2018 IEEE Latin American Conference on Computational Intelligence (LACCI)
Guadalajara, Jalisco, Mexico, November 7-9, 2018

Discovering SQL Queries from Examples using Intelligent Algorithms

Denis Mayr Lima Martins*, Gottfried Vossen* and Fernando Buarque de Lima Neto†
*University of Münster, ERCIS
Leonardo-Campus 3, 48149 Münster, Germany
Email: {denis.martins, vossen}@wi.uni-muenster.de
†University of Pernambuco, Computer Engineering Program
Rua Benfica 455, 50720-001, Recife, Brazil
Email: fbln@ecomp.poli.br

Abstract—**Formulating database queries in terms of SQL is often a challenge for journalists, business administrators, and the growing number of non-database experts that are required to access and explore data. To alleviate this problem, we proposed a Query By Example (QBE) approach powered by intelligent algorithms that discovers database queries from a few tuple examples provided by the user. We investigated the effectiveness of three algorithms, namely, Greedy Search, Genetic Programming, and CART decision trees in discovering queries in two distinct databases. To the best of our knowledge, no other research has focused on the comparative analysis of such algorithms in the context of QBE. Our results show that CART decision trees were capable of discovering the most accurate queries. However, CART tends to produce long queries, which may hinder user interpretation. Finally, we suggest that the use of Interactive Evolutionary Computational Intelligence may improve the quality of queries discovered by Genetic Programming and may naturally incorporate diverse user preferences in the discovery process.**

1. Introduction

With the increasing amount of data being generated every day, interacting with databases has become a vital activity in many scenarios such as data-driven decision-making, healthcare, scientific research, social media, and many others. In the context of relational databases, SQL queries are considered the most common *medium* for end-users to access, analyze, and manipulate data. However, the process of constructing accurate SQL queries raises two particular challenges. First, users are required to materialize their mental queries by using the SQL syntax, which is considered to have a steep learning curve. This process is commonly highly incremental and users frequently spend a considerable amount of time correcting and refining query statements and predicates rather than executing queries and evaluating answers [1]. And second, users are required to be familiar with the database structure and content [2], which also is time-consuming, particularly when users are non-database experts. To alleviate these problems, we investigate in this paper how intelligent algorithms can be applied to automatically construct SQL queries by asking users to provide data examples that correspond to or even satisfy their mental query.

Formulating accurate SQL queries is often challenging, particularly for the increasing number of non-database specialists who work in data intensive workplaces (e.g., journalists, astrophysicists, biologists) but typically lack technical knowledge. For these users, the process of query formulation usually consists of a series of trial-and-error attempts that may lead to the problem of user fatigue, when users desist to continue with the query construction [1].

With the objective to enable non-database experts to effectively query databases, the Query By Example (QBE) paradigm [3] has been employed in several research proposals. In QBE, database queries are automatically discovered from a set of tuple examples provided by the user, as we illustrate in the following example.

Consider that a Web page containing information about Brazilian soccer players displays tuples T2, T3, and T4, retrieved from the data table depicted in Table 1. Assume that a student wants to present the same information in a different Web page but has no access to the query used in the Web application. In this situation, a QBE system would allow the student to use T2, T3, and T4 as input examples to an intelligent algorithm that discovers a query such as `SELECT Name, Scored goals FROM Players WHERE Matches` $\leq$ `900` which appropriately retrieved the target information.

TABLE 1. SOCCER PLAYERS DATASET.

ID	Player	Scored goals	Matches
T1	Pelé	1159	1265
T2	Romário	1002	850
T3	Ronaldo	420	624
T4	Zico	406	596

QBE is especially useful in cases where an unknown query has to be discovered in order to reproduce a particular data view that was manually created or generated by a legacy system that is difficult or costly to maintain. In such situations, queries may be needed as part of a scientific document to allow reproducibility of results or to reduce maintenance costs (e.g., incorporating the discovered query in a less expensive system that will be used to replace the legacy one). Moreover, QBE can also assist non-database-expert users by producing alternative queries to potentially uncover insights about the data structure and schema or to reveal unexpected, hidden relationships within the data [4]. Discovering alternative queries is particularly of interest in data analytics and exploration scenarios.

