Toward Unification of Taxonomy Databases in a Distributed Computer Environment

Hajime Kitakami*, Yoshio Tateno and Takashi Gojobori****

* Hiroshima City University
151-5 Ozuka, Numata-Chou, Asa-Minami-Ku
Hiroshima-Shi 731-31, Japan
E-mail:kitakami@its.hiroshima-cu.ac.jp

** National Institute of Genetics
1111 Yata, Mishima-Shi
Shizuoka-Ken 411, Japan
E-mail:ytateno@ddbj.nig.ac.jp, tgojobor@ddbj.nig.ac.jp

Abstract

All the *taxonomy databases* constructed with the DNA databases of the international DNA data banks are powerful electronic dictionaries which aid in biological research by computer. The taxonomy databases are, however not consistently unified with a relational format. If we can achieve *consistent unification* of the taxonomy databases, it will be useful in comparing many research results, and investigating future research directions from existent research results. In particular, it will be useful in comparing relationships between phylogenetic trees inferred from molecular data and those constructed from morphological data. The goal of the present study is to unify the existent taxonomy databases and eliminate inconsistencies (errors) that are present in them. Inconsistencies occur particularly in the restructuring of the existent taxonomy databases, since classification rules for constructing the taxonomy have rapidly changed with biological advancements. A repair system is needed to remove inconsistencies in each data bank and mismatches among data banks. This paper describes a new methodology for removing both *inconsistencies* and *mismatches* from the databases on a distributed computer environment. The methodology is implemented in a relational database management system, SYBASE.

1. Introduction

The increasing number of genome projects in the world have led to the further development of biological sciences and an advancement of the technology involved. In particular, the comparative and evolutionary studies of different genomes have become very important. This has promoted a restructuring of taxonomies constructed from various biological data. So far, such restructuring is carried out by a specialist who is interested in specific species or genes. Taxonomy data are usually very complicated because of the nature of its tree structure. It is thus important to make a taxonomy database which can be maintained in and searched through a computer system. The taxonomy database is a kind of electronic dictionary for searching automatically for any taxonomy information on the computer system. If the electronic dictionary can be used freely by biologists, it will contribute not only to genome research but also to other biological sciences. Above all, the evolutionary studies would gain tremendous benefits from a taxonomy database.

A typical taxonomy database is constructed at each of the international DNA data banks (Kahn & Cameron 1990, Higgins & Fuchs et al. 1992, Burks & Cinkosky & Gilna & Fickett & Gifford et al. 1990-1992, Roberts 1993, Kitakami & Ikeo & Ugawa & Tateno & Gojobori et al. 1994) which are EMBL (European Molecular Biology Laboratories) Data Library, GenBank-NCBI (National Center for Biotechnology Information)/GSDB-LANL (Genome Sequence Database constructed at Los Alamos National Laboratory) and DDBJ-NIG (DNA Data bank of Japan constructed at the National Institute of Genetics). These data banks shown in Figure 1 have been collaborating in many areas through mutual exchanges of data over the international computer network.

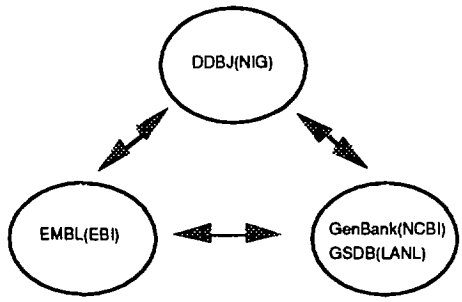

Figure 1. International DNA data banks

The aim of this paper is to present a method for producing a consistent taxonomic database that can unify (or integrate) the taxonomy databases supplied by these data banks into a single unified taxonomy database. Before unification, it is necessary for each taxonomy database to be consistent within itself and any pair of taxonomy databases should also be consistent with each other. We propose a new methodology for finding

inconsistencies and constructing a consistent taxonomy database as a step toward the unified taxonomy database in a distributed computer environment. The methodology is implemented in the relational database management system, SYBASE (Sybase Inc. 1989).

In summary, section 2 describes a unification processing of taxonomy databases. In section 3, we describes the basic concept of a *recursive join* which is a search engine of the implemented system. In section 4, we describe a new *neighborhood search* subsystem which is needed for clarification of error structures in the taxonomy tree. In section 5, we describe the definition of *integrity constraints* for detecting errors. In section 6, we describe the system configuration for implementing the new methodology. In section 7, we discuss related work for comparing the system with another system in constructing the unified taxonomy database. The final section is a summary of our research results.

2. Unification Processing of Databases

A unified database is constructed by unifying (or integrating) the existent taxonomy databases. If we can achieve the construction, we can obtain the largest taxonomy database in the international DNA data banks. It will be useful in comparing many research results, and investigating future research directions from existent research results. In particular, it will be useful in comparing relationships between phylogenetic trees inferred from molecular data and those constructed from morphological data.

We assume that integrity constraints, ICs, will be defined to make consistency in the existent databases, ΔDB. ICs consist of both *structural integrity constraint*, SICs, and *cooperative integrity constraints*, CICs. Both will be discussed in the section 5. Our unification processing is carried out by adding consistently other databases, OtherDBs, to the core database, CoreDB, which is one of the existent databases. The processing is summarized as shown in (U1) through (U4) below.

