Bias Generator for View Discovery in Deductive Databases

NITTAYA KERDPRASOP and KITTISAK KERDPRASOP

School of Computer Engineering
Suranaree University of Technology
Nakorn Ratchasima, THAILAND
{nittaya, kerdpras}@ccs.sut.ac.th

ABSTRACT

We propose a framework of incorporating the deductive database system with the view-induction feature. These additional views benefit query answering. However, to make view induction practical for the real-world databases, biasing the discovery process is a necessity due to a huge search space. We thus develop the algorithm to generate a bias for the problem of view discovery. The bias generation is driven by a sequence of user's queries. The participation level of query predicates in the discovered view definitions is allowed to be adjusted accordingly.

KEY WORDS: intelligent databases, view discovery, bias generator

1. INTRODUCTION

A view is a virtual relation derived from base relations. Most database systems provide their users the mechanism to define views in order to isolate the data of interest and to speed up querying. Typically, a set of view definitions is created by the database designer. We propose to turn the conventional database into an intelligent system by adding the ability to learn from database contents the view definitions that are relevant to the user's queries.

The learning capability comes from the integration of Machine Learning technique into the database system. The technique of particular interest is Inductive Logic Programming (ILP)[1][2]. This consideration is due to the knowledge representation format and the learning power of the ILP systems. The ILP representation formalism conforms to the language of first-order logic used in deductive databases. The ILP technique also provides sufficient power of inducing views defined among relations, rather than within a relation as do other Machine Learning techniques, such as decision tree induction [3][4].

However, incorporating an ILP into the deductive database system as a learning unit requires some mechanism to automatically control the search space of view discovery. Otherwise, the discovery process could be so inefficient that the view-induction feature is less desirable. It is thus the purpose of this paper to present the algorithm to generate the appropriate bias for the view-discovery task. This bias is in the form of a language bias, which can control the size of the search space by putting restriction on the format of view definitions allowed in the search space. Our algorithm of generating the language bias is directed by the user's queries. Thus, only view definitions relevant to the user's interest are discovered.

The rest of this paper is organized as follows. The next section discusses the method to discover views, the learning system, and the technique to bias the view-discovery process. Section 3 presents our algorithms of generating bias to control the view discovery. In Section 4, we show the results of some experiments. The conclusion is presented in Section 5.

2. ILP AND VIEW DISCOVERY

2.1 The Framework of Discovering Views

Conventional deductive database systems consist of three finite sets: a set of facts or base relations (called extensional database –EDB), a set of deductive rules or views (called intensional database –IDB), and a set of integrity constraints (IC). All of these are in clausal form [5]. Once the database is created, its contents (i.e. , EDB, IDB, IC) are statically stored and remain unchanged until the next update. The proposed system (as shown in Figure 1) transforms the static deductive database system into the dynamic one by incorporating the ability to induce additional views. This induction process is triggered by the query and the induced views are eventually added into the database storage.

2.2 ILP Learner

There are a number of successful ILP systems, for example, MIS [6], FOIL[7], GOLEM[8], LINUS[9]. Most

