CHAPTER 6

RESULTS AND DISCUSSION

6.1 INTRODUCTION

This chapter documents the results of proposed association rules mining algorithms using real datasets.

This thesis re-presented the following two pattern mining algorithms i and ii are compared with proposed algorithm iii. RSDCA

i) Rapid Closed Frequent Itemset Algorithm,

ii) Rapid Frequent Pattern and these re-presented algorithms are compared with proposed algorithm

iii) Refined Search Divide and Conquer Algorithm (RSDCA).

6.2 EXPERIMENTAL SETUP

The performance of above said algorithms are independently compared with existing algorithm in earlier chapters but this chapter showcases comparison related to execution time of all the three algorithms with various datasets.
6.3 COMPARATIVE STUDY

This chapter used chess and mushroom dataset for comparing performance of proposed algorithm RSDCA with RCFIA and RFP. This research is used support threshold to compare efficiency of the Rapid Closed Frequent Itemset mining Algorithm (RCFIA), Rapid search based Frequent Pattern mining algorithm (RFP) with Refined Search Divide and conquer Algorithm (RSDCA) algorithms. The following figures from 6.1 to 6.2 are illustrated the execution time of the proposed algorithm RSDCA with above mentioned dataset.
Figure 6.1 Comparison of RCFIA, RFP and RSDCA using Chess dataset in various Support level
Figure 6.2 Comparison of RCFIA, RFP and RSDCA using Mushroom dataset in various Support level
The proposed algorithm is compared by using Chess and Connect datasets with various support threshold. In low support RSDCA performed well compared with RFP and RCFIA. In higher support level, two algorithms RFP, RCFIA performed inferior when compared to RSDCA. The proposed rapid searching technique is adapted with apriori based frequent itemset, FP-Growth based and closed set mining algorithm.

Proposed RSDCA adapted for multi processor system for execute the programs, where the communication between processors and enhance the memory caches effectively. This research deals the memory caches and related search using divide and conquer approach. Divide and Conquer algorithm yield accurate and precise results than equivalent iterative method.

Defined simple loop that adds each datum to a single variable or by a Divide and Conquer Algorithm called pair wise summary that divides the data set into two sets. Then the divide and conquer method computes the each set recursively, and then adds the two sums. Even the second method uses the same number of additions as first, and gives extra work to the recursive calls, it is more perfect.

Frequent pattern mining in a large database plays an important role in many data mining task. Much kind of methods adopts apriori-like candidate-generation-and-test approaches. However, above said methods are come across the severe phases and challenges in mining datasets with lengthy patterns. But the proposed RSDCA uses the Divide and Conquer Algorithm for not only the best utilization of memory caches and also effectively use the
processor speed. In very large data sets it is very difficult to extract the frequent patterns within the system memory space and with high speed. The system needs extra memory caches and high speed processor or it has to read the datasets more than once. But in proposed RSDCA with Divide and Conquer Algorithm uses the system memory caches only once since it divide the problems into more than sub problem. And it executes the problems recursively with the help of multiprocessor within the memory caches itself.

Hence, this RSDCA first develop the required or needed sub-tree only instead of developing the full FP tree. The Full FP tree may not fit in the system memory caches itself. But the proposed RSDCA system constructs the searching elements sub-tree only. And the Divide and Conquer method which is used in the proposed algorithm divide the sup-tree into more than one conventional sub-tree recursively to find the frequent patterns. At lost the result of more than one conditional sub-tree are merged to form the result of original one.

The multi processor which is used in the system help to process the problem parallel. And the memory cache within the system is enough to process the more than one sub-problems recursively. In this way the proposed system utilizes the memory caches and processors effectively and efficiently.
The proposed rapid search technique is improved the performance of apriori based frequent itemset mining algorithms.

