Abstract—The testing of database and application system is an  
important issue in software testing and it is concerned by academic  
research recently. This paper proposed an approach that combined  
white-box testing with black-box testing for database stored  
procedures testing. This method selected a set of testing path by the  
technique of path coverage and generated test cases by the technique  
of program slices. The experiments proved that the approach can be  
used for test cases generation efficiently and implemented the path  
coverage testing. This approach provided a reference for the testing  
of program which is coded by the forth-generation programming  
language

I NTRODUCTION  
In order to assure the quality of software, testers need to  
test not only the software system, but also the database in  
Database-driven Applications. Since the quality of databases  
directly influences that of software, some functional defects of  
application were caused by incorrect database design and  
implementation. Therefore it is necessary to make sure that the  
database implementation could be correct before application  
testing. For a large and complex database application system, it  
is difficult for programmers not to make mistakes while  
designing and implementing. However, the traditional  
approach of software testing always ignores or simplifies the  
database testing, so it might induce potential crisis for software.  
Even though testing of database and its applications is different  
from general software testing, little work has been done on  
research of its approach. With the extensive Web Applications,  
testing of database will be a hot academic issue of software  
testing.  
The so-called database testing is required to verify and  
validate the internal structure and functions of databases,  
including testing for database internal functions, database  
schema, data quality etc.. Database internal function testing  
checks database triggers, database stored procedures and stored  
functions, database class and so on. Database schema testing  
checks database structure, including databases, tables, views,  
indexes, constraints, rules, relations (reference integrity). Data  
quality testing checks the integrity and consistency of data.  
Database testing has its own characters. First, database  
programming language SQL is a declarative programming  
language, so the execution procedure of program is concealed  
in code level and difficult to fulfill path coverage testing.  
Besides, the input and output of database testing have multiple  
format: data, data set, database states. Same input in different  
conditions may generate different testing results. In addition,  
the strategy of database system adopts Three-Valued Logic  
instead of Boolean Logic, and the analysis of testing results  
requires more states to determine.  
Database stored procedure is an important component of  
database, and test for it is also the emphasis of database  
internal function testing. Database stored procedure testing  
mainly adopts black-box testing so far. However, the database  
stored procedure is a code in unit level and manipulates  
database and is related to input data and database states. If  
database stored procedure is tested with black-box testing, it is  
hard to cover all paths and statements of execution. Besides,

database stored procedure is stored stably like data in database  
and it will spend much more time to design test cases.  
Therefore, to apply automated testing technique as an efficient  
assistant for human testing seems significant and will improve  
the efficiency of testing.  
This paper focused on the testing of database stored  
procedure and proposed an approach that combined the while-  
box testing with black-box testing. It adopted technique of path  
coverage to generate test paths and generated test cases  
automatically by technique of program slice. With testing  
technique of Z-Path Coverage, it generated test paths for  
database stored procedure by graphic matrix and depth-first  
search algorithm. Then it constructed a constraint system for  
test data with methods of extracting program slices and  
replacing predicates. Moreover, it found solutions and gained  
test data that meets the demand of constraints with Genetic  
Algorithm. Finally, it generated formatted test paths and test  
cases for database stored procedure automatically, including  
input parameters and data in database. The experiments proved  
that we could generate test cases efficiently with this approach and implemented path coverage testing

III. G ENERATING T ESTING P ATH OF D ATABASE STORED  
P ROCEDURE B ASED ON Z-PATH COVERAGE  
Z-path coverage testing is a variation of general path testing,  
which is based on program Control Flow Graph (CFG). It  
simplified the loop mechanism and abandoned minor factors of  
path coverage. The set of extracted basic executive paths is  
called Z-path.

A. Generating Control Flow Graph of Database Stored  
Procedure  
Same as other programs, database stored procedure  
includes sequential, conditional branch and loop structure. In  
conditional branch structure, when a condition is compound  
condition, it is necessary to rewrite the conditions as nested for  
generating path conveniently. For example, the CFG of  
statement " if a and b then procedure x; else procedure y;  
endif;" is shown in Fig. 1(a), and the CFG of statement " if a or  
b then procedure x; else procedure y; endif;" is shown in Fig.  
1(b).

