Applying Theory Revision to the Design of Distributed Databases

Fernanda Baião[1], Marta Mattoso[1], Jude Shavlik[2], Gerson Zaverucha[1]

[1] Department of Computer Science – COPPE, Federal University of Rio de Janeiro (UFRJ)
PO Box 68511, Rio de Janeiro, RJ 21941-972 Brazil
{baiao, marta, gerson}@cos.ufrj.br
[2] Computer Sciences Department, University of Wisconsin-Madison
1210 West Dayton Street, Madison, WI 53706 USA
shavlik@cs.wisc.edu

Abstract. This work presents the application of theory revision to the design of distributed databases to automatically revise a heuristic-based algorithm (called analysis algorithm) through the use of the FORTE system. The analysis algorithm decides the fragmentation technique to be used in each class of the database and its Prolog implementation is provided as the initial domain theory. Fragmentation schemas with previously known performance, obtained from experimental results on top of an object database benchmark, are provided as the set of examples. We show the effectiveness of our approach in finding better fragmentation schemas with improved performance.

1 Introduction

Distributed and parallel processing on database management systems are efficient ways of improving performance of applications that manipulate large volumes of data. This may be accomplished by removing irrelevant data accessed during the execution of queries and by reducing the data exchange among sites, which are the two main goals of the design of distributed databases [28]. However, in order to improve performance of these applications, it is very important to design information distribution properly.

The distribution design involves making decisions on the fragmentation and placement of data across the sites of a computer network. The first phase of the distribution design is the fragmentation phase, which is the focus of this work. To fragment a class of objects, it is possible to use two basic techniques: horizontal and vertical fragmentation [28], which may be combined and applied in many different ways to define the final fragmentation schema.

The class fragmentation problem in the design of a distributed database is known to be an NP-hard problem [28]. There are a number of works in the literature addressing the horizontal [7, 14, 31] or vertical [6, 15] class fragmentation technique, but not both. Even when the designer decides to use a horizontal fragmentation algorithm to one class and a vertical fragmentation algorithm to

another class, he is left with no assistance to make this decision. Our previous work proposed a set of heuristics to drive the choice of the fragmentation technique to be applied in each class of the database schema. Those heuristics were implemented in an algorithm called "analysis algorithm" [2], and were incorporated in a methodology that includes the analysis algorithm, horizontal and vertical class fragmentation algorithms adapted from the literature. Experimental results reported in [3, 4] show applications that were executed 3.4 times faster when applying the fragmentation schema resulted from our methodology, compared to other alternative fragmentation schemas proposed by other works in the literature.

Experimental results from real applications can continuously provide heuristics for the design of distributed object databases (DDODB) that may be incorporated in our analysis algorithm. Indeed, we have tried to manually improve the analysis algorithm using experimental results from [23, 25], which required a detailed analysis of each result and manual modifications on the analysis algorithm. However, the formalization of new heuristics from these experiments and their incorporation in the analysis algorithm, while maintaining previous heuristics consistent, proved to be an increasingly difficult task.

This work proposes the use of Theory REvisioN on the Design of Distributed Databases (TREND3), showing how it automatically improves our analysis algorithm through the use of the FORTE system [29]. TREND3 is a module of a framework that handles the class fragmentation problem of the design of distributed databases, defined in [5].

There are approaches in the literature addressing the DDODB problem [4, 6, 7, 13, 14, 15, 16, 20, 24, 31]. However, none of them addresses the problem of choosing the most adequate fragmentation technique to be applied to each class of the database schema. Some works have been applying machine learning techniques to solve database problems. For example, [8, 9] present an approach for the inductive design of deductive databases, based on the database instances to define some intentional predicates. Also, relational bayesian networks were used to estimate query selectivity in a query processor [19] and to predict the structure of relational databases [18]. However, considering the design of distributed databases as an application for theory revision is a novel approach.

The paper is organized as follows: in section 2, the design of distributed databases is defined and our framework for the design of distributed databases is described. Theory revision is briefly reviewed in section 3, while in section 4 we show how to improve a DDODB analysis algorithm through the use of the FORTE system. Experimental results on top of the OO7 benchmark [12] are presented in section 5. Finally, section 6 presents some conclusions and future work.

2 A Framework for the Design of Distributed Databases

This section defines the problem of designing a distributed database, focusing on the object-oriented model, and presents a framework we propose for the class fragmentation phase of the distribution design.

2.1 The Design of Distributed Databases

The distribution design of a database makes decisions on the fragmentation and placement of data across the sites of a computer network. The first phase of the distribution design is the fragmentation phase, which is the process of isolating into fragments specific data accessed by the most relevant applications that run over the database.

In an object-oriented database, data is represented as objects. The set of objects sharing the same structure and behavior define a class, and classes may be related to each other through relationships. A database schema describes the set of classes and relationships. The UML diagram representing the database schema of the OO7 benchmark [12] is illustrated in figure 1. The OO7 benchmark is a generic application on top of a database of design objects assembled through the composition of parts. We may notice, for example, that each composite part is related to N atomic parts (through the "parts" relationship), and that each atomic part "is part of" one composite part.