6.4 SUMMARY

This chapter is primarily focused to compare the support threshold and efficiency of the proposed Refined Search Divide and Conquer Algorithm (RSDCA) with Rapid Closed Frequent Itemset mining Algorithm (RCFIA) and Rapid search based Frequent Pattern mining algorithm (RFP) on the voluminous data bases for finding hidden itemsets using this efficient and innovative association based algorithms for rapid search mining also find the efficiency and speed research through execution time calculation.
CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

The thesis has investigated three association rule mining algorithms using rapid search technique. The efficiency of algorithms, conciseness of results and problem of rule mining with rapid searching technique are discussed in each chapter. This chapter concludes about RSDCA.

An array of novel algorithms have been proposed in the recent past to produce efficient mining of frequent itemsets, each in turn claiming to outperform its predecessors on a set of standard databases. The proposed rapid search technique was using frequent and closed itemset mining to improve performance of mining operations. The main advantages of the algorithm over existing algorithms derived through the data structures and consideration of memory setup and related execution time. Further this research deals

1. Interactive mining with different supports.
2. Faster execution time.
3. Reduce large amount of search spaces.

The RSDCA algorithm have implemented and compared with Apriori, AprioriTID, FP-growth and Eclat algorithms. The experimental results are also showed that the RSDCA is more efficient and scalable in large datasets and outperform than other above mentioned algorithms.
This research proposed rapid search technique for mining frequent itemsets from large voluminous databases. Using this technique, this thesis proposed a new Rapid Searching based on FP growth algorithm. These methods more efficient compared to traditional Apriori, FP-Growth and Eclat.

This research work offers a new association rule mining algorithm using closed frequent itemset. The RFCIA is also adapted rapid searching procedure for mining frequent itemsets. Experimental results clearly showcases that RFCIA algorithm finds most of the possible association rules with satisfying minimum thresholds. It performed well in high as well as small dense datasets.

The proposed rapid searching procedure is successfully integrated with frequent and closed frequent pattern mining algorithm. It means that the rapid search procedure is mined frequent itemsets from synthetic and real datasets. This procedure plays a major role to improve the performance of these rule mining algorithms.
7.2 FUTURE ENHANCEMENT

There are many opportunities and challenges for continued investigation related to association rule mining and rules.

The RSDCA algorithm is compared with recent and various pattern mining algorithms. This algorithm implemented using hash tree can be tested with some other data structures. The future work may be incorporated the constraints with RSDCA procedure to generate domain dependent itemsets.

The rapid searching technique can be extended with Eclat algorithms. This technique is successfully interacted with synthetic and some real dataset but in future it can extend with retail market, telecommunication, network analysis, security domains.

The implementation of RSDCA, RFP and RCFIA algorithms need to be extended to handle the commercial database for building a commercial package. This extension is conceptually straight forward. The algorithms designed and evaluated in this thesis address the computationally expensive step of RSDCA, RCFIA and RFP namely, to produce frequent itemsets and association rules. A commercial system would need to utilize the output of this step to produce association rules and present them to the end user using an intuitive and appealing front-end.
Annexure

I. Sample database testified using oracle objects

SQL> steps in creating a database

SQL> 1. create the parameter file for a new database

SQL> 2. create a service under windows nt/2000/xp for running oracle as a service (typically) for the instance

SQL> 3. connect using sql*plus as internal

SQL> 4. shutdown the database

SQL> 5. startup the instance / database with your new parameter file by specifying the location of the parameter file eg startup pfile=c:\uma\init.ora

SQL> shutdown immediate

Database closed.

Database dismounted.

ORACLE instance shut down.

SQL> startup nomount;

ORACLE instance started.

Total System Global Area 87861276 bytes

Fixed Size 75804 bytes

Variable Size 57094144 bytes

Database Buffers 30613504 bytes

Redo Buffers 77824 bytes
SQL> create database uma

2  controlfile reuse
3  maxlogfile 32
4  maxlogmembers 2
5  maxdatafiles 256
6  maxinstances 16
7  maxloghistory 1815
8  logfile
9  group 1 'c:\uma\logfiles\log1.log' size 5m
10  group 1 'c:\uma\logfiles\log1.log' size 5m,
11  group 2 'c:\uma\logfiles\log2.log' size 5m,
12  group 3 'c:\uma\logfiles\log2.log' size 5m
13  group 3 'c:\uma\datafile\uma.dbf' size 500m reuse
14  noarchivelog character set WE8ISO8859P1;

taxlogfile 32

SQL> @f:\oracle\ora81\rdbms\admin\catalog.sql;

Package created.