In loop, we only take entrance and non-entrance situations  
into consideration for "WHILE" loop. So that we can regard  
loop statement as conditional branch statement and process a  
loop body as a "black-box". For instance, the statement "while  
a begin procedure x; end procedure y; " is shown in Fig. 2. In  
this way, it is possible that the branch in the loop has not been  
tested. If necessary, tester can adopt piling method to undertake  
an individual test for the loop body.

B. Constructing Graph Matrix for the Storage of Control  
Flow Graph  
Graph Matrix is an assistant tool to produce paths  
automatically in general path testing, which is used to describe  
CFG. The size of Graph Matrix equals the number of nodes in  
CFG and the values in the matrix represent connection weights  
among edges using two symbols, usually "0" or "1". "0" means  
non-connection while "1" represents connection. The generated  
Graph Matrix of database stored procedure is

N i, i=0,1,...,n-1 ҏ represents each node in database stored  
procedure andҏ ai,j, i, j=0,1,...,n-1 represents connection weight  
between two nodes.  
534

C. Computing of Loop Complexity in CFG  
We need to figure out the loop complexity in order to get  
the number of independent paths in sets of Z-path, namely  
minimal number of test cases which the test needs. Independent  
path is a path which can’t be represented by any linear  
combinations of other independent paths. Three methods are  
usually used for computing of loop complexity:  
1) V(G)=E  
−n+2 , E is the number of edges in CFG and n  
is the number of nodes.  
2) V(G)=F+1, F is the number of nodes containing  
rhombic-shape judgment in CFG.  
3) V(G)=I, I is the number of areas segmented by control  
flow lines in CFG.  
Besides, it is also feasible to compute loop complexity  
based on Graph Matrix. Firstly, we count the sum of all the  
connection weights in every row together:  
n-1  
i ij  
j=0  
v = a , 0,1, , 1i n= −¦   
(3−2)  
Then, we can obtain its loop complexity:  
1  
0  
( ) 1 ( 1)  
n  
i  
i  
V G v  
−  
=  
= + −¦ . ˄3−3˅  
D. Generating the Test Paths for Database Stored  
Procedures  
Since the Graph Matrix of database stored procedure is an  
adjacent matrix of directed graph, we can generate test paths  
for database stored procedure by using depth-first search  
algorithm. The advantages of applying depth-first search are its  
convenience to implement, avoidance of overlapped paths and  
realization of whole path coverage.  
IV. G ENERATION OF T EST D ATA FOR D ATABASE S TORED  
P ROCEDURE B ASED ON PROGRAM S LICE  
As we know, they are related between the running results of  
database stored procedure and the database states and the  
execution of database stored procedure is difficult to be tracked.  
Thus, while generating test cases by dynamic methods, we may  
gain the test data which have no value to use due to neglecting  
the database states. In this paper, we try to build combine static  
method with trial method. The static method is used to build  
constraint system and the trial method is used to solve the  
constraint system.  
A. Abstracting Related Program Slice From Test Paths  
We abstract all the statements from every path in set of path  
by using technique of program slice, including the statements  
of condition, assignment, SQL etc.. Taking the execution path  
as interested point, we extract all paths in database stored  
procedure to form program slice:  
( ), ,i i i ip w w W p P∈ ∈ ѽ (4-1)  
ҏ  
W is a set of path, Wi ҏ is ith path in the set, P is the  
correspondent set of slice for W. Then, let us focus our