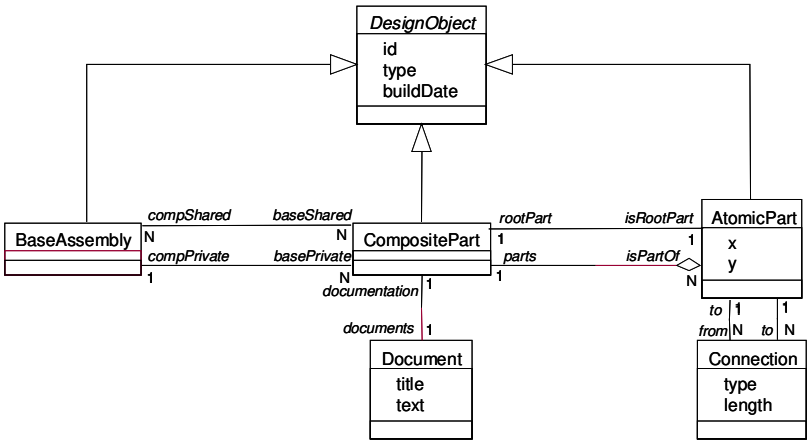

Fig. 1. The OO7 benchmark database schema

Given the schema of the database to be distributed, as in any distribution design methodology, we need to capture the set of operations over the database and quantitative information in order to define a fragmentation schema to be applied on the database schema, which is the goal of the fragmentation phase.

The operations are captured by decomposing the application running over the database, and are classified into selection, projection or navigation operations according to the definitions from [2]. Quantitative information needed includes the cardinality of each class (i.e., its estimated size: small, medium or large) and the execution frequency of each operation.

The fragmentation schema is composed of the choice of a fragmentation technique and the definition of a set of fragments for each class of the database schema. The two basic fragmentation techniques to be applied on a class are horizontal and vertical fragmentation [28]. Vertical fragmentation breaks the class logical structure (its attributes and methods) and distributes them into fragments. Horizontal fragmentation distributes class instances across the fragments. Thus, a horizontal fragment of a class contains a subset of the whole class extension. Horizontal fragmentation is usually subdivided into primary and derived horizontal fragmentation. Primary horizontal fragmentation basically optimizes selection and projection operations, while derived horizontal fragmentation addresses the relationships between classes and improves performance of navigation operations. It is also possible to apply both vertical and primary horizontal fragmentation techniques to a class simultaneously (which we call hybrid fragmentation) or to apply different fragmentation techniques to different classes in the database schema (which we call mixed fragmentation).

In the object oriented data model, additional issues contribute to increase the difficulty of the class fragmentation and turn it into an even more complex problem. Our previous work proposed a set of heuristics implemented by an algorithm (called "analysis algorithm") [2]. Some examples of the heuristics proposed are "*in the case of a selection operation on a class with a large cardinality, this class is indicated to primary horizontal fragmentation*", or "*in the case of a projection operation on a class with a large cardinality that is not derived horizontally fragmented, this class is indicated to vertical fragmentation*". The algorithm was also capable of handling conflicts during the fragmentation schema definition.

2.2 The Framework for the Class Fragmentation Problem in the DDODB

The framework we propose for the class fragmentation problem in the design of distributed databases integrates three modules: the DDODB heuristic module, the theory revision module (TREND3) and the DDODB branch-and-bound module (figure 2).

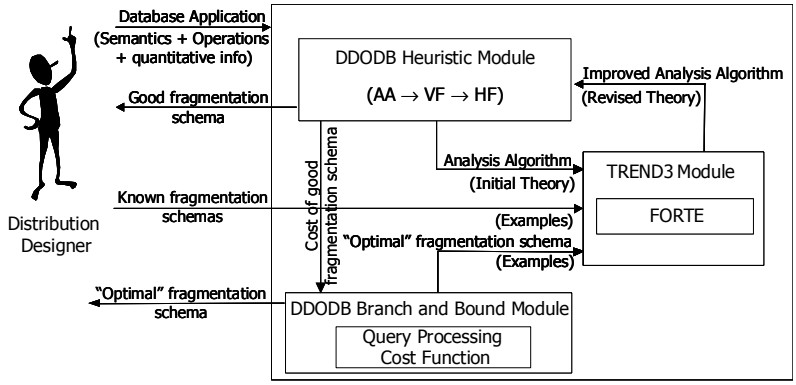

Fig. 2. The overall framework for the class fragmentation in the DDODB

The distribution designer provides input information about the database schema (its semantics – classes and relationships – and additional quantitative information such as the estimated cardinality of each class) and applications (projection, selection and navigation operations) that will be executed over the database. This information is then passed to the DDODB heuristic module. The DDODB heuristic module defines a set of heuristics to design an adequate fragmentation schema for a given database application. The execution of the heuristic module algorithms (analysis algorithm, vertical fragmentation and horizontal fragmentation) will follow this set of heuristics and quickly output an adequate fragmentation schema to the distribution designer. Previous results using the heuristic module are presented in [2, 3].