(U1) ICs = { SICs, CICs }, ΔDB = CoreDB $\cup$ OtherDBs,
OtherDBs = { DB_1, DB_2, ..., DB_n }

(U2) Build_consistency(DB, SCIs) for $\forall$ DB $\in \Delta$DB

(U3) Build_consistency((CoreDB $\cup$ DB_i), CICs)
for $\forall$ $DB_i \in$ OtherDBs

(U4) NewDB = CoreDB
while(i > n) do
NewDB = NewDB $\cup$ (DB_i - NewDB), $DB_i \in$ OtherDBs
end_while

After defining the core database in (U1), each of existent databases is consistently revised with the structural integrity constraints, SICs, in (U2). We currently define the DDBJ-taxonomy database as the core database and revise only the DDBJ-taxonomy database in (U2). The core database is consistently revised with one of the other databases and the cooperative integrity constraints, CICs in (U3). We mainly use either GSDB or GenBank as one of other databases, OtherDBs. Finally, each of other consistent databases is added to the core database in (U4). In this paper, we are concerned with *two building processes* which are (U2) and (U3) in the unification processing.

3. Recursive Join

A taxonomy has a tree structure in which each taxonomic unit is connected with some immediately subordinate unit. Let us denote the former and latter units by "parent" node and "child" nodes, respectively. The relationship between a child node "X" and a parent node "Y" can be represented by a binary relation "(X,Y)", where "X" and "Y" are defined by the same domain "D".

Let R={ (a,b) | a$\in$D, b$\in$D } and S={ (b,c) | b$\in$D, c$\in$D } be two binary relations. The join of "R" and "S", denoted as R$\triangle$S, is defined as { (a,c) | $\exists$b ((a,b) $\in$R, (b,c)$\in$S) }.

Let R_0 ={ (a_0,b_0) | $a_0 \in$ D, $b_0 \in$ D }, $R_1 = R_0$, $R_i = R_{i-1} \Delta R$. The recursive join of "R" with an initial relation "R_0", denoted as "[R]", is defined as { $\cup_{i>0} R_i$ }. Let us consider R={ (a,b) | a$\in$D, b$\in$D }, where "a" is a child node of "b". The recursive join "R@R_0" with a initial relation R_0 computes all lineages from each node stored in the initial relation "R_0" to the root node. We call this "lineage processing". The other hand is the case of R={ (a,b) | a$\in$D, b$\in$D }, where "a" is the parent node of "b". In this case, the recursive join "R@R_0" with a initial relation R_0 computes all progeny from each node stored in the initial relation "R_0" to the root node. We call it "posterity processing".

The recursive join (Bancilhon & Ramakrishnan 1986) is represented by the following algorithm:

```
TempR= R0 ; R'= R0 ;
while ( R' ≠ Ø) do
    R' = R'Δ R ;
    TempR = TempR ∪ R' ;
end_while
output TempR,
```

where TempR is a temporary relation (or table) with the same schema as the concerned relation, R.

In this paper, the previous binary relation is implemented as follows:

```
taxonomy( tx_id, tx_tl_id, tx_tx_idp, tx_nodename, _)

taxlevel( tl_id, tl_levname, _)
```

Each node of the taxonomy tree is stored in the "taxonomy" relation and the level or ranking of the node in the taxonomy tree is stored in the "taxlevel" relation. "tx_id" of the "taxonomy" relation specifies the identifier for each node. "tx_tl_id" specifies the pointer

to "tl_id" of the "taxlevel" relation. "tx_tx_idp" specifies the pointer to a parent node in the "taxonomy" relation. "tx_nodename" specifies the name of the node. "tl_id" of the "taxlevel" relation specifies the identifier for each level name.

4. Neighborhood Search

In constructing a taxonomy database, it is very important to make it consistent. However, we do not have definitions clear enough to create inconsistency checking in the database design phase, because the biological information change due to rapid advancements in biology. After the design phase, we can incrementally clarify definitions related to the inconsistency checking in the database administration phase. The inconsistent (error) nodes stored in the "taxonomy" relation can be incrementally detected in the administration phase. We can repair these inconsistencies by using the neighborhood search functions which visualize neighborhood nodes. The functions instruct us to conduct correct revisions in perspective. Three of the functions including the recursive join algorithm are as follows:

(1) lineage search function

Let us store R_0 with a given node and consider R={ (a,b) | a∈D, b∈D }, where "a" is a child node of "b". This function can be implemented by "R@R_0" and searches for a path from a given node to the root node, which does not have any parent node in the tree structure. The path is a set of nodes found by searching in the "taxonomy" table. Appendix-1 shows an example of the program that was implemented in the control flow language of SYBASE called the *"stored procedure"*. The single join part of the processing is implemented by the following program:

```
select x.acc_num, y.tx_id, y.tx_tx_idp,
       tl_levname, tx_nodename
from   tempdb..work_table x,
       taxonomy y, taxlevel
where  tx_tl_id=tl_id
and    y.tx_id=x.tx_idp,
```

where "termpdb..work_table" means the previous temporary relation, TempR. Our approach includes the object-oriented database concept (Andleigh & Gretzinger 1992), since the stored procedure is a kind of method for hiding the table structures of the "taxonomy" table.