interested point on the programs which are related to input  
parameters and preinstall data in database. Extractions of  
program slice are as follows:  
( , ), ,j i i is BP w s p w W∈ ∈ ѽ (4-2)  
Here, BPj is a set of program slice relating to test data,  
mainly including: branch and loop statements determining the  
choices of path, SQL statements such as Select, Update, Insert,  
Delete and assignment statements etc. .  
B. Extracting and Substituting of Predicates in Program  
Slice  
Here we extract and replace predicates by static approach.  
Conditional predicates in program slice include conditional  
expressions in branch statements and judgment conditions in  
loop statements. Constraint conditions after "WHERE"  
conditional expression constrain preinstall data of database in  
the form of r: E 1 OP E2 . If the expressions of E 1, E2 ҏ have not  
been changed in the execution of predicate r: E 1 OP E 2, we can  
put all predicates together to build up a constraint system.  
Finally, test data can be generated by the simultaneous solution  
of this constraint system.  
Extraction of program slice may also include assignment  
statements. During the execution of program, if predicate r: E 1  
OP E2 is executed after an assignment statement, expression E1 ,  
E 2 in this predicate are very likely to be changed because of the  
influence of assignment statements. Since the predicate is  
extracted by static methods, we can hardly know the changes  
of predicate expressions during the execution of database  
stored procedure. Therefore, it is advisable to apply technique  
of back-to-front predicate substitution which searches the  
program slice from back-to-front. When meeting a predicate,  
we record it; when meeting an assignment sentence, we update  
all the recorded predicates in order to obtain the changes of  
predicate.  
C. Establishing Constraint System of Test Data Generation  
The so-called constraint system is a system which joins all  
predicates of constraining test data together. It is denoted as:  
1 1  
2 2  
1 1 2  
2 1 2  
1 2  
:  
:  
:  
: n n  
r r  
r r  
r r  
n  
r E OP E  
r E OP E  
R  
r E OP E  
°  
°  
®  
°  
°  
̄  
  
(4-3)  
Whether E 1 and E 2 are the same or not, it will not affect the  
solution of constraining set.  
In database stored procedure, except for establishing a  
constraint system of condition predicates in order to guarantee  
all paths achievable, it is necessary to build up a constraint  
system for every SQL statement block. The former one is path-  
constraint system while the latter one is a constraint system of  
preinstall data in database. First of all, we build up a constraint  
system for test paths, extract condition predicates after  
substitution and join a series of inequalities together, namely  
constraint system of test-path. Then, it is necessary to establish  
a constraint system of database preinstall data. Two kinds of

situations need to be considered. One situation is that the  
preinstall data are not related to paths, so we just join  
"WHERE" conditional predicates together according to serial  
numbers of SQL statements. Another situation is that preinstall  
data are related to paths, and then we put the constraint system  
of database state into path constraint system and solve it.  
D. Solving the Constraint System of Generation Test Data  
The general approaches to solve the constraint system are  
Linear Programming and Non-linear Programming. Our goal  
of solving this system is to generate test data meeting the  
constraints. One thing what we should pay attention to is that  
we only need one feasible solution instead of a set of solution.  
All solutions or the maximum and minimum solutions among  
them are not necessary here. In the database stored procedure,  
it is difficult to ensure all constraint conditions are strictly  
greater or less. Even if they are, it is still inevitable to deal with  
strictly greater or less situations in the choices of supplying  
mutual paths. Besides, for nonlinear constraint systems, it is  
essential to change the nonlinear constraint system into an  
approximate linear one by solving partial derivatives, even  
though such a change may cause the nonlinear constraint  
algorithm incomplete. Genetic algorithm, on the other hand,  
which is free from the limitation of search space, needs no  
assistant information and has special advantage of processing  
problems relevant to large space, multi-peaks, nonlinear and  
global optimization efficiently. So, this paper applied genetic  
algorithm to seek for the solution of constraint system. Detailed  
steps are as follow:  
1) Code  
Number of parameters is determined by constraint system  
while the range of parameters is decided by testers. We  
represent code in binary code and the formula is:  
max min  
min  
( )  
2 1m  
b a a  
a a −  
= + − (4−4)  
The range of parameter  ∈ [ҏmax,  min ], m denotes m-  
digits binary number, b represents parameter  [19]. As for  
multi-parameters, we change code string into character string  
and regard it as the target of genetic algorithm.  
2) Generation of initial population  
We generate to form an initial population randomly.  
3) Selection of fitness function  
We transform every predicate r: E1 OP E2 in constraint  
system into the form of r’:f OP’ 0, shown in TABLE I: F=¦ fi.  
denotes fitness function. The lower parameter of adaptive  
degree is, the bigger value of fitness function becomes.