In this paper, we investigate how intelligent algorithms, namely, greedy search, genetic programming, and CART decision trees can be applied in discovering database queries in a QBE setting. The manuscript is organized as follows. Section 2 introduces the appropriate problem definition and describes relevant research conducted on the topic of QBE. Section 3 introduces both the employed algorithms and the performance metrics used to evaluate queries discovered by the intelligent algorithms. Section 4 reports the conducted experiments and obtained results of the two selected problem scenarios, i.e., discovering unknown queries and searching for alternative queries. Section 5 presents a short discussion about the obtained results. Finally, Section 6 includes a brief conclusion and points out future research venues.

2. Problem definition and Related Work

The original proposal of QBE, presented in the 1970s by Zloof [3], employed a form-based user interface to construct tuple examples describing a possible answer to a user's mental query. However, the substantial research conducted on this topic in recent years has been treated the problem of discovering database queries from data examples in two distinct modes. For a comprehensive list of QBE-related research conducted on both relational and non-relational databases, we refer the reader to Mottin et al. [5].

The work of Zhang and Yuyin [6] and of Tran et al. [4] has considered QBE as a classification problem. In their designs, different Decision Tree algorithms were applied to generate classifiers that appropriately classify tuples in the database as positive or negative, whenever they match, respectively, the examples provided by the user or not. In this sense, the classifier itself can be seen as a predicate that appropriately describes patterns in the data that matches the tuples targeted by the user.

On the other hand, Bonifati et al. [7] and Martins et al. [8] characterized QBE as a search problem in which a large space of candidate queries is explored. They employed heuristic search algorithms (i.e., greedy search and genetic programming, respectively) to find a query that appropriately retrieves the described in the data examples. In this setting, the search space is composed of all candidate queries that are possibly constructed from the given database.

In either case, the problem of discovering database queries from tuple examples, as described in the example above, can be formulated as follows. Given a dataset `D`, find a query `Q` that retrieves all the tuple examples `E` provided by the user such that $E \subseteq Q(D)$. Notice that this definition allows intelligent algorithms to discover queries that can retrieve not only the entire set of example tuples but also additional tuples that may be of interest to the user. For instance, additional tuples may provide useful insights about unforeseen patterns in the data. According to Cumin et al. [9], allowing such diversity within query results is particularly beneficial in data exploration and knowledge discovery scenarios.

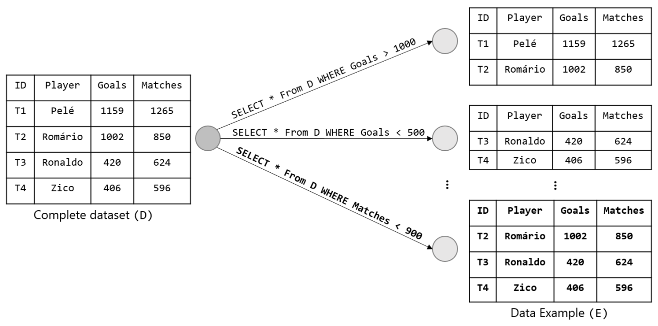
In this work, we consider a simplified version of this problem following the approach introduced in our previous work [8] in which only queries in the form of `SELECT * FROM database WHERE TSD` are considered. TSD is defined as a *tuple set descriptor* (TSD) rule, i.e., a set of predicates that appropriately captures the hidden patterns within `E` and returns all data tuples in `E`. Each predicate in TSD has the form of `Column operator Value`, where `Column` is a column in `D`, `operator` $\in \{=, <>, >, \geq, <, \leq\}$, and Value can assume any value present in the columns of `D`. Additionally, predicates in TSD are combined by the Boolean operators `AND` and `OR`.

3. Intelligent Algorithms for Discovering SQL Queries

Following the research literature, our goals are to analyze the effectiveness of three intelligent algorithms, namely, Greedy Search, Genetic Programming, CART decision trees in the context of QBE. Next, we provide a brief description of these algorithms.

3.1. Greedy Search

A Greedy Search (GS) algorithm employs the heuristic strategy of choosing the most promising option at each search stage. In the case of QBE, a GS starts with an empty TSD rule and at each iteration of the algorithm, all single predicates available are evaluated using a predefined evaluation function. The best-evaluated predicate is then chosen to compose the TSD. In this sense, QBE can be seen as a graph search in which queries are considered edges while nodes are the state of database `D` after executing the candidate query, i.e., query results. Figure 1 shows this graph-based representation referring to the illustrative example described in Section 1.