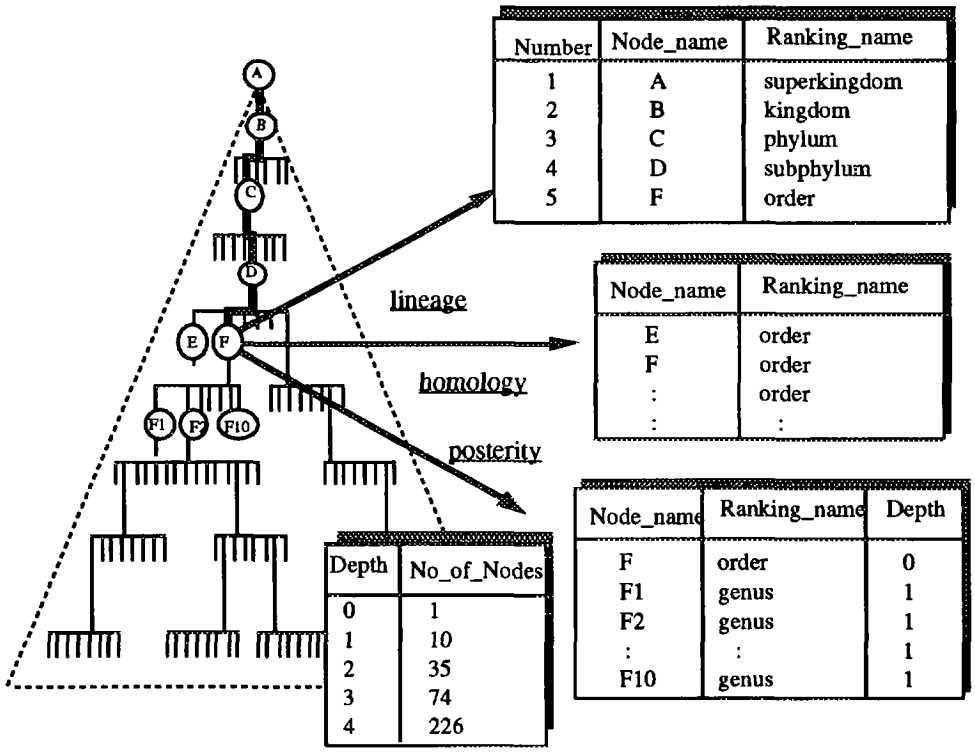

Figure 2. Neighborhood search

(2) posterity search function

We would like to make this function search for all paths from a given node to its leaf nodes, which do not have any child node in the tree structure. If the join of "R' " and "R ", denoted as "R'△R" in the previous recursive join algorithm, is defined as { (a,c) | ∃b ((b,a)∈ R', (c,b)∈ R) }, this function can be implemented by "R@R0". The number of the nodes found by the posterity processing is generally so large that the system can not visualize in any single window system at the same time. Thus we make this function visualize only in a given node and its child node. If we repeatedly use this function, we can find all paths from a given node to the leaf nodes in the tree structure.

In addition, we implemented an option to compute statistics which show the number of nodes for each level, in searching all progeny.

(3) homology search function

This function searches for nodes of the same level as a given node in the tree structure. If we search for a parent node of a given node, we can look for all children nodes of the parent node in the tree structure. This search visualizes homologous nodes for a given node. The homologous nodes include the given node and are in the same classification .

Figure 2 shows a representation of the neighborhood search. If we use the three functions, we can see a partial tree structure which spans both up-and-down and left-and-right nodes for a given node. We can define the neighborhood area inferred from any node using the neighborhood search functions. The neighborhood area can cover several nodes visualized by use of the functions. If we change the given node to an other node in the area, we can find the new area defined by the change. If we repeatedly apply the neighborhood search, we can move to any area in the original tree structure.

In addition, we also have a search function which looks for the accession numbers of the data entry to be connected with the taxonomy databases.

5. Integrity Constraints

It is important to maintain high quality in a database, including both facts (data) and inference rules. This enables us to construct a database system to carry out consistency checking using integrity constraints as defined by experts, with automatic revision of the database to eliminate inconsistencies (Kitakami & Kunifuji et al. 1994).

The set of facts should be manually revised by biological experts, since the taxonomy database only has a set of facts without inference rules. The existent taxonomy database of DDBJ includes both syntax and semantic errors. The syntax errors can be easily found, if we can clarify structural inconsistencies which destroy the tree structure. The semantic errors can be found, if we can clarify the correct and invariable contents for the database with the advancement of biology. The difficulty lies in that a single data bank can not define the correct contents of the taxonomy database by itself and there are no perfect taxonomy dictionaries which reflect world-wide advancements in biology.

We are involved in an international collaboration aimed at defining the correct contents of the taxonomy database. This provides a chance to construct a consistent taxonomy database using the system.

Let us define two kinds of integrity constraints for detecting errors and inconsistencies. One is named the *structural integrity constraint*, which searches for any abnormal partial trees and nodes. The other is the *cooperative integrity constraint*, which searches for inconsistent nodes among the taxonomy databases of the international data banks. We can logically define the two kinds of integrity constraints as follows:

(1) Structural Integrity Constraints

The integrity constraints can detect (a) *undefined node names*, (b) *data duplication*, (c) *invalid pointers to parent nodes*, so that they can detect abnormal partial trees and nodes in the existent taxonomy database.

structural_inconsistent1 <-
name(node) = ∅,
node ∈ DBi.
structural_inconsistent2 <-
name(nodep) ≠ ∅,
name(nodep) = name(nodeq),
nodep ∈ DBi, nodeq ∈ {DBi - nodep }.
structural_inconsistent3 <-
parent_id(nodep) ≠ ∅,
parent_id(nodep) ≠ id(nodeq),
nodep ∈ DBi, nodeq ∈ DBi.
structural_inconsistent4 <-
name(nodep) ≠ "root",
parent_id(nodep) = ∅,
nodep ∈ DBi.