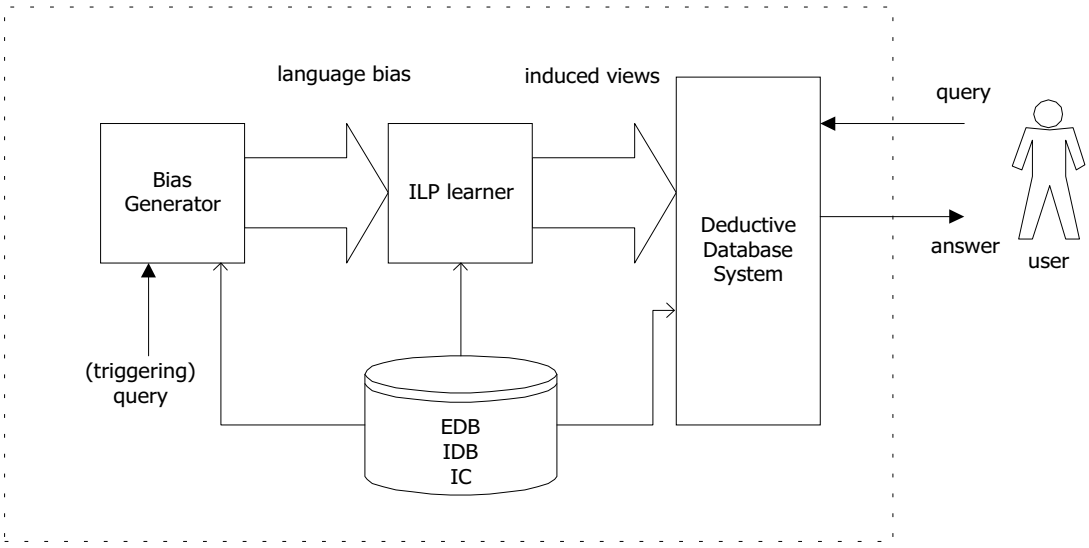

Figure 1: Deductive database system with view-induction feature

of these systems induce new definitions or concepts from positive and negative examples. But in the environment of deductive databases in which solely positive examples (represented as true facts and rules) exist, this kind of ILP setting seems to be less beneficial. Nevertheless, there have been some attempts [9][10] on applying this kind of ILP systems to learn interesting rules from deductive databases by giving negative examples either explicitly or automatically generated under the *closed world assumption*, i.e., all examples not stated in the database are negative examples. Apparently, this means of representing negative examples is not practical for the real-world databases. Fortunately, there is an alternative setting of learning from positive examples only, that is, nonmonotonic setting [11].

In the nonmonotonic setting each example is a Herbrand interpretation, not a clause as in the normal ILP setting. An example can be viewed as a set of facts that describe the specific properties of the example. Such an example is called a *model*. A model may contain multiple facts about multiple relations. For example, consider the single example consisting of the following facts *{animal (snoopy), animal(tweetie), can_walk(snoopy), can_fly (tweetie)}*, the concepts induced by a nonmonotonic ILP system are *{animal(X) ← can_walk(X), animal(X) ← can_fly(X) }*.

The induced concepts are no longer aimed at implying the positive examples, classifying positive examples from the negative ones, nor predicting the unseen examples as in the normal ILP. They indeed aim at representing a set of regularities that hold for the examples. These regularities are actually views or virtual relations in the context of databases. Thus, this nonmonotonic setting is well suited for the problem of view discovery in the databases. An ILP system that operates in this setting is CLAUDIEN (CLAUsal Discovery Engine) [12][13].

2.3 Bias in ILP

One problem that most ILP systems have in common is the huge size of the concept search space. The enormous size makes a complete search very inefficient. In order to make search more tractable, most ILP systems use *bias*. A bias is any mechanism that can constrain the search for the desired concepts in a learning system [14]. Bias can be divided into two categories: preference bias and language bias. Preference bias [15] constrains how the learner searches the search space and when to stop searching. Language bias constrains the concept space itself by imposing restriction on the form of concepts allowed in the search space. Thus, this kind of bias defines what to search.

Early ILP system embedded language bias in the system [16]. But in recent systems, language bias is implemented as a modifiable unit in which users can specify bias to better suit their specific tasks. The kind of bias used in the CLAUDIEN system is a language bias called DLAB (Declarative LAnguage Bias) [17]. The CLAUDIEN system employs the concept of template, called *dlab_template*, to specify the set of concepts allowed in the search space. For instance,

dlab_template('P(X,Y) ← 1-4: [Q(X),Q(Y),P(X,Y), P(Y,X)]').
dlab_variable('P', 1-1, [parent, father, mother]).
dlab_variable('Q', 1-1, [male, female]).

is the specification of concepts allowed in the concept space. The induced concepts are in the form of rules:

head ← *body*. The concepts should have one predicate in the head, which is either the predicate *parent*, *father*, or *mother*, and one to four predicates in the body. The body predicates are the combinations of the predicates *parent*, *father*, *mother*, *male*, and *female*. The dlab_variables *P* and *Q* are placeholders for real predicates. The variable *P* can be substituted by the predicates *parent*, *father*, or *mother*, and *Q* can be substituted by the predicates *male* or *female*.