SELECT * From D WHERE Goals > 1000
SELECT * From D WHERE Goals < 500
SELECT * From D WHERE Matches < 900
⋮
⋮
Complete dataset (D)
Data Example (E)

ID	Player	Goals	Matches
T1	Pelé	1159	1265
T2	Romário	1002	850
T3	Ronaldo	420	624
T4	Zico	406	596

ID	Player	Goals	Matches
T1	Pelé	1159	1265
T2	Romário	1002	850

ID	Player	Goals	Matches
T3	Ronaldo	420	624
T4	Zico	406	596

ID	**Player**	**Goals**	**Matches**
T2	**Romário**	**1002**	**850**
T3	**Ronaldo**	**420**	**624**
T4	**Zico**	**406**	**596**

Figure 1. Query by Example as a graph search.

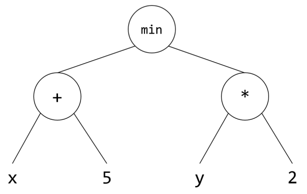


Figure 2. Tree representation of a genetic individual.

3.2. Genetic Programming

Genetic Programming (GP) is an evolutionary algorithm, initially proposed by Koza [10] in the early 1990s, for evolving computer programs in a process inspired by the Darwinian concept of the survival of the fittest. GP performs a domain-independent, stochastic process of transforming an initial population of computer programs into a better one composed of novel programs that are expectantly more suitable to solve the specific problem under consideration.

GP individuals are commonly represented similar to syntax trees that describe a particular program logic. In this tree-based representation, leafs portray variables and constants (i.e., the terminal set) while nodes represent functions and operations (i.e., the function set). Figure 2 illustrates the syntax tree representation of a program that computes $min(x + 5, y * 2)$.

The traditional GP algorithm starts with a random-generated population of n individuals based on both terminal and function sets. An evaluation function is employed to assess the appropriateness (i.e., fitness) of each individual in solving the problem in consideration. Whenever an individual is accepted as a solution to the problem, the algorithm terminates. Otherwise, the evolutionary process continues by selecting a subset of individuals for producing new individuals through mating. Mating or crossover aims to create a new population of more appropriate individuals w.r.t. the specific evaluation function by combining parent individuals. Next, the algorithm applies small random modifications (i.e., mutations) in parts of the individuals to ensure that diverse portions of the search space are explored. Finally, the new generation of individuals replaces the initial population and the evolutionary process starts again. Commonly, only a limited number of generations (i.e., iterations of the algorithm) is performed.

In the context of QBE, the biologically-inspired search performed by GP has the goal of finding a computer program that describes query predicates to compose the TSD. This search is similar to the approach of evolving classification rules that accurately assign a class or category to unforeseen data observations [8], [11].

3.3. Classification and Regression Trees

Classification and Regression Trees (CART) are one of the most popular methods used to construct classification models in data mining and analytics [12]. CART mine data in its raw form so that missing data, as well as continuous and nominal attribute values, are straightly handled without the need for costly preprocessing steps such as binning and feature normalization. Additionally, classifiers generated by CART are represented as a set of comprehensible rules readily understandable for human users.

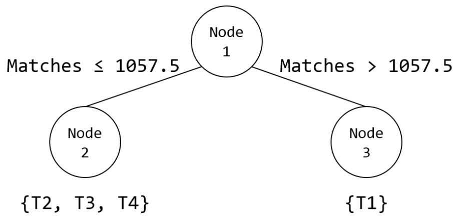


Figure 3. CART decision tree expressing the predicate that retrieves tuples {`T2, T3, T4`} as described in the our motivating example (Section 1).

CART recursively produces binary partitions of the data in the form of `IF condition THEN` data instance goes *Left*, `ELSE` data instance goes *Right*, where `condition` has the form of $Column \leq Value$. Figure 3 illustrates a hypothetical CART decision tree produced from the example described in Section 1. As depicted in the figure, every partition splits data tuples between two tree nodes following the `IF-THEN-ELSE` rule discovered by the algorithm.