(2) Cooperative Integrity Constraints

There are sometimes (a) *incorrect spellings*, (b) *reverse turns for up-and-down relationships between parent and child nodes*, and (c) *unfinished restructuring (for example, splitting and merging)* in the tree structure. We can detect these errors through comparison with the taxonomy databases of the international DNA data banks over international computer networks.

cooperative_inconsistent1 <-
name(nodei) = name(nodej),
lineage(nodei) ≠ lineage(nodej),
nodei ∈ DBi, nodej ∈ DBj,
DBi ≠ DBj, DBi ∈ *ΔB* , DBj ∈ *ΔB* .
cooperative_inconsistent2 <-
name(nodei) = name(nodej),
level(nodei) ≠ level(nodej),
nodei ∈ DBi, nodej ∈ DBj,

$$DB_i \neq DB_j,\ DB_i \in \Delta B,\ DB_j \in \Delta B.,$$

where "lineage(a)" is the lineage, "name(a)" is the name, "level(a)" is the level (or ranking), "parent_id(a)" is the parent identifier, and "id(a)" is the identifier of the node, "a". If one of the databases is consistent with the structural integrity constraints, it is called a self-consistent database. The cooperative integrity constraint is applied only to the self-consistent database in order to check more semantic errors.

We show some examples of *SQL expressions* to implement these logical definitions of the two kinds of integrity constraints. Namely, if one of the integrity constraints does not have an empty solution, its integrity constraint is true and is inconsistent with the database. If one of them has an empty solution, the integrity constraint is false and is consistent with the database.

Undefined Node Name (the first structural integrity constraint)

We should avoid node names represented by blanks or NULL. This situation can be shown using the following query:

```
select    tx_nodename
from      taxonomy
where     tx_nodename  in  ( " ", NULL )
```

Data Duplication (the second structural integrity constraint)

If a user stores an input node without knowledge of the existence of the same node, data duplication occurs. We must avoid duplication with the exception of "sp.", which means unknown species and represents an unknown node name. We can show the situation using the following query:

```
select    tx_nodename
from      taxonomy
where     tx_nodename ≠ "sp." -----
exception
group by  tx_nodename
having    count( tx_nodename ) > 1
```

Invalid Pointers to Parent Nodes (the third and forth structural integrity constraints)

The normal tree structure has invalid (undefined) pointers for only the parent node of the root node and valid pointers for the parent node of another node. An abnormal tree structure including errors, has an invalid pointer for the parent node of nodes other than the root node. We can show the situation using the following two queries:

```
select    tx_nodename
from      taxonomy, taxlevel
where     tx_tl_id    = tl_id
and       tl_levname ≠ "root"
and       tx_tx_idp   = NULL

select    tx_nodename
from      taxonomy  x
where     x.tx_tx_idp ≠ NULL
and       not exists(
             select  tx_id
             from    taxonomy
             where   tx_id = x.tx_tx_idp )
```

The first cooperative integrity constraint

This checks whether a pair of child nodes have the same name but one of the parent nodes has a different name. If the database is inconsistent, the following query does not have an empty solution:

```
select    x.tx_nodename
from      taxonomy  x,  other_taxonomy  y
where     x.tx_nodename  =  y.tx_nodename
and       x.tx_tx_idp    ≠  y.tx_tx_idp,
```

where the "taxonomy" table is one of the tables stored in the DDBJ-taxonomy database and the "other_taxonomy" table is one of the tables stored in the EMBL/GenBank/GSDB-taxonomy database. If the query has one or more solutions, it means that the DDBJ-taxonomy database is inconsistent and error nodes are shown on a display.

Table 1. Computer environments

Category / Databank Name	Database System	Computer System
DDBJ (NIG)	Sybase	CRAY SMP-42
GenBank (NCBI)	Sybase	Sun690
GSDB (LANL)	Sybase	Sun2000
EMBL (EBI)	ORACLE	Micro VAX 3100/80

The second cooperative integrity constraint

This checks whether a pair of nodes have the same name but one of the nodes has a different level (or ranking) name. If the database is inconsistent, the following query does not have an empty solution:

```
select    x.tx_nodename
from      taxonomy  x,  other_taxonomy  y
          taxlevel  z,  taxlevel w
where     x.tx_nodename  =  y.tx_nodename
and       x.tx_tl_id     =  z.tl_id
and       y.tx_tl_id     =  w.tl_id
and       z.tl_levname   ≠  w.tl_levname
```

We defined these SQL expressions as stored procedure in the relational database management system, SYBASE. The stored procedure including the structural integrity constraints is named *"self"* and the another procedure including the cooperative integrity constraints is named *"diff"*.

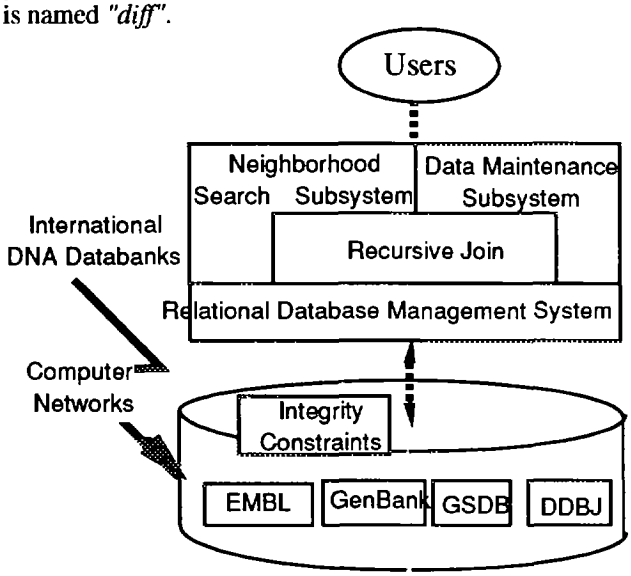

Figure 3. System configuration

6. System Configuration

Figure 3 shows the system configuration for building a consistent taxonomy database. The neighborhood search and database maintenance subsystems are developed by defining *stored procedures* implemented in both *SQL* and *Control Flow Language (CFL)* of SYBASE. The DDBJ-taxonomy database has integrity constraints which are defined by the stored procedure, since it is necessary to construct a consistent database. We need other taxonomy databases to define the cooperative integrity constraints. Other taxonomy databases are acquired automatically from EMBL, GenBank and GSDB over international computer networks. Table 1 shows the computer environments of each international DNA data bank.