Formally, DLAB is composed of a finite set of *dlab-templates* to which the concepts in the search space conform. The formal syntax and semantics of DLAB can be defined as follows.

Definition 2.1 DLAB syntax [13]

1. A DLAB grammar is a pair (*T*, V), where *T* is a set of DLAB templates , and *V* is a set of DLAB variables (*V* can be an empty set).

2. A DLAB template has the form
dlab_template(Template)
where *Template* is a string surrounded by single quote, and has the form $A \leftarrow B$ in which A and B are DLAB atoms.

3. A DLAB variable is of the form
dlab_variable (P_o , *Min - Max,* [P_1 , ... , P_n])
where P_o is the quoted atom, *Min* and *Max* are integers, $0 \le \text{Min} \le \text{Max} \le n$, and P_i is a predicate symbol or a function symbol.

4. A DLAB atom is either of the form
$P (t_1 , \dots , t_n)$ or *Min - Max : [ListAtoms]*
where *P* is a predicate symbol, t_i is a DLAB term, *Min* and *Max* are integers, $0 \le \text{Min} \le \text{Max} \le$ length of *[ListAtoms],* and *ListAtoms* is a list of DLAB atoms.

5. A DLAB term is either
(a) a variable, or
(b) of the form $f (t_1 , \dots , t_n)$
where *f* is a function symbol, t_i is a DLAB term, or
(c) of the form *Min-Max : [ListTerms]*
where *Min* and *Max* are integers, $0 \le \text{Min} \le \text{Max} \le$ length of [*ListTerms*], *ListTerms* is a list of DLAB terms. □

Definition 2.2 DLAB semantics [13]
Let G be a DLAB grammar, then dlab_gen(G) =
{dlab_dcg(A) ← dlab_dcg(B) | (A ← B) ∈ G }.
dlab_dcg(E) → [E], { E ≠ Min – Max: L }.
dlab_dcg(Min – Max: [ ]) → { Min ≤ 0 }, [ ].
dlab_dcg(Min – Max: [_|L] → dlab_dcg(Min – Max: L)
dlab_dcg (Min – Max: [E|L] → {Max > 0},
dlab_dcg(E),dlab_dcg((Min-1) – (Max-1): L). □

From the DLAB semantics, *dlab_gen* generates all concepts in the corresponding concept space, whereas *dlab_dcg* generates a list of all logical atoms. The head and the body of concepts generated by *dlab_gen(G)* are written as lists: [A_1, … , A_m] ← [B_1, … , B_n].

The following examples [13] demonstrate the generation of all valid concepts from each DLAB grammar. These examples illustrate the effect of the different declaration of "Min-Max" constructs.

Example 2.1: A DLAB grammar G_1 =
{ h ← 0-len: [a,b,c] }.
All valid concepts in the search space, according to the grammar G_1, are
{ [h] ← []
[h] ← [a]
[h] ← [b]
[h] ← [c]
[h] ← [a,b]
[h] ← [a,c]
[h] ← [b,c]
[h] ← [a,b,c] }.
The size of the search space = 8. □

Example 2.2: A DLAB grammar G_2 =
{ h ← 1-len: [a,b,c] }.
All valid concepts in the search space, according to the grammar G_2, are
{ [h] ← [a]
[h] ← [b]
[h] ← [c]
[h] ← [a,b]
[h] ← [a,c]
[h] ← [b,c]
[h] ← [a,b,c] }.
The size of the search space = 7. □

Example 2.3: A DLAB grammar G_3 =
{ h ← 1-1: [a,b,c] }.
All valid concepts in the search space, according to the grammar G_3, are
{ [h] ← [a]
[h] ← [b]
[h] ← [c] }.
The size of the search space = 3. □

Example 2.4: A DLAB grammar G_4 =
{ h ← len-len: [a,b,c] }.
The only valid concept in the search space, according to the grammar G_4, is
{ [h] ← [a,b,c] }.
The size of the search space = 1. □

The next example [11] illustrates how to specify a DLAB grammar in a more complex domain than the previous examples.