Data columns and values composing the binary partitions are selected based on a splitting criterion such as the Gini index or the Entropy criterion. CART aims to find the data attribute which produces a partition that minimizes the splitting criterion (e.g., a partition that reduces the Gini index to 0).

In QBE, the decision trees produced by CART are used to construct the TSD. Similarly to GP, trees express classification rules describing whether a data observation (i.e., data tuple) is part of the data example provided by the user [4].

3.4. Evaluating candidate queries

For evaluation the appropriateness of discovered queries, we followed the work of [13] by combining two widely used metrics, i.e., sensitivity and specificity. *Sensitivity* measures how well a query performs in retrieving all the example tuples and *specificity* assesses how precise query results are from the exact set of example tuples specified in the input. Equations 1 and 2 show that both metrics can be computed by considering:

- True positives (TP): The number of tuples returned by the query that were in fact contained in the example tuples;
- True negatives (TN): The number of tuples that are not contained in the example tuples and were also not retrieved by the query;
- False positives (FP): The number of tuples returned by the query that were not contained in the example tuples;
- False negatives (FN): The number of tuples in the example tuples that were not retrieved by the query.

$$Sensitivity = \frac{TP}{(TP + FN)} \quad (1)$$

$$Specificity = \frac{TN}{(TN + FP)} \quad (2)$$

Ideally, appropriate queries must present sensitivity equals to 1.0 (i.e., all tuples in the data example are returned by the query) and specificity close to 1.0 (i.e., a small amount of tuples not described in the data example is returned in query results). In this setting, GP and GS can be applied to discover queries that maximize the fitness or evaluation function in Equation 3.

$$f(Q) = Sensitivity \times Specificity \quad (3)$$

Moreover, as concise queries are likely to be more easily interpreted, and consequently preferred by non-database experts, discovered queries can be also evaluated by their conciseness. In this context, *conciseness* can be expressed as $1/l$, where l represents the number of conditional predicates in the `WHERE` clause of an SQL query.

4. Experiments and Results

In this section, we describe two experiments to analyze the effectiveness of the selected intelligent algorithms in the tasks of discovering unknown and alternative queries from data examples. Implementations and experiments were conducted in the Python programming language using a 2.83 GHz Intel Core 2 Quad with 4 GB RAM.

We employed the Iris dataset[1] and the 1993 New Car Data[2] in our experiments, since both datasets describe real-world scenarios that are likely to be addressed by non-technical users (e.g., biologists and business persons). The selection of data examples employed in the experiments was carefully designed to consider diverse coverage of the two datasets. In this context, *coverage* refers to the number of tuples selected to construct a data example.

All experiments described in this work followed the same parameter setting. We employed a Genetic Programming algorithm with crossover and mutation rates configured as 0.9 and 0.1, respectively. The population size was set to 200 individuals, generated using the Ramped half and half method [10]. A tournament selection strategy of size six was employed in which new individuals were generated by a simple one-point crossover strategy. Moreover, a uniform mutation was used to perform changes in the individuals. The maximum number of generations was set to 100. In addition, all results reported in this section represent the average value obtained after 10 executions of each algorithm.

4.1. Discovering unknown queries

In this experimental case, we are interested in evaluating the effectiveness of the selected algorithms in discovering an unknown query that accurately reproduces a specific view of the data that was possibly manually created by a user. To the input of data examples, we randomly selected data tuples from the datasets according to the following coverage rates of 1, 15, 50, and 100 for the Iris dataset and 1, 10, and 25 for the 1993 New Car Data.

In our results, all algorithms achieved the maximum value of sensitivity (i.e., 1.0), which indicates their ability to cover all the tuples contained in the data examples. On the other hand, specificity values varied for both GP and GS, depending on the coverage rate of the examples as indicated in Figure 4 and Figure 5. Similarly, Figure 6 and Figure 7 report the resulting average of query conciseness output by the algorithms.

