L-th level choice nodes by considering the cost distributions of its children and proceeds with a bottom-up computation of the remaining nodes until the root. At the root node, the best relaxation child node is selected, and the remaining ones are pruned. Upon suggesting this new relaxation, the algorithm continues further based on the user response. Essentially, the task is to probabilistically compute the *cost distribution* (or pdf of cost in short) of the relaxation and choice nodes at level L and higher by computing the *Sum, Min, or Max convolutions* [2] of the respective pdfs. Efficiency is guaranteed by approximating the convolutions using histograms. We omit the details for brevity and refer to the main IQR article [6].

3. SYSTEM DEMONSTRATION

In this demonstration, the users will experience IQR's domain independent approach for efficient interactive relaxation suggestions under different optimization objectives using both exact and approximate solutions. The users will be able to select the US Homes or the Yahoo Autos as the underlying data source. The former database has 18 attributes and over 250,000 tuples, while the latter has 31 attributes and about 500,000 tuples. The demo will first show the scalability of the system with empty-queries of increasing size. The users will then select the dataset and build their own query (Figure 4a), with the guidance of the system. If it results in an empty-answer, then they will select a specific optimization objective and an algorithm type (exact or approximate), as shown in Figure 4b. After that the users will be suggested interactive relaxations (Figure 4b, 4c) at end of which the results and additional statistics will be returned (Figure 4d). Based on users' interactions, the statistics page presents interesting information, such as, the effectiveness of the returned results towards the optimization objective, the average relaxation time, the total number of suggested relaxations, the total number of pruned nodes in the relaxation tree by the underlying algorithm, etc. Additional statistical information will be provided, such as the typical response times and the mean number of interactions computed during the demo session. In particular, the users will experience IQR under three different optimization objectives.

1. **Relevance Maximization** - This is an instantiation of Semi-Dynamic preference computation as shown in Figure 4b. The process ends when the users arrive at a non-empty results (Figure 4d), or no further relaxation could generate non-empty answer. The final results maximize relevance and the exact solution may return different relaxations or tuples than the approximate one to the users, while the latter will appear more interactive. The users will be presented interesting statistics about her interactive relaxation session at the end of the relaxation process (Figure 4d).
2. **Profit Maximization** - This is an instantiation of Static preference computation and the attendees will experience that IQR will attempt to return the most expensive homes or cars to them based on their respective interactions.
3. **Effort Minimization** - This is an instantiation of Dynamic preference computation and the users will experience that IQR will attempt to return non-empty results by suggesting minimum number of relaxations .

References

[1] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *CIDR*, 2003.

[2] B. Arai, G. Das, D. Gunopulos, and N. Koudas. Anytime measures for top-k algorithms on exact and fuzzy data sets. *VLDB J.*, 18(2):407–427, 2009.

[3] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley New York, 2011.

[4] Y. Bishop, S. Fienberg, and P. Holland. *Discr. Multivariate Analysis: Theory and Practice*. MIT Press, 1975.

[5] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *TODS*, 2006.

[6] D. Mottin, A. Marascu, S. B. Roy, G. Das, T. Palpanas, and Y. Velegrakis. A probabilistic optimization framework for the empty-answer problem. *PVLDB*, 6(14):1762–1773, 2013.