Deriving Queries from Results using Genetic Programming

Tae-Wan Ryu and Christoph F. Eick

Department of Computer Science
University of Houston
Houston, Texas 77204-3475
{twryu,ceick}@cs.uh.edu

Abstract

This paper centers on the problem of finding commonalities for a set of objects belonging to an object-oriented database. In our approach, commonalities within a set of objects are described by object-oriented queries that compute this set of objects. The paper discusses the architecture of a knowledge discovery system, called MASSON, which employs genetic programming to find such queries. We also report on an experiment that evaluated the knowledge discovery capabilities of the MASSON system.

1. Introduction

A database consists of extensional information and intensional information. Extensional information is physically stored data or database instances. Intensional information is the descriptions and the high-level abstracts of a database such as schema, relationships, and other implicit information. Most traditional database systems have focused on storing, maintaining and accessing the extensional information efficiently in databases. Extracting intensional information with respect to an extensional data collection is relatively difficult using conventional database systems. For example, suppose there is a police suspect database which contains information about persons and their activities, and a police officer has two drug-dealer suspects {Joe, Mary} who may be involved in a particular case. Then the police data analyst may be interested in knowing ***"What do Joe and Mary have in common ?"*** In order to answer this question, the data analyst has to conduct a time-consuming search process in which he has to find a query or a set of queries that returns the exactly same instances, {Joe, Mary} in this case, as its result. Suppose he found the following SQL form of query,

" (SELECT ssn name address
FROM person purchase
WHERE (amount-spent > 1000) and (payment-type = 'cash')
and (store-name = 'flea-market'))"

which returns {Joe, Mary} as its result, which states that they both have spent more than $1,000 cash for shopping in a 'flea-market'. This information might lead the police to investigate suspicious activities in flea-markets. However, in the example we have given in the above, it is not very obvious what kind of queries the user has to write. Accordingly, an automatic tool that facilitates the task for the data analyst is desirable. This paper will describe such a tool for extracting intensional information in an object-oriented database.

2. Deriving Queries From Results

Deriving queries from results is the process of finding a desired query or a set of queries from a set of objects. In this approach, the user's role is not to derive a query but the knowledge discovery in databases (KDD) system (Piatetsky & Frawley 1991) will derive a query or a set of queries by accessing database schema information as well as database instances through the database interface. The user of the KDD-system does not need in-depth knowledge about the database schema. We claim that this approach actually discovers useful or interesting intensional information implicitly stored in a database. We are developing a prototype system called MASSON that employs *deriving queries from results* approach in the context of object-oriented database. Figure 1 depicts the architecture of the system. MASSON takes a database name and object set (or database instances) as its input and accesses the database given from a user for domain knowledge and schema information. The user may also supply domain knowledge to restrict the search space if possible. MASSON uses genetic programming (GP) (Koza 1990) to generate many different queries and to search a query or a set of queries that describe the commonalities of the given object set. The generated queries are sent to an object-oriented database management system (OODBMS) for execution. The system will then evaluate those returned results from OODBMS according to how well they cover the given target object set. Based on the results of the evaluation process the GP search engine will generate new queries based on the principles of evolution by giving fitter query a better chance to reproduce. In order to evaluate an individual query q_i in a population, we use the following fitness function f: $f(q_i) = T - (h_i * h_i)/n_i$, where $n_i > 0$, $T \geq h_i$, and $i =$ 1, 2, ... population size. (T is the cardinality of the set of objects whose commonalities have to be determined, h_i is the number of hits for an individual query q_i, n_i is the cardinality of query q_i's result) This function is our standardized fitness function (Koza 1990), which means the smaller the fitness value,