4.2. Discovering alternative queries

The task of discovering alternative queries is frequently performed to search for queries that are easier to understand or cheaper to compute while retrieving the same set of data tuples returned by an original query. Additionally, alternative queries may uncover hidden patterns and relationships

1. http://archive.ics.uci.edu/ml/datasets.html
2. http://ww2.amstat.org/publications/jse/datasets/93cars.txt

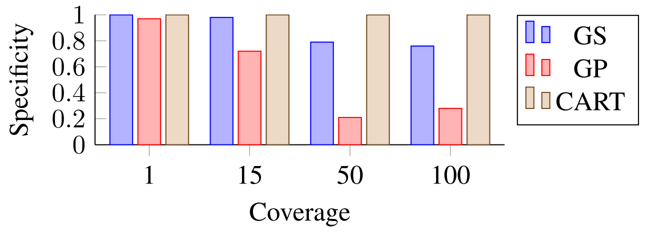


Figure 4. Average specificity achieved by the algorithms over different coverage rates of the Iris dataset.

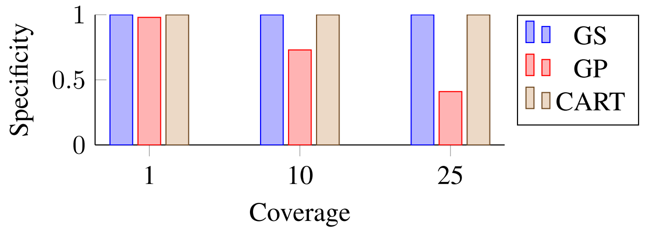


Figure 5. Average specificity achieved by the algorithms over different coverage rates of the Iris dataset.

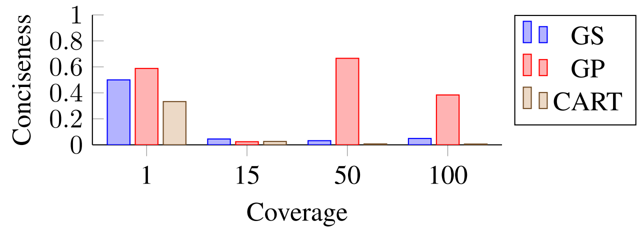


Figure 6. Average conciseness of the queries generated over the Iris dataset.

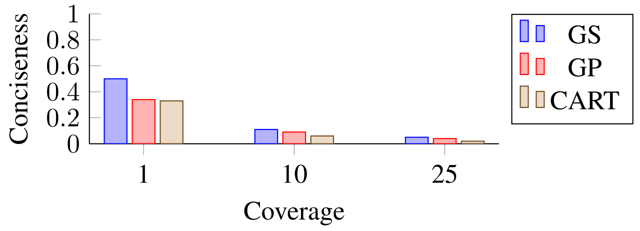


Figure 7. Average conciseness of the generated queries considering the 1993 New Car Data.

among the data that can be exploited in data mining and analytics projects. For instance, a discovered query may be employed to construct accurate data classifiers.

In this experiment, we designed a set of input queries whose resulting tuples are used as data examples. Again, to ensure that data examples present diverse values of the coverage factor, input queries were designed to output data examples of size 1, 13, 50, and 115. Input queries are reported in Table 2. Notice that we employed only the Iris dataset[3] in this case, as it appropriately contains a class label for each data tuple in the dataset. Also, for the sake of clarity, we only report discovered queries in which sensitivity reached the value of 1.0 because they are more likely to reveal hidden patterns that appropriately characterize the data examples.

TABLE 2. INPUT QUERIES EMPLOYED TO GENERATE DATA TUPLE EXAMPLES.

ID	Original query predicate	Coverage
Q1	petlen $\geq$ 4.0 AND petwid = 1.8 AND class = "Iris Versicolour"	1
Q2	seplen $>$ 6.5 AND petwid $\geq$ 1.3 AND petwid $\leq$ 1.9	13
Q3	class = "Iris Setosa"	50
Q4	class = "Iris Versicolour"	50
Q5	class = "Iris Virginica"	50
Q6	sepwid <> 3.0 AND petlen <> 1.4	115

5. Discussion

Building upon our prior research [8], our current results demonstrate that the selected algorithms can be effectively applied for QBE. Since sensitivity is easy to ensure (i.e., any method can reaching 100% sensitivity when returning all tuples in the dataset), specificity may be seen as a measure of diversity in query results. In this sense, only decision trees, particularly those generated by the CART algorithm, provided the most accurate results for QBE. In the employed datasets, CART was capable of discovering accurate queries that precisely retrieve all the provided data examples. However, given the binary-split strategy performed by CART, queries discovered by this algorithm tend to be much longer than the original queries, which may hinder user interpretation. Moreover, less concise queries may represent a form of data overfitting (i.e., a loss of generalization of the model) and real-world applications may require a predefined degree of diversity in query results. In this sense, CART may not be appropriate in data exploration scenarios where users are often interested in finding tuples sharing some degree of similarity with the provided data examples.