Example 2.5: Suppose the train-schedule database has the schema *train(Hour, Min, From, To)*. The following is the instances of a train-schedule traveling from one city to another city with the leaving time being specified as hour and minute.

Train(8,08 ,chicago, denver).
Train(8,10 ,st_Louis, washington_DC).
Train(9,08 ,chicago, denver).
Train(9,10 ,st_Louis, washington_DC).
Train(8,13 ,chicago, buffalo).
Train(8,43 ,chicago, buffalo).
Train(9,13 ,chicago, buffalo).
Train(9,43 ,chicago, buffalo).
Train(8,25 ,chicago, denver).
Train(9,25 ,chicago, denver).
Train(8,17 ,madison, buffalo).
Train(8,47 ,madison, buffalo).
Train(9,17 ,madison, buffalo).

The concept to be learned in this example is a functional dependency among the database instances. The DLAB grammar, which is the specification to induce a functional dependency from the train-schedule database, is as follow.

DLAB grammar (dlab_template, dlab_variable).
dlab_variable = ϕ.
dlab_template =
{ ' 1-1 : [Hour1 = Hour2, Min1 = Min2,
From1 = From2, To1 = To2]
$\leftarrow$
len-len : [train (Hour1, Min1, From1, To1),
train (Hour2, Min2, From2, To2),
0 - len : [Hour1 = Hour2, Min1 = Min2,
From1 = From2, To1 = To2]]'}

The functional dependency induced by CLAUDIEN, according to the given database and the language bias specification, is the constraint rule

From1 = From2 $\leftarrow$ train(Hour1, Min1, From1, To1),
train(Hour2, Min2, From2, To2),
Min1 = Min2, To1 = To2. □

The challenge of controlling the concept discovery via a bias is that the bias specification has to be expressive in order to capture a broad group of concepts. But the price is a large search space. On the contrary, if the bias is too restrictive, it might exclude all interesting concepts. Finding the appropriate form of bias specification for the problem of view discovery is thus the objective of our research.

3. BIAS GENERATOR

In this section we present some notation used in our bias generator algorithm. The algorithm configures the appropriate form of the bias grammar from the query and the input parameter specifying number of query atoms expected to appear in the discovered views.

A = a set of variables appeared as terms in the predicates.
B = a set of all predicates appeared in the EDB and IDB.
S = a set of considered queries. This set may contain either a query or a sequence of queries if inter-query relationship is expected.
hmin , *hmax* = the minimum (maximum) number of predicates p , p $\in$ B, appeared in the head of the derived views.
bmin , *bmax* = the minimum (maximum) number of predicates q , q $\in$ S, appeared in the body of the derived views.
head = a set of predicates allowed to appear (in disjunctive form) as the head of the derived rules. The number of predicates is in the range *hmin* to *hmax*.

$$\{ [p(a_1, a_2, \ldots, a_n)]_{hmin\text{-}hmax} \mid a_i \in A, p \in B\}$$

body = a set of predicates allowed to appear (in conjunction form) as the body of the derived rules.

$$\{ [q(a_1, a_2, \ldots, a_n)]_{bmin\text{-}bmax} \mid a_i \in A, q \in S\}$$

Definition 3.1 Participation weight (W)
W specifies the minimum number of query predicates expected to appear in the body of the derived views. □

Example 3.1 Given the query :
?- *grandfather (X, john), male(john)* .
and W = 50%. This implies that the discovered view definitions are expected to have at least one atom, *grandfather* or *male*, appeared as part of the body of views. The discovered view might be
grandson(Y,X) $\leftarrow$ grandfather(X,Y) □

Algorithm 3.1 BIAS GENERATOR
Input: a set S of query or sequence of queries
a participation weight W (in percent)
Output: the grammar G specifying bias for view discovery

