A New Implementation for Maximal Itemsets Miner using Oracle PL/SQL

Hussein K. Khafaji[a], Mais A. Al-Sharqi[b], Oleksiy Nedashkivskiy[c], and Pawel Falat[d]

[a] *Al-Rafidain University College, Baghdad, Iraq*
[b] *University of Information Technology and Communications Baghdad, Iraq*
[c] *National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," Kyiv, Ukraine*
[d] *University of Bielsko-Biala, 2 Willowa str., 43-309, Bielsko-Biala, Poland*

Abstract

The problem of determining how to implement a data mining system as looslly-coupled or tightly-coupled remains a major challenge as it affects the performance of the system and the consumption of memory and computation resources. The aim of the research is to propose a new approach to data mining design based on aggregating a data mining system with data intended for mining under the umbrella of database management systems. The authors produced a new algorithm to mine maximal itemsets depending on Bees' algorithm named Maximal Itemsets Mining Algorithm Based on Bees' Algorithm, MMIBA. MMIBA was implemented as loosely-coupled miner. This research presents a new implementation for MMIBA using Oracle PL\SQL. The aim of this implementation is to combine the mining system with the data to be mined. This approach excludes many drawbacks associated with other approaches such as the conflict of data environment and mining system environment, data transfer between these different environments, and data format conversion due to the mismatch of the formats that are supported by various environments. This approach dominants the loosely-coupled implementation in considerable amount of execution time and memory consumption. The proposed system was tested using many real and synthetic databases with wide range of properties including size, number of items, database sparseness or its density. Many values for minimum support and conflict were used in these tests to prove the robustness of the designed system. Experiments showed that the techniques of implementing mining algorithms affected their efficiency, and this was demonstrated by increasing the efficiency of MIMBA when it was implemented by collecting data with the miner that uses it in a single software environment.

Keywords

Maximal itemset, embedded miner, PL\SQL, MMIBA.

1. Introduction

The discovery of frequent itemsets (FI) is one of the very important subjects in data mining [1]. The FI concept was firstly produced by Agrawal et al. [2]. A frequent itemset is a group of items coming frequently together in set of transactions satisfying minimum support (min_sup) [3–5]. The big challenge lies in the huge number of FIs that are generated from mining process especially when min_sup is set to small value [2, 6].

A FI is maximal if it is the longest itemset that satisfies min_sup and all its superset is infrequent [7]. Maximal frequent itemset (MFI) considered as dense representation of FIs because of their relatively low number [6]. For this reason, MFIs has rapidly been a paramount task in data mining field [3]. The number of MFIs is so small in comparative with the number of FIs. All the FIs can be generated from the set of MFIs. Due to two reasons many attempts were done to mine maximal itemsets instead of the mining of FIs to keep time and efforts complexities [6, 7].

Emerging Technology Trends on the Smart Industry and the Internet of Things, January 19, 2022, Kyiv, Ukraine
EMAIL: dr.hkml811@yahoo.com (H. K. Khafaji); newmisk@gmail.com (M. A. Al-Sharqi); al_1@ua.fm (O. Nedashkivskiy); pfalat@ad.ath.bielsko.pl (P. Falat)
ORCID: 0000-0002-1830-0568 (H. K. Khafaji); 0000-0002-9059-9031 (M. A. Al-Sharqi); 0000-0002-1788-4434 (O. Nedashkivskiy); 0000-0002-3593-9750 (P. Falat)

The object of study is the process of mining MFIs by utilizing the capabilities of Oracle DBMS which provides PL/SQL as programming language to implement the mining system in addition to its characteristic as repository to hold the data. This utilization is accomplished by selecting a suitable mining algorithm that is implementable by PL/SQL.

The subject of study is the use of the PL/SQL or any programming tool provided by a DBMS to implement the mining system which depends on a PL/SQL implementable algorithm.

The purpose of the work is to increase a data miner's throughput with minimum allocated computer system resources by transferring many of the miner duties from the miner space to the BDMS space.

2. Problem Statement

Mining of the frequent itemsets from large transaction databases is one of the inherently difficult problems that classified as NP-complete problem. In addition to this difficulty, miners are implemented using loosely or tightly-coupled techniques, which leads to storing data in flat files that are difficult to manage, or that the data remotely stored in DBMS tables separated from the programming language environment, leading to a lot of I/O operations and repeated data transformation and transferring. Both techniques reduce the efficiency of algorithms and consume memory, computation resource, and I/O devices.

3. Review of the Literature

Durand et al [2] has introduced the FIBAD approach to compute nearly borders of frequent itemsets by utilizing dualization and computation of approximate minimal transversals of hypergraph. They used maximal frequent and infrequent itemsets to represent positive and negative borders respectively. They found that their method outperformed other methods in producing borders having the highest quality.