Figure 4 shows the building process of the taxonomy database. After detecting error nodes using the previous integrity constraints, we can analyze the error structures for each error node to correctly revise all the error nodes. The structure is clarified by the neighborhood search functions which provide *lineage, posterity* and *homology* search processing for the tree structure. After the analysis is completed, we can revise the taxonomy database using data maintenance functions, which provide *allocation of new nodes, linkage between two nodes, merging among identical nodes* and *garbage collection for unused nodes*. If we can finish the revision for one error node, we can repeatedly do the same processing for other error nodes. Execution traces using these commands are shown in Appendix-2.

In addition, we implemented both a *write* (or save) function to disk and an *undo* (or restore) function to execute error recovery for data revision. Both of them are implemented in CFL and the transaction management function of SYBASE.

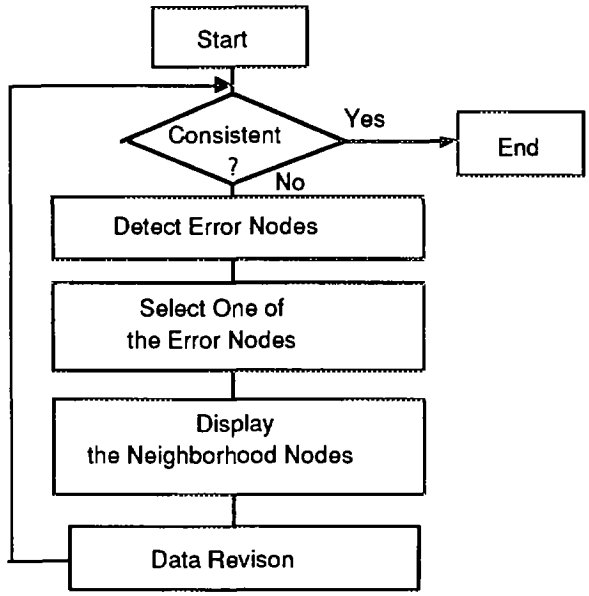

Figure 4. Building process for the taxonomy database

7. Related Work

TaxMan (Federhen 1994) and TAXSON (Clark & Chung & Yu 1993) produced by NCBI are building and search systems for the taxonomy database, respectively. AWB (Annotators workbench) (Burks & Cinkosky & Gilna & Fickett & Gifford et al. 1990-1992) created at LANL and IDEA (Interactive Data Entry and Annotation) (Kahn & Cameron 1990, Higgins & Fuchs et al. 1992) at EBI are, respectively, building systems for not only the DNA database but also the taxonomy database. AWB and IDEA have a mutual connection between the DNA and taxonomy databases.

Though the number of taxonomy nodes has increased to 2 times with the DNA database in the past 2 years, the taxonomy database managed by TaxMan is specified in ASN.1 (Abstract Syntax Notation 1) and is stored in a flat-file system. The flat-file system is inadequate for managing the taxonomy database, since we need both flexibility and high performance to manage the taxonomy database. TaxMan has the same neighborhood search functions as our system, but does not have an automatic error detection function using the integrity constraint concept or a powerful merging function among the identical nodes. Moreover, TaxMan does not have a posterity search function with

computation of statistics, either. The taxonomy database managed by TAXSON is stored in the relational database system, SYBASE. The TAXSON database is loaded from the TaxMan database; this allows users to use the power of the relational model for integrating the data with other systems and the power of the tree-structured model for maintaining the data itself.

Both AWB and IDEA are stored in the relational database system, but neither has the useful neighborhood search function. They have a function which searches the taxonomy database from the DNA database but another function to do it in the reverse.

8. Conclusions

We presented a new methodology for implementation of a system to build consistency in the taxonomy database. We focused on the existent taxonomy databases and presented methodologies designed for error detection and revision. The error detection mechanisms are represented by the structural and cooperative integrity constraints specified as stored procedures. The procedures are implemented in both SQL and CFL. If we apply an artificial intelligence approach to the implementation, we can obtain another kind of a program implemented in prolog (BIM 1990). This approach is made up with a smaller program than that is the previous approach. If we need more complex and higher processing to search and integrate taxonomy databases in the future, the artificial intelligence approach would be preferable in implementing more efficient processing.

Moreover, we proposed neighborhood search functions using the recursive join technique. The neighborhood search function provides a method for revising error nodes detected by these constraints in perspective. The error revisions are carried out using the data maintenance subsystem.

The system has been successfully used to repair the DDBJ-taxonomy database by DDBJ-staff since early autumn 1993. DDBJ-staff could repair 30 percent of about 4,000 errors were found by both the structural and cooperative integrity constraints at DDBJ.

We are planning to automatically unify (or integrate), the processes from error detection to the error revision using a graphic interface. The implemented system has recursive join processing as a search engine. Since the taxonomy database is increasing in size in an explosive manner, it is important to realize high performance for the recursive join. Parallel processing (Zang et al. 1993) is one possible solution for high performance.