On the other hand, due to their heuristic nature, both GS and GP can be easily customized to allow some sort of diversity in both the structure of discovered queries (e.g., diverse predicates) and their respective results. Also, as long queries can be penalized during the search process (e.g., by incorporating the conciseness factor into the evaluation function), GS and GP allow users to find a personalized balance between accuracy and diversity. For instance, an Interactive Evolutionary Computation [14] approach may permit users to interactively incorporate their preferences during the search phase, which is not possible when using the traditional CART algorithm. Moreover, a thorough study of the impact of GP parameters is recommended to analyze whether improved results may be achieved.

Despite its preliminary character, the reported results indicate a trade-off that as the number of randomly selected data examples increase, the difficulty of finding a classification rule that accurately describes and retrieves all tuples also increases. In this sense, a user experiment is required to investigate whether this effect is caused by the randomness of the example selection.

We acknowledge that the limitations of this work are two-fold. First, the conducted experiments considered only single-relation databases. We did not consider more complex experiment scenarios in which many data tables are employed, and consequently, join queries are required. Second, this work focused on simple data retrieval queries that use only a subset of SQL features. In this sense, no aggregation queries were contemplated. We plan to incorporate both additional SQL features and multi-relation databases into our future work.

6. Conclusion

In this paper, we reported our experimental investigation of how intelligent algorithms can be employed help the increasing number of non-database-expert users to discover accurate database queries without the need of dealing with the SQL syntax. As non-database-expert users frequently struggle to accurately translate their mental queries into database queries, we use of the Query By Example paradigm, in which database queries are automatically constructed via tuple examples provided by users.

We selected three intelligent algorithms, namely, a greedy search, a genetic programming, and a decision tree to test their performance in the context of QBE. These algorithms perform a search in a large space of all candidate queries that can be formulated to retrieve tuples from a database. Our findings show that the selected algorithms are suitable for allowing users to discovery unknown queries

3. For the sake of space and simplicity, original column names sepal length, sepal width, petal length, petal width, and class were renamed to seplen, sepwid, petlen, petwidth, and target, respectively.

TABLE 3. Alternative queries discovered by each algorithm.

Query	Algorithm	Sensitivity	Specificity	Conciseness	Discovered query predicate
Q1	GS	1.0	1.0	0.5	seplen $=$ 5.9 AND sepwid $>$ 3.0
	GP	1.0	0.91	0.625	target $=$ "Iris Versicolour" AND petwid $\leq$ 1.8
	CART	1.0	1.0	0.5	petwid $>$ 1.75 AND target $=$ "Iris Versicolour"
Q2	GS	1.0	1.0	0.5	seplen $\geq$ 6.6 AND petwid $<$ 2.0
	GP	1.0	0.88	0.74	Not found
	CART	1.0	1.0	0.5	seplen $>$ 6.55 AND petwid $\leq$ 1.95000004768
Q3	GS	1.0	1.0	1	petlen $\leq$ 1.9
	GP	1.0	1.0	1	petwid $<$ 1.0
	CART	1.0	1.0	1	petlen $\leq$ 2.45
Q4	GS	1.0	0.917	0.33	Not found
	GP	1.0	0.78	0.23	Not found
	CART	1.0	1.0	0.07	petlen $>$ 2.45 AND petwid $\leq$ 1.75 AND petlen $\leq$ 4.94 AND petwid $\leq$ 1.65 OR petlen $>$ 2.45 AND petwid $\leq$ 1.75 AND petlen $>$ 4.94 AND petwid $>$ 1.54 AND seplen $\leq$ 6.94 OR petlen $>$ 2.45 AND petwid $>$ 1.75 AND petlen $\leq$ 4.85 AND seplen $\leq$ 5.94
Q5	GS	1.0	0.88	0.2	Not found
	GP	1.0	0.793	0.83	Not found
	CART	1.0	1.0	0.06	petwid $\leq$ 1.75 AND petlen $\leq$ 4.94 AND petwid $>$ 1.65 OR petwid $\leq$ 1.75 AND petlen $>$ 4.94 AND petwid $\leq$ 1.54 OR petwid $\leq$ 1.75 AND petlen $>$ 4.94 AND petwid $>$ 1.54 AND petlen $>$ 5.44 OR petwid $>$ 1.75 AND petlen $\leq$ 4.85 AND seplen $>$ 5.94 OR petwid $>$ 1.75 AND petlen $>$ 4.85
Q6	GS	1.0	1.0	0.5	sepwid $<>$ 3.0 AND petlen $<>$ 1.4
	GP	1.0	0.47	0.32	Not found
	CART	1.0	1.0	0.11	sepwid $\leq$ 2.95 AND petwid $>$ 0.25 AND sepwid $>$ 2.95 AND sepwid $>$ 3.04 AND petlen $\leq$ 1.45 AND petlen $\leq$ 1.34 AND sepwid $>$ 2.95 AND sepwid $>$ 3.04 AND petlen $>$ 1.45