Package body created.

Grant succeeded.

View created.

drop public synonym v$statname

*
II. Data access through client–server mode using Middleware

DOC>Package UTL_HTTP, for making HTTP (hyper-text transfer protocol) callouts
DOC>from PL/SQL and SQL. It can be used to access data on the internet, or to call
DOC>Oracle Web Server Cartridges. The package contains two similar entrypoints,
DOC>each of which takes a string URL (universal resource locator), contacts that
DOC>site, and returns the data (typically HTML -- hyper-text markup language)
DOC>obtained from that site.

DOC>function request (url in varchar2, proxy in varchar2,
DOC>wallet_path in varchar2, wallet_password in varchar2)
DOC>return varchar2;

DOC>UTL_HTTP.REQUEST returns up to the first 2000 bytes of the data retrieved
DOC>from the given URL.
SVRMGR> select utl_http.request('http://www.oracle.com/')
   from dual;

UTL_HTTP.REQUEST('HTTP://WWW.ORACLE.COM/')

<html>
<head><title>Oracle Corporation Home Page</title>
<!--changed Jan. 16, 14
1 row selected.

The optional proxy parameter to request() can be used to specify a proxy
server to use when making the HTTP request.
The optional wallet_path and wallet_password parameters to request()
used to specify a client-side wallet. The client-side wall contains
the list of trusted certificate authorities required for HTTPS request.
format of wallet_path is 'file:/<local-dir-for-client-side-wallet>'.
wallet_password is the password required to open the wallet.
This is the specification of packaged function
UTL_HTTP.REQUEST_PIECES,
which uses type UTL_HTTP.HTML_PIECES:
type html_pieces is table of varchar2(2000) index by binary_integer;
function request_pieces (url in varchar2,
max_pieces natural default 32767,
proxy in varchar2 default NULL,
wallet_path in varchar2, wallet_password in varchar2)
`return html_pieces;`

`UTL_HTTP.REQUESTPieces` returns a PLSQL-table of 2000-byte pieces of the data retrieved from the given URL. The optional second argument places a bound on the number of pieces retrieved. For example, the following

```sql
set serveroutput on
/
declare
x utl_http.html_pieces;
begin
x := utl_http.request_pieces('http://www.oracle.com/', 100);
dbms_output.put_line(x.count || ' pieces were retrieved.');
dbms_output.put_line('with total length ');
if x.count < 1
then dbms_output.put_line('0');
else dbms_output.put_line((2000 * (x.count - 1)) + length(x(x.count)));
end if;
end;
/
```

Here is the output:
Statement processed.

4 pieces were retrieved.
with total length
7687

The optional proxy parameter to request_pieces() can be used to specify a
proxy server to use when making the HTTP request.

The optional wallet_path and wallet_password parameters to request() can
be used to specify a client-side wallet. The client-side wall contains
the list of trusted certificate authorities required for HTTPS request. The
format of wallet_path is 'file:/<local-dir-for-client-side-wallet>'.
wallet_password is the password required to open the wallet.

Below is the specification for package UTL_HTTP. It describes the
exceptions that can be raised by functions REQUEST and
REQUESTPieces:
III. Finding the objects through oracle transaction control methods

SQL> create or replace view DBA_PENDING_TRANSACTIONS(formatid, globalid, branchid)