Acknowledgments

We wish to thank David Lipman, James Ostell, Tim Clark and Scott Federhen of NCBI for helpful discussion. We also thank Michael J. Cinkosky and Maria Mcleod of LANL for comments regarding taxonomy revisions. Thanks also to Rainer Fuchs and David Hazledine of EMBL Data Library for their help in automatically obtaining the EMBL-taxonomy database with the relational model. Finally, we also thank Mari Saito of DDBJ and all the other DDBJ-staff involved in developing the taxonomy database system.

References

Patricia Kahn and Graham Cameron 1990, EMBL Data Library, *Methods in Enzymology*, Vol. 183.

Desmond G. Higgins, Rainer Fuchs, Peter J.Stoeher and Graham N.Cameron 1992, The EMBL Data Library, *Nucleic Acids Research*, Vol.20, *Oxford University Press.*

Christian Burks, Michael J. Cinkosky, Paul Gilna, et al 1990, GenBank: Current Status and Future Directions, *Methods in Enzymology*, Vol. 183.

Michael J. Cinkosky, James W. Fickett, Paul Gilna, and Christian Burks 1991, Electronic Data Publishing and GenBank, *Science*, Vol. 252.

Christian Burks, Michael J. Cinkosky,William M. Fischer, Paul Gilna, Jamie E.-D.Hayden, Gifford M. Keen, Michael Kelly, David Kristofferson and Julie Lawrence 1992, GenBank, *Nucleic Acids Research*, Vol.20, *Oxford University Press.*

Leslie Roberts 1993, Research News,Managing the Genome Data Deluge, *Science*,Vol.262, No.22.

Hajime Kitakami, Yukiko Yamazaki, Kazuho Ikeo, Yoshihiro Ugawa, Tadasu Shini, Naruya Saitou, Takashi Gojobori, and Yoshio Tateno 1994, Building and Search System for a Large-scale DNA Database, *Digest of One-Day Colloquim "Molecular Bioinformatics", The Institution of Electrical Engineers (IEE) Press*, London.

Sybase Inc. 1989, SYBASE Transact-SQL User's Guide.

Francois Bancilhon and Raghu Ramakrishnan 1986, An Amateur's Introduction to the Recursive Query Processing Strategies, *Proceedings of the ACM SIGMOD '86*, Washington D.C., pp.16-52.

Prabhat K. Andleigh and Michael R. Gretzinger 1992, Distributed Object-Oriented Data-Systems Design, *Prentice Hall Inc.*

Hajime Kitakami, Susumu Kunifuji, Taizo Miyachi, and Koich Furukawa 1984, A Methodology for Implementation of a Knowledge Acquisition System, *Proceedings of the IEEE 1984 International Symposium on Logic Programming, IEEE Computer Society*, pp.131-142

Scott Federhen 1994, TaxMan User's Manual, *To be available at NCBI's ftp site ("ncbi.nlm.nih.gov").*

Tim Clark, Cynthia Chung, and X. Yu 1993, The NCBI Taxonomy Database, *NCBI Preliminary Report.*

BIM 1990, Prolog by BIM(BIM_Prolog) Reference Manual, *BIM (Belgium).*

Yanchung Zang, Xiaofang Zhou, and Maria Orlowska 1993, On Horizontal Fragmentation and Computation of Transitive Closures, *Proceedings of the International Symposium on Next Generation Database Systems and Their Applications, Kyushu University*, pp.49-55, Fukuoka, Japan,.

Appendix-1. An Example of Programming for a Lineage Search Implemented in both SQL and CFL

```
create procedure lineage @nodename char(80) as

declare @tuples int
declare @cnt   int

select  @cnt=1
create table tempdb..lineage(
        tx_id char(16),tx_idp char(16),
          level tinyint ,level_name char(32),node_name char(80))
create table tempdb..temp_table(
          tx_id char(16),tx_idp char(16),
          level tinyint ,level_name char(32),node_name char(80))
create table tempdb..work_table(
        tx_id char(16),tx_idp char(16),
          level tinyint ,level_name char(32),node_name char(80))

insert tempdb..temp_table
   select tx_id,tx_tx_idp,@cnt,tl_levname,tx_nodename
   from  taxonomy,taxlevel
   where  tx_tl_id=tl_id
   and   tx_nodename=@nodename
insert tempdb..work_table
   select * from tempdb..temp_table
insert tempdb..lineage
   select * from tempdb..temp_table
select @tuples=count(*) from tempdb..temp_table
delete tempdb..temp_table

while @tuples!=0                                  /* Begin Recursive Join */
begin
     select @cnt=@cnt+1
     insert tempdb..temp_table
          select x.acc_num,y.tx_id,y.tx_tx_idp,   /*     Single Join      */
                 @cnt,tl_levname,tx_nodename
          from  tempdb..work_table x,
                taxonomy y,taxlevel
          where  tx_tl_id=tl_id
          and    y.tx_id=x.tx_idp

     delete tempdb..work_table

     insert tempdb..work_table
             select * from tempdb..temp_table
     insert tempdb..lineage
             select * from  tempdb..temp_table

     select @tuples=count(*)
     from  tempdb..temp_table

     delete tempdb..temp_table
end                                               /*  End Recursive Join */

select * from tempdb..lineage
drop table tempdb..lineage, tempdb..temp_table, tempdb..work_table
```

Appendix -2. Execution Traces

This shows two examples of the building process for the DDBJ-taxonomy database.