The set of heuristics implemented by the DDODB heuristic module may be further automatically improved by executing a theory revision process through the use of inductive logic programming (ILP) [27, 29, 34]. This process is called Theory REvisioN on the Design of Distributed Databases (TREND3). The improvement process may be carried out by providing two input parameters to the TREND3 module: the Prolog implementation of the analysis algorithm (representing the initial theory, or the background knowledge) and fragmentation schemas with previously known performances (representing a set of examples). The analysis algorithm is then automatically modified by a theory revision system (called FORTE) so as to produce a revised theory. The revised theory will represent an improved analysis algorithm that will be able to output a fragmentation schema with improved performance, and this revised analysis algorithm will then substitute the original one in the DDODB heuristic module. In [26], it has been pointed out that machine learning algorithms that use background knowledge, thus combining inductive with analytical mechanisms, obtain the benefits of both approaches: better generalization accuracy, smaller number of required training examples, and explanation capability.

Additionally, the input information from the distribution designer may be passed to our third module, the DDODB branch-and-bound module. This module represents an alternative approach to the heuristic module, and obtains (at a high execution

cost) the best fragmentation schema for a given database application. The *branch-and-bound* procedure searches for an optimal solution in the space of potentially good fragmentation schemas for an application and outputs its result to the distribution designer. The algorithm bounds its search for the best fragmentation schema by using a query processing cost function during the evaluation of each fragmentation schema in the hypotheses space. This cost function, defined in [30], is responsible for estimating the execution cost of queries on top of a distributed database. The resulting fragmentation schema generated by the heuristic module is used to bound evaluations of fragmentation schemas presenting higher estimated costs. Finally, the resulting fragmentation schema generated by the branch-and-bound algorithm, as well as the fragmentation schemas discarded during the search, may generate examples (positive or negative) to the TREND3 module, thus incorporating the branch-and-bound results into the DDODB heuristic module.

3 Theory Revision

The theory revision task [34] can be specified as the problem of finding a minimal modification of an initial theory that correctly classifies a set of training examples. Formally, it is defined as shown in figure 3.

Given:	a target concept C a set P of positive instances of C a set N of negative instances of C a hypothesis language L an initial theory T expressed in L describing C
Find:	a revised theory RT expressed in L that is a minimal modification of T such that RT is correct on the instances of both P and N

Fig. 3.: The theory revision task

A theory is a set of (function-free) definite program clauses, where a definite program clause is a clause of the form of (1).

$$\alpha \leftarrow \beta 1 ,\ldots, \beta n . \qquad (\mathbf{1})$$

where α, $\beta 1$ ... βn are atomic formulae.

A concept is a predicate in a theory for which examples appear in the training set. An instance, or example, is an instantiation (not necessarily ground) of a concept. An instance of the concept "cardinality" is

```
cardinality( connection, large )
```

Each instance i has an associated set of facts F_i, which gathers all the instances of a concept in the training set. A positive instance should be derivable from the theory augmented with its associated facts, while the negative instances should not.

In the DDODB domain, the set of facts define a particular database schema definition (classes with their cardinalities, relationships – of a specific type - between

classes) and the applications (operations with their frequencies, classifications and their accessed classes) that run on the database.

```
class( atomicPart )
class( compositePart )
cardinality( atomicPart, large )
cardinality( compositePart, small )
relationship( rootPart )
relationshipType( rootPart, '1:1' )
relationshipAccess( rootPart, compositePart, atomicPart )
operation( o1, 100 )
classification( o1, projection )
accessedClasses( o1, [atomicPart] )
```

The correctness of a theory is defined as follows: given a set P of positive instances and a set N of negative instances, a theory T is correct on these instances if and only if (2) holds.

$$\forall\, p \in P\colon T \cup Fp \models p$$
$$\forall\, p \in N\colon T \cup Fp \not\models p\ . \qquad (2)$$

The revision process of an initial domain theory works by performing a set of modifications on it, in order to obtain a correct revised theory. The modifications performed on a theory are the result of applying revision operators that make small syntactic changes on it. A correct revised theory that is obtained through a minimal modification of the initial theory is achieved by minimizing the number of operations performed. By requiring minimal modification, we mean that the initial theory is assumed to be approximately correct, and therefore the revised theory should be as semantically and syntactically similar to it as possible.

Related works in the literature [10, 11] presented a detailed comparison among many theory refinement systems in the literature, concentrating in theory revision systems, which - in general - have better results than theory-guided systems. The analysis included systems such as FORTE [29], A_3 [33] and PTR+ [22]. The author proposed a framework for classifying theory revision systems and a methodology for evaluating how well an algorithm is able to identify the location of errors independently of its ability to repair them. The performance analysis on the FORTE system when compared to other in different domains demonstrated that it searches a larger space of revised theories, and thus may find a more accurate candidate than either PTR+ or A_3. Also, FORTE attempts to repair many more revision points than other systems, because it generates and evaluates more repair candidates. Therefore, the FORTE system was chosen to perform the improvement procedure of our DDODB algorithms.

FORTE (First Order Revision of Theories from Examples) is a system for automatically refining first-order Horn-clause knowledge bases. This powerful representation language allows FORTE to work in domains involving relations, such as our DDODB domain.

FORTE is a theory revision system, in the sense that it modifies incorrect knowledge by applying the "*identify and repair"* strategy. It performs a hill-climbing

search in the hypothesis space, by applying revision operators (both specialization and generalization) to the initial domain theory in an attempt to minimally modify it in order to make it consistent with a set of training examples. By doing that, FORTE preserves as much of the initial theory as possible. Furthermore, revisions are developed and scored using the entire training set, rather than just a single instance, which gives FORTE a better direction than if revisions were developed from single instances. More details on the FORTE system may be found in [10, 29].