Method:
(1) Initialization
hmin =1 , hmax =1
(2) Initialize set of variables and constants
$A' = A \cup \{ C \mid C$: constant(s) appeared in q_i , $q \in S\}$
(3) Compute the minimum length of view body
(maximum length can be directly computed from S)
bmin = $\lceil W * bmax / 100 \rceil$
(4) Define the specification for *head* and *body*
head = $\{[p(a_1,a_2,\ldots,a_n)]_{hmin\text{-}hmax} \mid a_i \in A', p \in B\}$
body = $\{[q(a_1,a_2,\ldots,a_n)]_{bmin\text{-}bmax} \mid a_i \in A', q \in S\}$
G : head $\leftarrow$ body

□

Algorithm 3.2 VIEW DISCOVERY
Input: a bias grammar G
the learning system CLAUDIEN [12]
a data set D
Output: discovered view definitions

Method:
(1) Transform the bias grammar G into the DLAB grammar [17] format.
(2) Run CLAUDIEN (controlled by the DLAB grammar) on D, the expected outcome is a set V of discovered views.
(3) Substitute each constant in V with a variable.

□

Running Example

Data set
EDB: female/1, male/1, mother/2, father/2, daughter/2, son/2, husband/2, wife/2, uncle/2, aunt/2, grandson/2

IDB: grandfather/2, grandmother/2, parent/2

Predicates in query sequence :
{grandfather(Y, 'Ben') , aunt(Y,X),male('Ben')} = S

A participation weight (in percent) : *W = 60 %*
Thus,
A' = {X,Y,'Ben'}
B ={mother, father, daughter, son, husband, wife, uncle, aunt, grandson, grandfather, grandmother, parent}
head = { [p(name, Y), p(Y,name)] $_{1\text{-}1}$ *| p ∈B, name = 'Ben'}*
body = { [grandfather(Y, name), aunt(Y, X), male (name)] $_{2\text{-}3}$ *| name ='Ben'}*

Transform the bias *head← body* into the DLAB grammar:

dlab_template('1-1:[p(name,Y),p(Y,name)]
←
2-len: [grandfather(Y, name), aunt(Y, X), male(name)] ').
dlab_variable (name, 1-1, ['Ben']).
dlab_variable (p, 1-1, [mother, father, daughter, son, husband, wife, uncle, aunt, grandson, grandfather, grandmother, parent]).

The rule discovered by CLAUDIEN is

grandson('Ben', A) :- grandfather(A, 'Ben'), male('Ben').

Transform this rule into view definition by substituting constant with variable:

grandson (X, A) :- grandfather (A, X) , male (X). □

4. EXPERIMENTS AND RESULTS

4.1 Data sets for Experiments

The experiments were designed to test our algorithm and to see the effect of a parameter W on the discovered views. The data sets used in the experiments are as follows.

EDB predicates :

female /1	*male /1*
mother /2	*father /2*
daughter /2	*son /2*
husband /2	*wife /2*
uncle /2	*aunt /2*

IDB :
grandfather(X,Y) :- father(X,Z),parent(Z,Y).
grandmother(X,Y) :- mother(X,Z),parent(Z,Y).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).

IC: none

Queries :
We test our algorithms on two kinds of queries: a single query and a sequence of queries.

Example of a single query :
?- grandfather (Y,'Ben'), male ('Ben'), aunt (Y,X).

Example of a sequence of queries :
?- grandfather (Y,'Mark').
?- mother (Y,'Mark'), male ('Mark').
?- aunt (Y, X).

4.2 Results

A number of queries in both categories are generated and tested on the algorithm. For each kind of queries, we adjust the participation weight (W) on the bias grammar to see its effect on the discovered view definitions. The average number of views discovered, number of valid views expected from all views discovered, learning time, and size of the search space are summarized and shown in Table 1.