(Example-1) Structural Integrity Constraints

In this example, redundant nodes are detected by executing *"self"* command which executes the second structural integrity constraint. We try to repair inconsistencies in the database. Let us select "family(Corticiaceae)" node as one of redundant nodes. After checking a structure of the redundant node by use of *"posterity"* command, the redundant node is merged by *"merge"* command. After that, we are looking at neighborhood nodes around the "Corticiaceae" node using *"posterity"* command.

```
1> self ddbj, 2, prn
2> go
tx_id,      tx_idp,     Rank( Node)
-----------------------------------------------------
  :
ddbj12308,  ddbj12278,  family( Corticiaceae)
ddbj19356,  ddbj12278,  family( Corticiaceae)
  :

(2032 rows affected, return status = 0)

1> posterity ddbj, name, Corticiaceae
2> go

Number  tx_id,      tx_idp,     Rank( Node)
------  ----------------------------------------------
  0     ddbj12308,  ddbj12278,  family( Corticiaceae)
  0     ddbj19356,  ddbj12278,  family( Corticiaceae)
  1     ddbj16463,  ddbj12308,  genus( Athelia)
  1     ddbj19355,  ddbj19356,  genus( Phlebia)
  1     ddbj12309,  ddbj12308,  genus( Phanerochaete)
  1     ddbj16957,  ddbj12308,  genus( Christiansenia)

(6 rows affected, return status = 0)

1> merge ddbj, name, Corticiaceae
2> go

tx_id      Node_name
---------  ----------------------------------------------
ddbj12308  Corticiaceae...Garbage

(1 rows affected, return status = 0)

1> posterity ddbj, name, Corticiaceae
2> go

Number  tx_id,      tx_idp,     Rank( Node)
------  ----------------------------------------------
  0     ddbj19356,  ddbj12278,  family( Corticiaceae)
  1     ddbj16463,  ddbj19356,  genus( Athelia)
  1     ddbj19355,  ddbj19356,  genus( Phlebia)
  1     ddbj12309,  ddbj19356,  genus( Phanerochaete)
  1     ddbj16957,  ddbj19356,  genus( Christiansenia)

(5 rows affected, return status = 0)
1>
```