4 Theory revision on the design of distributed databases

This section proposes a knowledge-based approach for improving the DDODB analysis algorithm through the use of theory revision. The goal of applying this knowledge-based approach is to automatically incorporate in the analysis algorithm changes required to obtain better fragmentation schemas. These improvements may be found through additional experiments, thus the theory revision can automatically reflect the new heuristics implicit on these new results.

In order to apply the FORTE system to the DDODB problem, we had to model and represent all relevant information from the DDODB domain in an adequate way as required by FORTE. This basically included representing both our initial domain theory and the set of examples.

4.1 The Initial Domain Theory

In our TREND3 approach, we use our analysis algorithm as the initial domain theory. The overall structure of our set of rules is presented in figure 4. The complete Prolog implementation of the analysis algorithm is shown in [1].

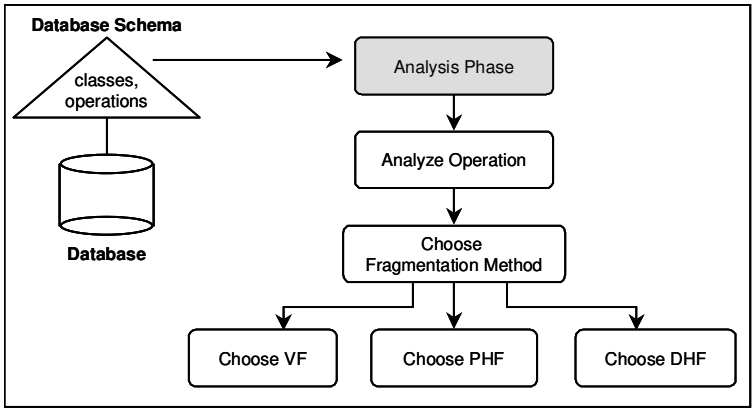

Fig. 4. The overall structure of our set of rules for the analysis algorithm

FORTE assumes that the initial domain theory is divided into two files: the "fundamental domain theory"(FDT) file (with predicates that are assumed correct) and the "initial theory to be revised"(THY) file (with predicates subject to the revision process).

The Fundamental Domain Theory. The FDT file contains one clause for each of the attributes and relations used in the examples (which are defined in the DAT file through the predicate `example/4`, explained later), plus one clause for each object type. Given a database schema and a set of applications, then objects, their attributes and the relations between objects are fixed and represent all the information that is needed by the analysis algorithm, and therefore need not be revised.

FORTE is responsible for decomposing the set of examples and create extensional definitions for these attributes, relations and objects that are accessed through the FORTE predicate `example/1` illustrated in figure 5. The FDT file contains predicates from the initial domain theory that FORTE is not allowed to revise, and is illustrated in figure 5.

The predicate `navigatesFromTo/3` from figure 5 defines if an operation navigates from one class X to another class Y (or vice-versa) in a path expression. Additionally, we had to create predicates `isNotDerivedFragmented/1` and `isNotVerticallyFragmented/1` due to the fact that negated literals (general logic programs) are not addressed by FORTE revision operators.

The Initial Theory To Be Revised. The THY file contains predicates from the initial domain theory for FORTE to revise (i.e., concepts from the analysis algorithm that may be modified), and is illustrated in figure 6.

```
/*** Object types that represent the database schema (classes
     and relationships) and are used in the examples
***/
class( X ) :- example( class( X ) ).
relationship( R ) :- example( relationship( R ) ).

/*** Object types that represent the operations (extracted
     from applications) and are used in the examples
***/
operation( O ) :- example( operation( O ) ).

/*** Attributes that qualify object types and are
     used in the examples
***/
  /* attributes for classes */
  cardinality( X, C ) :- example( cardinality( X, C ) ).
  fragmentation( C, F ) :- example( fragmentation( C, F ) ).

  /* attribute for relationships */
  relationshipType( R,T ) :- example(relationshipType( R, T
)).

  /* attributes for operations */
  frequency( O, F ) :- example( frequency( O, F ) ).
  classification( O, C ) :- example( classification( O, C ) ).

/*** Relations between object types that are used
     in the examples
***/
relationshipAccess( X,Y,Z ):-
example(relationshipAccess(X,Y,Z)).
operationAccess( O, L ) :- example( operationAccess( O, L ) ).
navigates( O, X, Y ) :- example( navigates( O, X, Y ) ).

/*** Predicates which appear in the initial theory to be
     revised, but which FORTE is not allowed to revise
***/
isDerivedFragmented( X ) :-
   fragmentation((_,X),derivedFragmentation).
isNotDerivedFragmented( X ) :-
   \+ isDerivedFragmented( X ).
isVerticallyFragmented( X ) :-
   fragmentation( X, vertical ).
isNotVerticallyFragmented( X ) :-
   \+ isVerticallyFragmented( X ).
navigatesFromTo( O, X, Y ) :-
   operationAccess( O, ClassPath ),
   member( X, ClassPath ),
   member( Y, ClassPath ),
   navigates( O, X, Y ).
navigatesFromTo( O, X, Y ) :-
   operationAccess( O, ClassPath ),
   member( X, ClassPath ),
   member( Y, ClassPath ),
   navigates( O, Y, X ).
```