The participation weight (W) on the bias reflects the number of atoms allowed in the view-definition search space. The more percentage of this weight is, the smaller the search space will be. This effect is clearly shown in the last column of Table 1. However, a small search space does not always imply a good learning result. If we put too much restriction on the bias grammar (W=100 %), the ILP system could not learn anything. The appropriate value is around 50-75%.

Table 1: View discovery triggered by queries at various weighting

Query Type	W (%)	#Views discovered	#Views applicable	CPU time (learning)	Size of search space
Single	30	1	0	0.516	238
	60	1	1	0.600	136
	100	0	0	0.116	34
Sequence	25	3	1	0.650	360
	50	5	3	1.030	264
	75	2	2	0.350	120
	100	0	0	0.080	24

5. CONCLUSION

We have proposed the framework of including ILP learner into a deductive database system in order to induce additional view definitions from the database contents. The discovered views can extend the deductive power of the IDB. We also present the algorithms to generate a bias in a format of DLAB grammar. This bias can efficiently constrain the discovering process by reducing the size of the search space into a tractable one. The bias is generated by considering the user queries and the number of query predicates expected to participate in the view definitions. This number can be adjusted accordingly. However, in order to identify the exact appropriate-range of this number (the W-value), more experimentation on a real-world database is needed.

REFERENCES

[1] S. Muggleton, Inductive logic programming, *Proceedings of the 1st International Workshop on Algorithmic Learning Theory (ALT 90)*, Ohmsha, Tokyo, 1990, 42-60.

[2] S. Muggleton, Inductive logic programming, *New Generation Computing, 8*(4), 1991, 295-318.

[3] J.R. Quinlan, Induction of decision trees, *Machine Learning*, *1*(1), 1986, 81-106.

[4] J.R. Quinlan, *C4.5: Programs for Machine Learning* (San Mateo, CA: Morgan Kaufmann, 1993).

[5] J. W. Lloyd, *Foundations of Logic Programming*, 2nd edition (Springer-Verlag, 1987).

[6] E.Y. Shapiro, *Algorithmic Program Debugging* (ACM Distinguished Dissertation, MIT Press, 1982).

[7] J. R. Quinlan, Learning logical definitions from relations, *Machine Learning*, *5*(3), 1990, 239-266.

[8] S. Muggleton and C. Feng, Efficient induction of logic programs, in S. Muggleton (Ed.) *Inductive Logic Programming* (Academic Press, 1992) 281-298.

[9] S. Džeroski and N. Lavrač, Inductive learning in deductive databases, *IEEE Transactions on Knowledge and Data Engineering*, 5(6), 1993, 939-949.

[10] P. Brockhausen and K. Morik, Direct acess of an ILP algorithm to a database management system, *Proceedings of the Mlnet Familiarization Workshop*, Bari, Italy, 1996, 95-110.

[11] P.A. Flach, A framework for inductive logic programming, in S. Muggleton (Ed.) *Inductive Logic Programming* (Academic Press, 1992) 193-211.

[12] L. Dehaspe, W. Van Laer, and L. De Raedt, *CLAUDIEN: The Clausal Discovery Engine User's Guide 3.0*, Technical Report CW 239, Department of Computer Science, Katholieke Universiteit Leuven, 1996.

[13] L. De Raedt and L. Dehaspe, Clausal discovery, *Machine Learning*, 26(2/3), 1997, 99-146.

[14] P. Utgoff and T.M. Mitchell, Acquisition of appropriate bias for inductive concept learning, *Proceedings of the National Conference on Artificial Intelligence* (AAAI 82), Pittsburgh, 1982, 414-417.

[15] R.S. Michalski, A theory and methodology of inductive learning, in R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.) *Machine Learning: An Artificial Intelligence Approach*, Volume 1 (Palo Alto, CA: Morgan Kaufmann, 1983) 83-134.

[16] N. Lavrač and S. Džeroski, *Inductive Logic Programming: Technique and Application* (New York: Ellis Horwood, 1994).

[17] L. Dehaspe and L. De Raedt, DLAB: A declarative language bias formalism, *Proceedings of the International Symposium on Methodologies forIntelligent Systems* (ISMIS96), 1996, 613-622.