(Example-2) Cooperative Integrity Constraints

In this example, we detect errors through comparison with the DDBJ- and GSDB-taxonomy databases using *"diff"* command which executes the first cooperative integrity constraint. Both "family(Corticiaceae)" and "family(Polyporaceae)" nodes have same parent in the DDBJ-taxonomy database. We try to look at the neighborhood nodes around the "Corticiaceae" node using *"homology"* and *"lineage"* commands. The neighborhood nodes are shown in Figure 5. Moreover, we look at the GSDB- and NCBI-taxonomy databases using *"lineage"* commands. After that, we find that we need to create the "Aphyllopharales" node and link it to the "Hymenomycetes" node. In addition, we find that the "Corticiaceae", "Ganodermataceae", "Polyporaceae", and "Schizophyllaceae" nodes should be linked to the "Aphyllopharales" node. Finally, we display the neighborhood nods related to repaired nodes using both *"homology"* and *"posterity"* commands.

```
1> diff ddbj, gsdb, parent, prn
2> go

tx_id,      tx_idp,     Rank( Node)
-----------------------------------------------------
  :
ddbj19356,  ddbj12278,  family( Corticiaceae)
ddbj12311,  ddbj12278,  family( Polyporaceae)
  :

(631 rows affected, return status = 0)

1> homology ddbj, name, Corticiaceae
2> go

tx_id,      tx_idp,     Rank( Node)
-----------------------------------------------------
ddbj12279,  ddbj12278,  family( Agaricaceae)
ddbj12282,  ddbj12278,  family( Boletaceae)
ddbj12304,  ddbj12278,  family( Coprinaceae)
```

```
ddbj19356,  ddbj12278, family( Corticiaceae)
ddbj16350,  ddbj12278, family( Ganodermataceae)
ddbj12311,  ddbj12278, family( Polyporaceae)
ddbj12314,  ddbj12278, family( Russselaceae)
ddbj12317,  ddbj12278, family( Schizophyllaceae)
ddbj12324,  ddbj12278, family( Tricholomataceae)

(9 rows affected, return status = 0)

1> lineage ddbj, name, Corticiaceae
2> go

Number  tx_id,      tx_idp,     Rank( Node)
------  ----------------------------------------------
  1     ddbj10010,          ,  super_kingdom( Eukaryota)
  2     ddbj21016,  ddbj10010,  kingdom( Fungi)
  3     ddbj12532,  ddbj21016,  division( Eumycota)
  4     ddbj12241,  ddbj12532,  sub_division( Basidiomycotina)
  5     ddbj12277,  ddbj12241,  class( Hymenomycetes)
  6     ddbj12278,  ddbj12277,  order( Agaricales)
  7     ddbj19356,  ddbj12278,  family( Corticiaceae)

(7 rows affected, return status = 0)

1> lineage gsdb, name, Corticiaceae
2> go

Number  tx_id,      tx_idp,    Rank(Node)
------  ----------------------------------------------
   1    gsdb10010,          , super_kingdom( Eukaryota)
   2    gsdb12240,  gsdb10010, kingdom( Fungi)
   3    gsdb17277,  gsdb12240, division( Eumycota)
   4    gsdb12241,  gsdb17277, sub_division( Basidiomycotina)
   5    gsdb12277,  gsdb12241, class( Hymenomycetes)
   6    gsdb20488,  gsdb12277, order( Aphyllophorales)
   7    gsdb19356,  gsdb20488, family( Corticiaceae)

(7 rows affected, return status = 0)

1> lineage ncbi,name,Corticiaceae
2> go

Number  tx_id,      tx_idp,   Rank(Node)
------  ----------------------------------------------
   1    ncbi2759,          , kingdom( Eucaryotae)
   2    ncbi4751,  ncbi2759, phylum( Eumycota)
   3    ncbi5204,  ncbi4751, subphylum( Basidiomycotina)
   4    ncbi5302,  ncbi5204, superclass( Hymenomycetes)
   5    ncbi5303,  ncbi5302, order( Aphyllophorales)
   6    ncbi5304,  ncbi5303, family( Corticiaceae)

(6 rows affected, return status = 0)

1> newnode ddbj, Aphyllophorales, order, id, "12277"
2> go

tx_id      Node_name         Parent_name     tx_tx_idp
---------  ----------------- --------------- ---------
ddbj21936  Aphllophorales     Hymenomycetes  ddbj12277

(1 rows affected, return status = 0)

1> link ddbj, name, Corticiaceae, Aphllophorales
2> go

tx_id      tx_tx_idp  Node_name         Parent_name
---------  ---------- ----------------- ---------------
ddbj19356  ddbj21936  Corticiaceae      Aphllophorales

(1 rows affected, return status = 0)

1> link ddbj, name, Polyporaceae, Aphllophorales
2> go

tx_id      tx_tx_idp  Node_name         Parent_name
---------  ---------- ----------------- ---------------
ddbj12311  ddbj21936  Polyporaceae      Aphllophorales

(1 rows affected, return status = 0)

1> link ddbj, name, Ganodermataceae, Aphllophorales
2> go

tx_id      tx_tx_idp  Node_name         Parent_name
---------  ---------- ----------------- ---------------
ddbj16350  ddbj21936  Ganodermataceae   Aphllophorales

(1 rows affected, return status = 0)

1> link ddbj, name, Schizophyllaceae, Aphllophorales
2> go

tx_id      tx_tx_idp  Node_name         Parent_name
---------  ---------- ----------------- ---------------
ddbj12317  ddbj21936  Schizophyllaceae  Aphllophorales

(1 rows affected, return status = 0)

1> lineage ddbj, name, Corticiaceae
2> go

Number  tx_id,      tx_idp,     Rank( Node)
------  ----------------------------------------------
  1     ddbj10010,          ,  super_kingdom( Eukaryota)
  2     ddbj21016,  ddbj10010,  kingdom( Fungi)
  3     ddbj12532,  ddbj21016,  division( Eumycota)
  4     ddbj12241,  ddbj12532,  sub_division( Basidiomycotina)
  5     ddbj12277,  ddbj12241,  class( Hymenomycetes)
  6     ddbj21936,  ddbj12277,  order( Aphyllophorales)
  7     ddbj19356,  ddbj21936,  family( Corticiaceae)

(7 rows affected, return status = 0)

1> posterity ddbj, name, Aphyllophorales
2> go

Number  tx_id,      tx_idp,     Rank( Node)
------  ----------------------------------------------
  0     ddbj21936,  ddbj12277,  order( Corticiaceae)
  1     ddbj12311,  ddbj21936,  family( Polyporales)
  1     ddbj19356,  ddbj21936,  family( Corticiaceae)
  1     ddbj16350,  ddbj21936,  family( Ganodermataceae)
  1     ddbj12317,  ddbj21936,  family( Schizophyllaceae)

(6 rows affected, return status = 0)

1> homology ddbj, name, Agaricaceae
2> go

tx_id,      tx_idp,    Rank( Node)
----------------------------------------------------
ddbj12279,  ddbj12278, family( Agaricaceae)
ddbj12282,  ddbj12278, family( Boletaceae)
ddbj12304,  ddbj12278, family( Coprinaceae)
ddbj12314,  ddbj12278, family( Russselaceae)
ddbj12324,  ddbj12278, family( Tricholomataceae)

(5 rows affected, return status = 0)

1>
```

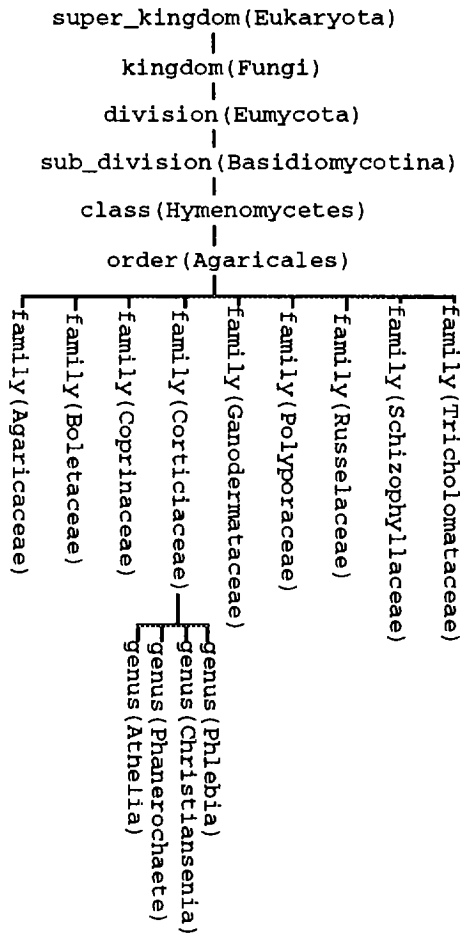

Figure 5. The neighborhood nodes around the "family(Corticiaceae)" node in the DDBJ-taxonomy database