Fig. 5. The fundamental domain theory

```
chooseDerivedHorizontalFragmentationMethod( Oi, X, Y ) :-
  fdt:classification(Oi,navigation),
  fdt:navigatesFromTo(Oi,Y,X),
  fdt:relationshipAccess(Name,X,Y),fdt:relationship( Name ),
  fdt:relationshipType(Name, 'N:1'),
  fdt:isNotVerticallyFragmented( X ),
  fdt:isNotDerivedFragmented( X ).
chooseDerivedHorizontalFragmentationMethod( Oi, Y, X ) :-
  fdt:classification(Oi,navigation),
  fdt:navigatesFromTo(Oi,X,Y),
  fdt:relationshipAccess(Name,X,Y),fdt:relationship( Name ),
  fdt:relationshipType(Name, '1:N'),
  fdt:isNotVerticallyFragmented( Y ),
  fdt:isNotDerivedFragmented( Y ).
chooseDerivedHorizontalFragmentationMethod( Oi, Y, X ) :-
  fdt:classification(Oi,navigation),
  fdt:navigatesFromTo(Oi,X,Y),
  fdt:relationshipAccess(Name,X,Y),fdt:relationship( Name ),
  fdt:relationshipType(Name,'1:1'),
  fdt:isNotVerticallyFragmented(Y),
  fdt:isNotDerivedFragmented( Y ).
choosePrimaryHorizontalFragment ationMethod( Oi, X ) :-
  fdt:classification(Oi, selection),
  fdt:operationAccess(Oi, [X]), fdt:cardinality( X, large ).
chooseVerticalFragmentationMethod( Oi, X ) :-
  fdt:classification(Oi, projection),
  fdt:operationAccess(Oi, [X|_]), fdt:cardinality( X, large ),
  fdt:isNotDerivedFragmented( X ).
```

Fig. 6: The initial theory to be revised

Intuitively, the clauses in figure 6 choose the fragmentation technique (derived horizontal, primary horizontal, vertical) to be applied to a class of the database schema according to the heuristics proposed in [2]. Hybrid fragmentation arises when both primary horizontal and vertical fragmentations are chosen, since their clauses are not exclusive.

4.2 The set of examples

Another essential information needed by FORTE for the theory revision process is the set of examples. For the TREND3 approach, they were derived from experimental results presented in [23, 25, 32] on top of the OO7 benchmark [12]. This benchmark describes a representative object oriented application and it has been used in many object database management systems to evaluate their performance in centralized environments. Unfortunately, there are no other performance results on top of distributed databases available in the literature, due to security or commercial reasons.

Each example represents a choice of the fragmentation technique to be applied to a class in a database schema. Positive/negative instances were generated by the choices that led to good/bad performance results in the distributed database. We obtained a total of 48 instances (19 positive and 29 negative).

The representation of an example in FORTE is an atomic formula as in (3),

```
example(PositiveInstances,NegativeInstances,Objects,Facts))   (3)
```

where PositiveInstance (NegativeInstance) is a list of positive (negative) facts of the concept to be learned, Objects are the representation of the application domain (in the DDODB domain, objects are represented as the classes and operations of the current application), and Facts are facts from the fundamental domain theory.

Figure 7 shows an example of choosing vertical fragmentation for class atomicPart, from the OO7 benchmark application, during the analysis of a projection operation.

```
example( [ chooseVerticalFragmentationMethod(o1,atomicPart)],
         [ ],
         [ class([ [designObject,  none,   none],
                   [baseAssembly,  small,  none],
                   [compositePart, small,  none],
                   [atomicPart,    medium, none],
                   [connection,    large,  none]
                 ]),
           relationship([ [componentsShared,  'N:N'],
                          [componentsPrivate, '1:N'],
                          [rootPart,          '1:1'],
                          [parts,             '1:N'],
                          [from,              '1:N'],
                          [to,                '1:N']
                        ]),
           operation([ [o1, projection] ])
         ],
         facts(
 [ relationshipAccess(compShared, baseAssembly,compositePart),
   relationshipAccess(compPrivate,baseAssembly,compositePart),
   relationshipAccess(rootPart, compositePart, atomicPart),
   relationshipAccess(parts, compositePart, atomicPart),
   relationshipAccess(from, atomicPart, connection),
   relationshipAccess(to, atomicPart, connection),
   query( q1, 100, [o1] ),
   operationAccess( o1, [atomicPart] ),
 ])
).
```

Fig. 7: A FORTE example from the OO7 benchmark application

In the example of figure 7, the positive instance is given by the term chooseVerticalFragmentationMethod(o1, atomicPart). There are no negative instances defined. The objects are the sets of classes, relationships and operations of the application, while the facts define existing relations between application objects (e.g.: which classes are accessed by each relationship, which operations compose a query, which classes are accessed by each operation). TREND3 examples are passed to FORTE in a data file (DAT). The DAT file contains examples from which FORTE will learn, and also defines execution parameters to guide the FORTE learning process. The complete description of the DAT file for the OO7 benchmark is in [1].

5 Experimental Results

This section presents experimental results of TREND3 on top of the OO7 benchmark, showing the effectiveness of our approach in obtaining an analysis algorithm that produces a better fragmentation schema for the OO7 benchmark application.