2  as 
3  (((select formatid, globalid, branchid 
4  from  gv$global_transaction 
5  where  refcount = preparecount)
6  minus
7    (select global_tran_fmt, global_foreign_id, branch_id 
8  from  sys.pending_trans$ tran, sys.pending_sessions$ sess 
9  where  tran.local_tran_id = sess.local_tran_id 
10    and  tran.state != 'collecting'
11    and  dbms_utility.is_bit_set(tran.session_vector, sess.session_id)=1)
12  ) 
13  union
14  (select global_tran_fmt, global_foreign_id, branch_id 
15  from  sys.pending_trans$ tran, sys.pending_sessions$ sess 
16  where  tran.local_tran_id = sess.local_tran_id 
17    and  tran.state != 'collecting'
18    and  dbms_utility.is_bit_set(tran.session_vector, sess.session_id)=1)
19  );

View created.
SQL> drop public synonym DBA_PENDING_TRANSACTIONS;
Synonym dropped.

SQL> create public synonym DBA_PENDING_TRANSACTIONS for DBA_PENDING_TRANSACTIONS;
Synonym created.

SQL> grant select on DBA_PENDING_TRANSACTIONS to select_catalog_role;
Grant succeeded.

SQL> comment on table DBA_PENDING_TRANSACTIONS is
  2  'information about unresolved global transactions';
Comment created.

SQL> comment on column DBA_PENDING_TRANSACTIONS.formatid is
  2  'format identifier of the transaction identifier';
Comment created.

SQL> comment on column DBA_PENDING_TRANSACTIONS.globalid is
  2  'global part (gtrid) of the transaction identifier';
Comment created.
SQL> comment on column DBA_PENDING_TRANSACTIONS.branchid is
2  'branch qualifier (bqual) of the transaction identifier';

Comment created.

SQL> set echo off

Grant succeeded.

Type created.

Grant succeeded.

Sequence created.

Table created.

0 rows deleted.

1 row created.

Commit complete.

View created.

Comment created.

drop public synonym DBA_QUEUE_TABLES
*

SQL> select tablespace_name, status from dba_tablespaces;

<table>
<thead>
<tr>
<th>TABLESPACE_NAME</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>ONLINE</td>
</tr>
</tbody>
</table>
SQL> create tablespace users
2  datafile 'c:\uma\datafile\users.dbf' size 100m;

Tablespace created.

SQL> create tablespace temp1
2  datafile 'c:\uma\datafile\temp.dbf' size 100m extent management local
uniform size 1m;
Tablespace created.

SQL> create tablespace inx
2  datafile 'c:\uma\datafile\index.dbf' size 100m;
Tablespace created.

SQL> select tablespace_name, status from dba_tablespaces;

<table>
<thead>
<tr>
<th>TABLESPACE_NAME</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>ONLINE</td>
</tr>
<tr>
<td>USERS</td>
<td>ONLINE</td>
</tr>
<tr>
<td>TEMP1</td>
<td>ONLINE</td>
</tr>
<tr>
<td>INX</td>
<td>ONLINE</td>
</tr>
</tbody>
</table>
SQL> desc dba_tablespaces;

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLESPACE_NAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>INITIAL_EXTENT</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>NEXT_EXTENT</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>MIN_EXTENTS</td>
<td>NOT NULL</td>
<td>NUMBER</td>
</tr>
<tr>
<td>MAX_EXTENTS</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>PCT_INCREASE</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>MIN_EXTLEN</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>STATUS</td>
<td></td>
<td>VARCHAR2(9)</td>
</tr>
<tr>
<td>CONTENTS</td>
<td></td>
<td>VARCHAR2(9)</td>
</tr>
<tr>
<td>LOGGING</td>
<td></td>
<td>VARCHAR2(9)</td>
</tr>
<tr>
<td>EXTENT_MANAGEMENT</td>
<td></td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>ALLOCATION_TYPE</td>
<td></td>
<td>VARCHAR2(9)</td>
</tr>
<tr>
<td>PLUGGED_IN</td>
<td></td>
<td>VARCHAR2(3)</td>
</tr>
</tbody>
</table>

SQL> create table emp

2  ( epmno number primary key,
3   ename varchar2(50) not null,
4   eaddr varchar2(100) ,
5   ecity varchar2(30) ,
6   eemail varchar2(50));