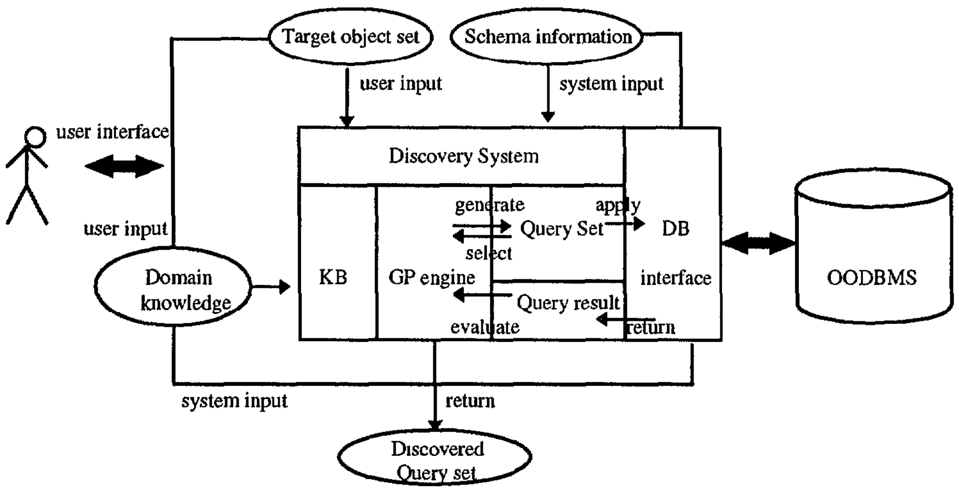

Figure 1: Architecture of MASSON

the fitter the individual is. The above fitness function depends on the number of hits and the cardinality of the query result. If a query does not make any hit ($h_t = 0$), then it has the value T, which is the worst fitness value. If, on the other hand the result of query q_t perfectly matches the set of objects given to the system, then its fitness value $f(q_t)$ is 0 ($h_t = n_t = T$ in this case). We call this case *perfect hit*. However, if a query is too general, it may contain superfluous instances (false positives) even if it made 100% of hits. If, on the other hand, a query is too specific then the number of hits of the query is less than the number of target instances – the difference is the number of false negatives. To cope with false positives and false negatives, we put the number of instances for an individual query, n_t as denominator. Moreover, because we take the square of the number of hits, false positives are punished less severely than false negatives by the fitness function: we are more interested in queries that return the target set or at large portions of the target set, even if they return objects that do not belong to the target set.

Object-oriented queries are used to describe commonalities among objects in our approach. The supported navigational query operators include *SELECT*, *RESTRICTED*, *RELATED*, *GET-RELATED*, and set operators (Ryu & Eick 1996). *SELECT* operator selects all the objects that satisfy the conditional predicate. *RESTRICTED* operator restricts the objects in the given set to those that are related to another class, according to the given predicate. The relationship operator *RELATED* selects all the objects from a class that are related to objects in another class through the relationship links. *GET-RELATED* operator is an inverse operator of RELATED. In addition, the set operators *UNION*, *INTERSECTION*, *DIFFERENCE* are supported.

The schema diagram shown in Figure 2 represents our experimental object-oriented database that contains information of persons and their related activities. Each class (or entity set) has slots and their values. A relationship or reverse relationship links a class to another class. The MASSON system was implemented by the PCL (Portable Common Loops) version of the CLOS (Common Lisp Object System) (Paepcke 1993) implementation.

We used GP as a search engine for MASSON (Ryu & Eick 1996). GP searches for a target program in the space of all possible computer programs (queries in our application) randomly composed of functions (query operators) and terminals (basic arguments for each operator) appropriate to the problem domain. Initially, a pre-defined number of queries that are syntactically legal, are generated by randomly selecting operators and their arguments from the function set and the terminal set respectively, forming the initial population. Each individual is evaluated based on the fitness function f. Fitter queries are then selected with higher probability to breed a new population, using three genetic operators: selection, crossover, and mutation. The selection operator is used to choose certain individuals based on their fitness values for generating the next generation. The crossover operation creates two new offsprings by exchanging subtrees between the two parents if we represent a query as a tree. The mutation operation produces a new offspring by replacing one parent subtree with a newly created subtree. The size, shape, and structures of queries can be dynamically changed when crossover or mutation operators are applied to each pair of selected queries (parent queries) during the breeding process. The selection-crossover-mutation cycle is repeated until an user defined termination criteria are satisfied.