Due to the small amount of examples available, and to overcome the overfitting problem during training, we applied k-fold cross validation approach to split the input data into disjoint training and test sets and, within that, a t-fold cross-validation approach to split training data into disjoint training and tuning sets [26, 21]. The revision algorithm monitors the error with respect to the tuning set after each revision, always keeping around a copy of the theory with the best tuning set accuracy, and the saved "best-tuning-set-accuracy" theory is applied to the test set. The experimental methodology built in FORTE, which is currently a random resampling, was adapted to follow the one above.

The experiments were executed with k = 12 and t = 4. Each run was executed with a training set of 33 instances, a tuning set of 11 instances and a test set of 4 instances, and obtained a revised theory as its final result. In all k runs, the best-tuning-set-accuracy was 100%.

Table 1 shows the results of the execution of 12 independent runs, and therefore each result refers to a different revised theory proposed by FORTE. We verified that all proposed revised DDODB theories were identical, and represented the final revised DDODB theory (figure 8).

By comparing the definitions of choosePrimaryHorizontalFragmentationMethod/2 and chooseVerticalFragmentationMethod/2 predicates in figures 6 and 8, it may be verified that the following revisions were made by FORTE:

1) **Rule addition:** The following rule was added:

```
chooseVerticalFragmentationMethod(A,B):-
   cardinality(B,medium),classification(A,projection).
```

2) **Antecedent deletion:** The antecedent fdt:cardinality(B,large) was removed from the rule:

```
choosePrimaryHorizontalFragmentationMethod(A,B):-
   fdt:classification(A,selection),
   fdt:operationAccess(A,[B]).
   fdt:cardinality(B,large).
```

Intuitively, these modifications show that medium-sized classes are also subject to vertical fragmentation in the case of a projection operation, and that classes may have primary horizontal fragmentation independent of its size.

By running both versions of the analysis algorithm on top of the OO7 benchmark application, we notice that class atomicPart is indicated for hybrid fragmentation (primary horizontal + vertical) after the revision (instead of derived horizontal fragmentation), as illustrated in table 2.

Table 1. Summary of the FORTE execution output.

K	Initial Training Accuracy	Initial Test Set Accuracy	Final Training Accuracy	Final Test Set Accuracy
1	61.36	100.00	93.18	100.00
2	61.36	100.00	93.18	100.00
3	63.64	75.00	95.45	75.00
4	63.64	75.00	93.18	100.00
5	63.64	75.00	95.45	75.00
6	63.64	75.00	95.45	75.00
7	61.36	100.00	93.18	100.00
8	70.45	0.00	93.18	100.00
9	61.36	100.00	93.18	100.00
10	70.45	0.00	93.18	100.00
11	70.45	0.00	93.18	100.00
12	63.64	75.00	93.18	100.00

```
chooseDerivedHorizontalFragmentationMethod(A,B,C):-
   fdt:classification(A,navigation),
   fdt:navigatesFromTo(A,C,B),
   fdt:relationshipAccess(D,B,C), fdt:relationship(D),
   fdt:relationshipType(D,N:1),
   fdt:isNotVerticallyFragmented(B),
   fdt:isNotDerivedFragmented(B).
chooseDerivedHorizontalFragmentationMethod(A,B,C):-
   fdt:classification(A,navigation),
   fdt:navigatesFromTo(A,C,B),
   fdt:relationshipAccess(D,C,B), fdt:relationship(D),
   fdt:relationshipType(D,1:N),
   fdt:isNotVerticallyFragmented(B),
   fdt:isNotDerivedFragmented(B).
chooseDerivedHorizontalFragmentationMethod(A,B,C):-
   fdt:classification(A,navigation),
fdt:navigatesFromTo(A,C,B),
   fdt:relationshipAccess(D,C,B), fdt:relationship(D),
   fdt:relationshipType(D,1:1),
   fdt:isNotVerticallyFragmented(B),
   fdt:isNotDerivedFragmented(B).
choosePrimaryHorizontalFragmentationMethod(A,B):-
   fdt:classification(A,selection),
   fdt:operationAccess(A,[B]).
chooseVerticalFragmentationMethod(A,B):-
   cardinality(B,medium), classification(A,projection).
chooseVerticalFragmentationMethod(A,B):-
   fdt:classification(A,projection),
   fdt:operationAccess(A,[B|C]),
   fdt:cardinality(B,large), fdt:isNotDerivedFragmented(B).
```

Fig. 8. The revised analysis algorithm

Table 2. Fragmentation techniques chosen by both versions of the analysis algorithm

Class	Initial version	Revised version
baseAssembly	primary horizontal	primary horizontal
compositePart	derived horizontal	derived horizontal
atomicPart	**derived horizontal**	**hybrid**
connection	derived horizontal	derived horizontal

We then compared the costs of the resulting fragmentation schemas obtained from the initial and the revised versions of the analysis algorithm, after executing the vertical and horizontal fragmentation algorithms (those algorithms were not considered for the revision process).

These costs were calculated according to the cost model from [30], assuming that the query optimizer was able to choose the most efficient way of executing each query (that is, choosing the least cost between the "naïve-pointer", value-based join and pointer-based join algorithms). The resulting costs are illustrated in figure 9.