4) Selection of parent population  
By using methods of crossover and mutation, we apply  
ratio and fitness probability of every individual to determine  
the genetic possibility of its offspring. The individual with high  
probability may be chosen for several times, so its genetic  
factor will expand in the group. Individual i fitness degree Fi  
ҏ  
and selection probability Pi ҏ should meet the following equation:  
1  
i  
i M  
i  
i  
F  
P  
F  
=  
=  
¦ (4-5)  
5) Selection of hybrid and mutation operators  
We select multiple-point hybrid as hybrid operator and  
binary mutation as mutation operator.  
6) Conditions of iterative termination  
ƻ1 Fixed number of generations reached (If so, it is  
common to have no-solution and it indicates that the path is not  
achievable or a conflict exists in conditions of WHERE  
statements) Individual maximum fitness degree in the group  
exceeds preinstall value. Every individual in the group satisfies  
all constraints.  
ƻ2 A solution is found that satisfies minimum criteria.  
ƻ3 Allocated budget (computation time/money) reached.  
ĺ The highest ranking solution's fitness is reaching or has  
reached a plateau such that successive iterations no longer  
produce better results.  
Ļ Manual inspection  
ļ Combinations of the above  
E. Generation of test data for database stored procedure  
In a table of database, it may contain n fields while the  
constraint system of preinstall data generated by SQL block  
may only include n-m parameters. It is possible to find the  
solutions of those n-m parameters with the above algorithm.  
But the other m parameters have no definite value so that we  
can’t insert rows into table correctly. We call these data the  
incomplete preinstall values, which have not been generated  
according to fields in table completely. So it is necessary to  
retrieve the fields in table of database and randomize data  
which have not been generated and output the generated results  
finally.  
V. E XPERIMENTS FOR V ERIFICATION OF AUTOMATED  
GENERATION TEST CASES IN DATABASE STORED PROCEDURE  
To verify the approach of automated test cases generation,  
we have designed a system prototype in Linux environment.  
This system is implemented with Java under B/S structure,  
Tomcat web server, MySQLV5.0 as background database and  
Eclipse as development platform.

Shown in Fig. 3, this system has five functional modules  
including path-generation module, automated data-generation  
module, results output module, human interaction module and  
system log record module. The input of this system consists of  
information of database stored procedure which needs to be  
tested and the expected number of test cases which need to be  
generated. The output includes formatted test cases files,  
database states and log record of system execution.  
In the MySQL stored procedure testing, after generating  
automated test cases for 80% of simply stored procedures( at  
most 20 lines and not including complex logic) and running  
testing by those generated test cases, we found out that 70% of  
test cases are executable and general path coverage testing has  
been completed. Compared with manual generation of test  
cases, this method improves efficiency and increases the  
quantity of test coverage.  
Our experiments showed that some specific problem  
solutions in database stored procedure still need to be improved.  
For example, our approach fails to cover the whole paths in  
loop body. In this case, we need to add additional test data by  
manual work. If aggregation functions or datetime functions  
exist in database stored procedure, the quantity of preinstall  
data in database possibly can not meet the demand of testing so  
that more copied data again needs to be added by manual work.  
Moreover, there is a problem of compatibility to other testing  
tools, such as the integration with test data generated by  
existing testing tools etc.  
VI. CONCLUSIONS  
In terms of database stored procedures testing, we  
improved software testing methods based on procedure and  
made it adapt to the software testing based on declaration. We  
proposed an approach which generates test paths and data  
automatically for database stored procedures testing. And at the  
same time, we designed and implemented the automated  
generation system. Since the research in this paper was  
exploratory, there are still future works on automated test cases  
generation in database stored procedure testing, including  
automated test cases generation in loop bodies, test cases

generation related to functions of system and user, dynamic  
generation of test cases and so on