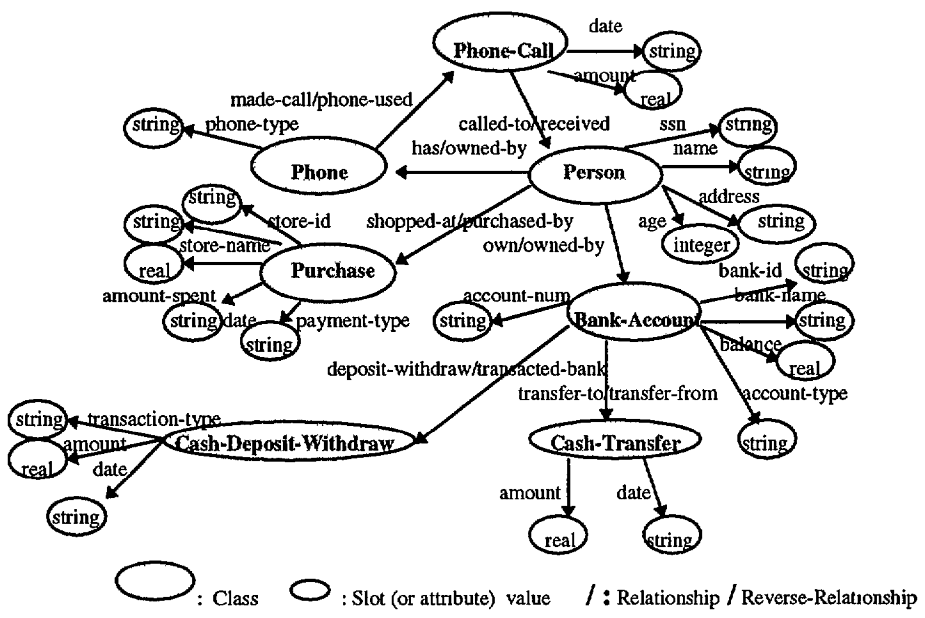

Figure 2: Schema Diagram for Personal Database

3. MASSON at Work

To demonstrate how MASSON works, and to evaluate the knowledge discovery capabilities of MASSON, we manually created 5 benchmark queries, Q1 ~ Q5.

Q1: *"Select persons who have transferred cash to anyone more than 4 times."*

```
(GET-RELATED (RESTRICTED bank-account (> transfer-to 4))
    owned-by
    person)
```

An object set that Q1 computes for a given example database consists of 22 persons. This object set was then used as the target object set when running the MASSON system. The query discovery process starts by generating 100 queries randomly. At generation 33, MASSON found the original query Q1. This was the first query that made a *perfect hit*, which is the case of $f_{33} = 0$ *(since $T=h=n=22$).* At generation 47, the average fitness value was significantly decreased to 2.83, and the best query that also made a perfect hit and has different semantics was:

"Select persons who have transferred cash to anyone more than 4 times and are age less than 30 or greater than 20."

```
(GET-RELATED (RESTRICTED BANK-ACCOUNT (> TRANSFER-TO 4))
              OWNED-BY
              (SELECT PERSON (OR (< AGE 30) (> AGE 20))))
```

The predicate in boldface is an additional predicate. Another perfect hit query was also found at generation 81.

Q2: *"Select persons who live in Houston."*

```
    (SELECT person (= address "Houston"))
```

The first perfect hit query to the object set provided by Q2 was found at generation 122. The query and the semantics were:

"Select persons who have telephone and live in Houston."

```
(GET-RELATED BANK-ACCOUNT OWNED-BY
   (SELECT (RELATED PERSON HAS PHONE) (= ADDRESS "Houston")))
```

The query Q2 was found at generation 135. Several other perfect hit queries that was different from Q1 was found.

Q3: *"Select persons who purchased more than $3,000 by cash in one store."*

```
(RELATED person shopped-at
  (SELECT purchase (AND (> amount-spent 3000)
                        (= payment-type 1))))
```

The first perfect hit query was found at generation 70. Even if the exactly same query Q3 was not found during 200 generations, syntactically and semantically almost similar query to Q3 was found at generation 94:

```
(RELATED PERSON SHOPPED-AT
   (SELECT (SELECT PURCHASE (= PAYMENT-TYPE 1))
           (> AMOUNT-SPENT 3002)))
```

The only difference between the discovered query and the query Q3 is the amount 3002 showed in boldface which is different from 3000. Actually, it is not possible to find the exact value 3000 unless the value is stored in the database, since MASSON only generates constants that appear in the database. Therefore we assume it found a query that is approximately the same as the original query Q3. Several other different perfect hit queries were also found.

Q4: *"Select persons that have more than 7 times of suspect activities records or spent more than $5,000 cash in a store."*

```
(O-UNION
   (RELATED person shopped-at
      (SELECT purchase (AND (> amount-spent 5000) (= payment-type 1))))
   (SELECT person (> nsuspect-act 7)))
```

Q4 is relatively a complex query. The first query that made perfect hit was found at generation 95:

"Select persons that have more than 7 times of suspect activities records or paid cash when shopping."

```
(O-UNION
   (SELECT PERSON (> NSUSPECT-ACT 7))
   (RELATED (O-UNION PERSON (SELECT (RESTRICTED (RESTRICTED
            PERSON (>= RECEIVED 3)) (< HAS 0))  (> NSUSPECT-ACT 7)))
     SHOPPED-AT
    (SELECT (GET-RELATED PERSON SHOPPED-AT (GET-RELATED
       PERSON SHOPPED-AT PURCHASE)) (= PAYMENT-TYPE 1))))
```

Other perfect hit queries was found. However, MASSON could not find the original query Q4 for 200 generations.