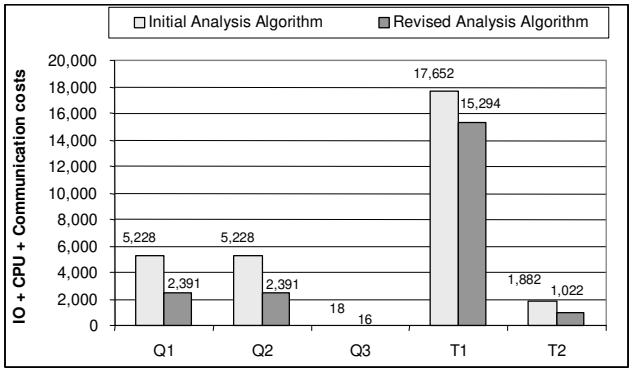

Fig. 9: Comparing the costs of the fragmentation schemas obtained from the initial and the revised analysis algorithm

Figure 9 shows the cost of executing each query from the OO7 benchmark application. The total cost of the OO7 benchmark application, according to the frequencies of each operation, can be calculated as:

Cost(OO7) = 100*cost(Q1) + 50*cost(Q2) + 10*cost(Q3) + 30*cost(T1) + 30*cost(T2)

Which produces the following costs for the two versions of the analysis algorithm that are being compared:

Cost_of_InitialVersion(OO7) = 1,370,410
Cost_of_RevisedVersion(OO7) = 848,297

Our results show the effectiveness of the TREND3 approach in revising the analysis algorithm and obtaining a new version that produced a fragmentation schema that reduced the cost (i.e., increased the performance) of the OO7 application in 38%.

6 Conclusions

Heuristic algorithms are used to address the intractability of the class fragmentation problem in the design of a distributed database, which is known to be an NP-hard problem. However, once defined, it is very difficult to improve them by manually defining and incorporating new heuristics from experimental results, while maintaining previous ones consistent.

This work presented a knowledge-based approach for automatically improving a heuristic DDODB algorithm through the use of theory revision. This approach is part of the framework that handles the class fragmentation problem of the design of distributed databases. The proposed framework integrates three modules: the DDODB heuristic module, the theory revision module (called TREND3) and the DDODB branch-and-bound module.

The focus of this work was to apply TREND3 to automatically revise the analysis algorithm of the heuristic module, according to experimental results on top of the OO7 benchmark presented as examples.

The revised algorithm led to an improvement of 38% in the overall system performance. This shows the effectiveness of our approach in finding a fragmentation schema with improved performance through the use of inductive logic programming.

Future work will include applying TREND3 to other applications, and the generation of examples to the TREND3 module using the branch-and-bound module to address the lack of performance results on top of distributed databases in the literature. Also, we intend to enhance the FORTE system to deal with negation as failure, extending the ideas already mentioned in previous works of our group [1,16].

Acknowledgements

The Brazilian authors would like to thank the Brazilian agencies CNPq and FAPERJ for providing financial support for this work, and Savio Leandro Aguiar for helping with the implementation of the experimental methodology in FORTE. Part of this work was done at the Computer Science Department of University of Wisconsin - Madison, USA, while the authors Fernanda Baião and Gerson Zaverucha were on leave from UFRJ.

References

1 Baião, F. (2001). A Methodology and Algorithms for the Design of Distributed Databases using Theory Revision. Doctoral Thesis, Computer Science Department – COPPE, Federal University of Rio de Janeiro, Brazil. Technical Report ES-565/02 (2002), COPPE/UFRJ.

2 Baião, F. & Mattoso, M. (1998). A Mixed Fragmentation Algorithm for Distributed Object Oriented Databases. Proc Int'l Conf Computing and Information (ICCI'98), Winnipeg, pp. 141-148. Also In: Special Issue of Journal of Computing and Information (JCI), 3(1), ISSN 1201-8511, pp. 141-148.