from sets of tuple examples of diverse sizes. More specifically, our experiments demonstrate that CART decision trees are capable of generating interesting database queries. However, queries discovered by CART achieved low values of conciseness, outputting long queries.

We argue that the promising results obtained in this work, may draw the researchers' attention to investigate different Computational Intelligence algorithm such as Ant Colony Optimization and Fish School Search in the context of QBE. Moreover, future research venues include the application of Interactive Evolutionary Computation [14] to improve Genetic Programming results and appropriately deal with a user's subjective evaluation of generated queries.

Acknowledgment

This research is financially supported by the Brazilian National Council for Scientific and Technological Development (CNPq) – Science without Borders Program.

References

[1] A. Nandi and H. Jagadish, "Guided interaction: Rethinking the query-result paradigm," *Proceedings of the VLDB Endowment*, vol. 4, no. 12, pp. 1466–1469, 2011.

[2] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu, "Making database systems usable," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07. New York, NY, USA: ACM, 2007, pp. 13–24.

[3] M. M. Zloof, "Query-by-example: the invocation and definition of tables and forms," in *Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA.*, 1975, pp. 1–24.

[4] Q. T. Tran, C.-Y. Chan, and S. Parthasarathy, "Query reverse engineering," *The VLDB Journal*, vol. 23, no. 5, pp. 721–746, 2014.

[5] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas, "New trends on exploratory methods for data analytics," *Proc. VLDB Endow.*, vol. 10, no. 12, pp. 1977–1980, Aug. 2017.

[6] S. Zhang and Y. Sun, "Automatically synthesizing sql queries from input-output examples," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE'13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 224–234.

[7] A. Bonifati, U. Comignani, E. Coquery, and R. Thion, "Interactive mapping specification with exemplar tuples," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: ACM, 2017, pp. 667–682.

[8] D. M. L. Martins, F. B. d. L. Neto, and G. Vossen, "Learning database queries via intelligent semiotic machines," in *Proceedings of the 2017 IEEE Latin American Conference On Computational Intelligence (LA-CCI).* IEEE, Nov 2017.

[9] J. Cumin, J.-m. Petit, V.-m. Scuturici, and S. Surdu, "Data exploration with sql using machine learning techniques," in *Proceedings of the nternational Conference on Extending Database Technology (EDBT)*, Mar 2017, pp. 96–107.

[10] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, no. 2, pp. 87–112, June 1994.

[11] T. Helmuth and L. Spector, "Evolving sql queries from examples with developmental genetic programming," in *Genetic Programming Theory and Practice X.* Springer, 2013, pp. 1–14.

[12] D. Steinberg and P. Colla, "Cart: classification and regression trees," *The top ten algorithms in data mining*, vol. 9, p. 179, 2009.

[13] N. Holden and A. A. Freitas, "A hybrid particle swarm/ant colony algorithm for the classification of hierarchical biological data," in *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE.* IEEE, 2005, pp. 100–107.

[14] H. Takagi, "Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation," *Proceedings of the IEEE*, vol. 89, no. 9, pp. 1275–1296, 2001.