Table created.
SQL> create procedure insert_emp as

2   eno number := 0;

3   ename varchar2(50) := 'Name eof the pwerson ';

4   eaddr varchar2(100) := 'address of the the name of the person generated by 
auto generator';

SQL> get insertemp.sql

1   create or replace procedure insert_emp(num in number)

2   is

3   eno number := 0;

4   vename varchar2(50) := 'Name eof the pwerson ';

5   veaddr varchar2(100) := 'address of the the name of the person generated by 
auto generator';

6   vecity varchar2(50) := 'Chennai';

7   veemail varchar2(50) := 'email.addrof.';

8   begin

9   for i in 1..num loop

10  eno:=i;

11  insert into emp values (i,vename||'-'||to_char(eno),veaddr||'
||to_char(eno),vecity,veemail||vename||'-'||to_char(eno)||'yahoo.com');

12  end loop;

13*  end;

14*  /

Procedure created.
SQL> connect internal;

Connected.

SQL> create tablespace roll_back
  2  datafile 'c:\uma\datafile\rollback.ora' size 100m extent management local;

Tablespace created.

SQL> alter tablespace roll_back online;

Tablespace altered.

SQL> create rollback segment rbs2 tablespace system;

Rollback segment created.

SQL> alter rollback segment rbs2 online;

Rollback segment altered.

SQL> spool to
SQL> get insertemp.sql

create or replace procedure insert_emp(num in number)
is
eno number := 0;
vename varchar2(50) := 'Name of the person';
veaddr varchar2(100) := 'address of the person generated by auto generator';
vecity varchar2(50) := 'Chennai';
veemail varchar2(50) := 'email.
begin
for i in 1..num loop
  eno:=i;
  insert into emp values (i,vename||'-'||to_char(eno),veaddr||'

  vecity,veemail||vename||'-'||to_char(eno)||'@yahoo.com');
end loop;
end;
/
Procedure created.
SQL> execute insert_emp(5);
BEGIN insert_emp(5); END;

SQL> connect internal;
Connected.

SQL> create rollback segment rbs1 tablespace users;
Rollback segment created.

SQL> alter rollback segment rbs1 online;
Rollback segment altered.

SQL> connect ramki/ramki;
Connected.

SQL> execute insert_emp(5);
PL/SQL procedure successfully completed.

SQL> connect internal;
Connected.

SQL> create tablespace roll_back
2  datafile 'c:\uma\datafile\rollback.ora' size 100m extent management local;
Tablespace created.
SQL> alter tablespace roll_back online;
Tablespace altered.
SQL> create rollback segment rbs2 tablescape system;
Rollback segment created.
SQL> alter rollback segment rbs2 online;
Rollback segment altered.
SQL> spool to
SQL> select tablespace_name "tablespace",
       filen_name "datafile",
       blocks "total blocks",
       bytes from DBA_DATA_FILES:
SQL> select tablespace_name "tablespace",
       filen_name "datafile",
       blocks "total blocks"
       from DBA_DATA_FILES
       /
filen_name "datafile",
```
SQL> desc DBA_DATA_FILES;

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_NAME</td>
<td></td>
<td>VARCHAR2(513)</td>
</tr>
<tr>
<td>FILE_ID</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>TABLESPACE_NAME</td>
<td></td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>BYTES</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>BLOCKS</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>STATUS</td>
<td></td>
<td>VARCHAR2(9)</td>
</tr>
<tr>
<td>RELATIVE_FNO</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>AUTOEXTENSIBLE</td>
<td></td>
<td>VARCHAR2(3)</td>
</tr>
<tr>
<td>MAXBYTES</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>MAXBLOCKS</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>INCREMENT_BY</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>USER_BYTES</td>
<td></td>
<td>NUMBER</td>
</tr>
<tr>
<td>USER_BLOCKS</td>
<td></td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

SQL> select tablespace_name ,
        filen_name, blocks, BYTES
      FROM DBA_DATA_FILES;

filen_name,
*