3 Baião, F., Mattoso, M., & Zaverucha, G. (1998a). Towards an Inductive Design of Distributed Object Oriented Databases. Proc Third IFCIS Conf on Cooperative Information Systems (CoopIS'98), IEEE CS Press, New York, USA, pp. 88-197.
4 Baião, F., Mattoso, M. and Zaverucha, G. (2001), "A Distribution Design Methodology for Object DBMS", submitted in Aug 2000; revised manuscript sent in Nov 2001 to *International Journal of Distributed and Parallel Databases,* Kluwer Academic Publishers
5 Baião, F., Mattoso, M., Zaverucha, G., (2002), A Framework for the Design of Distributed Databases, Workshop on Distributed Data & Structures (WDAS 2002), In: Proceedings in Informatics series, Carleton Scientific.
6 Bellatreche, L., Simonet, A., & Simonet, M. (1996). Vertical Fragmentation in Distributed Object Database Systems with Complex Attributes and Methods. Proc 7th Int'l Workshop Database and Expert Systems Applications (DEXA'96), IEEE Computer Society, Zurich, pp. 15-21.
7 Bellatreche, L., Karlapalem, K., & Simonet, A. (2000). Algorithms and Support for Horizontal Class Partitioning in Object-Oriented Databases. Int'l Journal of Distributed and Parallel Databases, 8(2), Kluwer Academic Publishers, pp. 155-179.
8 Blockeel, H., & De Raedt, L. (1996). Inductive Database Design. Proc Int'l Symposium on Methodologies for Intelligent Systems (ISMIS'96).
9 Blockeel, H., & De Raedt, L. (1998). IsIdd: an Interactive System for Inductive Database Design. Applied Artificial Intelligence, 12(5), pp. 385-420.
10 Brunk, C. (1996). An Investigation of Knowledge Intensive Approaches to Concept Learning and Theory Refinement. PhD Thesis, University of California, Irvine, USA.
11 Brunk, C., Pazzani, M. (1995). A Linguistically-Based Semantic Bias for Theory Revision. Proc 12th Int'l Conf of Machine Learning.
12 Carey, M., DeWitt, D., & Naughton, J. (1993). The OO7 Benchmark. Proc 1993 ACM SIGMOD 22(2), Washington DC, pp. 12-21.
13 Chen, Y., & Su, S. (1996). Implementation and Evaluation of Parallel Query Processing Algorithms and Data Partitioning Heuristics in Object Oriented Databases. Int'l Journal of Distributed and Parallel Databases, 4(2), Kluwer Academic Publishers, pp. 107-142.
14 Ezeife, C., & Barker, K. (1995). A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System. Int'l Journal of Distributed and Parallel Databases, 3(3), Kluwer Academic Publishers, pp. 247-272.
15 Ezeife, C., Barker, K. (1998). Distributed Object Based Design: Vertical Fragmentation of Classes. Int'l Journal of Distributed and Parallel Databases, 6(4), Kluwer Academic Publishers, pp. 317-350.
16 Fogel, L., Zaverucha, G. (1998). Normal programs and Multiple Predicate Learning. Proc 8th Int'l Conference on Inductive Logic Programming (ILP'98), Madison, July, LNAI 1446, Springer Verlag, pp. 175-184.
17 Fung, C., Karlapalem, K., Li, Q., (2002), Object-Oriented Systems: An Evaluation of Vertical Class Partitioning for Query Processing in Object-Oriented Databases, IEEE Transactions on Knowledge and Data Engineering, Sep/Oct, Vol. 14, No. 5
18 Getoor., L., Friedman, N., Koller, D., & Taskar, B. (2001). Probabilistic Models of Relational Structure. Proc Int'l Conf Machine Learning, Williamstown.
19 Getoor, L., Taskar, B., & Koller, D. (2001). Selectivity Estimation using Probabilistic Models, Proc 2001 ACM SIGMOD, Santa Barbara, CA.
20 Karlapalem, K., Navathe, S., & Morsi, M. (1994). Issues in Distribution Design of Object-Oriented Databases. In M. Özsu et al. (eds.), Distributed Object Management, Morgan Kaufmann Pub Inc., San Francisco, USA.

21 Kohavi, R (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In Proceedings of the IJCAI 1995, pp. 1137-1145.
22 Koppel, M., Feldman, R., & Segre, A. (1994). Bias-Driven Revision of Logical Domain Theories, Journal of Artificial Intelligence Research, 1, AI Access Foundation and Morgan Kaufmann, pp. 159-208.
23 Lima, F., & Mattoso, M. (1996). Performance Evaluation of Distribution in OODBMS: a Case Study with O2. Proc IX Int'l Conf. on Par & Dist Computing Systems (PDCS'96), ISCA-IEEE, Dijon, pp.720-726.
24 Maier, D. et al. (1994). Issues in Distributed Object Assembly. In M. Özsu et al. (eds.), Distributed Object Management, Morgan Kaufmann Publishers Inc., San Francisco, USA.
25 Meyer, L., & Mattoso, M. (1998). Parallel query processing in a shared-nothing object database server. Proc 3rd Int'l Meeting on Vector and Parallel Processing (VECPAR'98), Porto, Portugal, pp.1007-1020.
26 Mitchell, T. (1997). Machine Learning, McGraw-Hill Inc.
27 Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods, Journal of Logic Programming, 19(20), pp. 629-679.
28 Özsu, M., & Valduriez, P. (1999). Principles of Distributed Database Systems. New Jersey, Prentice-Hall, 2nd edition.
29 Richards, B., & Mooney, R. (1995). Refinement of First-Order Horn-Clause Domain Theories. Machine Learning, 19(2), pp. 95-131.
30 Ruberg, G., Baião, F., & Mattoso, M. (2002). "Estimating Costs of Path Expression Evaluation in Distributed Object Databases", In: Proceedings of the 13th International Conference on Database and Expert Systems Applications (DEXA 2002), LNCS v.2453, Springer Verlag, pp. 351-360.
31 Savonnet, M., Terrasse, M., & Yétongnon, K. (1998). Fragtique: A Methodology for Distributing Object Oriented Databases. Proc Int'l Conf Computing and Information (ICCI'98), Winnipeg, pp.149-156.
32 Tavares, F., Victor, A., & Mattoso, M. (2000). Parallel Processing Evaluation of Path Expressions. Proc XV Brazilian Symposium on Databases, SBC, João Pessoa, Brazil. pp. 49-63
33 Wogulis, J. (1994). An Approach to Repairing and Evaluating First-Order Theories Containing Multiple Concepts and Negation. Ph.D. Thesis, University of California, Irvine, USA.
34 Wrobel, S. (1996). First Order Theory Refinement. In L. De Raedt (ed.), Advances in Inductive Logic Programming, IOS Press, pp. 14-33