Q5: *"Select persons who have transferred more than $2,000 and have called more than 4 times and spent more than $500 cash."*

```
(O-INTERSECTION
   (GET-RELATED (GET-RELATED
     (SELECT cash-transfer (> amount 2000))
     transfer-from bank-account) owned-by person)
   (GET-RELATED (RESTRICTED (GET-RELATED
     (SELECT phone-call (> amount 500)) phone-used phone)(> made-call 4))
     owned-by person)))
```

This is another complex query. The first query that made perfect hit to the object set provided by the query Q5 was found at generation 112:

"Select persons who have telephone and have called to two persons not at 11/10/1987 and have made phone calls more than 4 times."

```
(GET-RELATED PHONE
  OWNED-BY
 (GET-RELATED (RESTRICTED (GET-RELATED PHONE-CALL PHONE-USED
   (GET-RELATED PHONE-CALL  PHONE-USED  (GET-RELATED (SELECT
   (RELATED (RESTRICTED PHONE-CALL (= CALLED-TO 2)) PHONE-USED
   (RELATED (GET-RELATED PHONE-CALL PHONE-USED PHONE)
      MADE-CALL (RELATED PHONE-CALL CALLED-TO  PERSON)))
   (NOT (EQUAL DATE "11 10 87")))  PHONE-USED PHONE)))
   (> MADE-CALL 4)) OWNED-BY
 (GET-RELATED (RESTRICTED (GET-RELATED PHONE-CALL
   PHONE-USED PHONE) (> MADE-CALL 3))  OWNED-BY PERSON)))
```

MASSON could not find the original query Q5 during 200 generations; however, it found several other queries that made perfect hit. These queries are semantically and syntactically different from Q5. This is an interesting discovery by MASSON since the input object set was obtained by Q5 consisting of a set operator, INTERSECTION. On the other hand, MASSON found those queries that consist of only selection and relationship operators. The semantics of those queries are also different from Q5. Table 1 shows summary for this experiment.

Query	Hit-at	Org	Othr		Nclass	Nvsl	Nop	Nsop	Ncop
Q1	33	Y	2		2	1	2	0	1
Q2	122	Y	6		1	1	1	0	1
Q3	70	Y	4		2	2	2	0	2
Q4	95	N	5		2	3	3	1	3
Q5	112	N	7		5	3	7	1	3

Table 1: Execution results and the complexity for the 5 test queries

The left part of Table 1 shows the benchmark results for the 5 test queries and the right part of the table shows relative complexity of each test query although these are not a precise metrics of complexity measure. In this table, Hit-at is the generation number that made first perfect hit. Org shows whether MASSON found the original query or not. Othr is the number of other queries found that made perfect hit other than the original query. Nclass is the number of classes that each test query traversed. Nvsl, Nop, Nsop, and Ncop are the number of slots, query operators, set operators, and conditional operators respectively. According to the table, the query Q4 and Q5 are relatively more complex than Q1, Q2, and Q3. MASSON could not find the exactly same queries as those queries Q4 and Q5 for 200 generations but found other queries that describe commonalities within the given target set.

4. Summary

In this paper, we proposed a problem on how to extract intensional information or concept descriptions for a set of objects from a database without other knowledge about the given object set. The major contribution of this paper is the presentation of a new approach to discovering commonalities within a given set of objects. We dealt with this problem by introducing *deriving queries from results* approach in which we try to find an object-oriented query that returns a given result or set of results for a given database. These derived queries present intensional information for the given set of objects. MASSON takes a database name and a set of objects belonging to the database as its input and returns a query or a set of queries as the result of the search process. We presented an example that demonstrated how MASSON works, and we reported on an experiment that evaluated MASSON's knowledge discovery capabilities.

References

Ryu, T.W and Eick, C.F. 1996. MASSON: Discovering Commonalities in Collection of Objects using Genetic Programming. *In Proceedings of the Genetic Programming 1996 Conference.*

Koza, John R. 1990. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: The MIT Press.

Paepcke, Andreas 1993. *Object-Oriented Programming: the CLOS perspective.* Cambridge, MA: The MIT Press.

Piatetsky-Shapiro, G. and Frawley, W.J. 1991. Knowledge Discovery in Databases: An Overview. *Knowledge Discovery in Databases.* AAAI/The MIT press. Pages1-27.