Ganesh et al [4] has mined maximal patterns from the large database used for knowledge discovery. The contribution of their paper includes (i)Transposed Representation of Database ii) reduction of Database scanning (iii) Pruning the candidate itemsets in each step. Vertical Frequent Mining is deemed simple to use and effectively implemented.

Kocak et al [7] are utilizing maximal frequent itemsets to propose a novel feature elicitation method based on data mining techniques for understanding significant patterns in octamer's cleavability of acquired immune deficiency syndrome disease, which is caused by human immunodeficiency virus. The extracted features are used in the classification process implies significant results which may be used when developing a new medicine.

Mais et al [6] has introduced a novel maximal itemsets algorithm by pairing MFI and Bees' algorithm (BA). They exhibited a MFI-oriented BA for the purpose of mining MFIs from transactional database (TDB). Mining Maximal Itemsets Bees' Algorithm (MMIBA) is population based stochastic algorithm. This paper depends on MMIBA therefore it will be explained in details. Elucidation of MMIBA algorithm is presented in Figure 1. Recall the parameters of MMIBA to be set such as N, M, E, nep, nsp, and ngh.

The MMIBA algorithm is population based, so it firstly prepares its population of FIs that satisfy min_sup by generating them randomly. It mines MFIs from TDB by selecting higher FIs and select the highest FIs out of the higher ones, making union between them in order to generate new generation. The union is done between the highest FIs and half number of randomly selected FI population. Where the remaining FIs after selection the highest (i.e. higher-highest) are united with quarter number of randomly selected FI population. Each of the remaining higher-FIs population unites with one randomly selected FIs.

1. Initialize population with random generated itemsets.
   1.1 Select frequent items from transactional DB (TDB) for new itemset generation.
   1.2 Determine the size of population represented by N.
   1.3 Generate unique N numbers represents N k-itemsets.
   1.4 Select k-items for each k-itemset from frequent items.

2. Evaluate support for each itemset.
3. While (stopping criterion not met) //generating new itemsets.
4. Select M longer frequent itemsets out of N and E longest frequent itemsets out of M.
5. Perform union operation (more for best itemsets)
   5.1 Perform union operation between each of E itemsets and half no. of frequent itemsets to generate new itemsets.
   5.2 Perform union operation between each of M–E itemsets and quarter no. of frequent itemsets to generate new itemsets.
   5.3 Calculate the support of the new generated itemsets.

6. Select the highest frequent itemset as candidate maximal frequent itemset.
7. Perform union operation for remaining itemsets randomly.
   7.1 Perform union operation between each of itemsets-M and one of randomly selected frequent itemsets to generate new itemsets.
   7.2 Calculate the support of the new generated itemsets.
8. End While.

Figure 1: MMIBA Algorithm

The MMIBA algorithm is population based, so it firstly prepares its population of FIs that satisfy min_sup by generating them randomly. It mines MFIs from TDB by selecting higher FIs and select the highest FIs out of the higher ones, making union between them in order to generate new generation. The union is done between the highest FIs and half number of randomly selected FI population. Where the remaining FIs after selection the highest (i.e. higher-highest) are united with quarter number of randomly selected FI population. Each of the remaining higher-FIs population unites with one randomly selected FIs.

4. Materials and Methods

Data mining algorithms usually deals with a huge data. Because these algorithms encounter issue of scalability with respect to vast data, traditional methods address this problem by selecting subset of data to be mined which is in-memory data [8, 9]. Another issue is that these algorithms depended on main-memory data structure, limited amount of data that can be handled [9]. In addition, most data mining algorithms can only be loosely coupled with data infrastructures in organization and are hard to pour into existing mission-critical application [10]. In loosely coupled connection, data is transferred either between database and main memory or dynamically in client/server architecture generating network traffic [11]. Most data mining algorithms have been used flat files as data source of information that means handling specific file-format. Data integrity is another issue that is needed to be solved via consolidation data mining methods with DBMSs [11-13]

DBMS technology presents many characteristics that make it valuable when performing data mining applications. Data sets, those are extremely larger than main memory, can be dealt with because the database itself is responsible for processing information, paging and swapping when needed. In addition, DBMS provides a simplified data management and closer integration to other systems [11].

Information used during mining processing is often clandestine. Consequently, DBMSs can be used as a way of supplying data security, which is the demand of many commercial databases. In this way, we avoid needing encryption algorithms to process information. [11]

Most versions of Oracle DBMS support tightly coupled integration within database through executing user-defined computation database thus averting unnecessary network traffic. One sort of this

implementation is the stored procedure, which is provided in Oracle PL/SQL within data dictionary. The aim of this paper is to introduce a new implementation of MMIBA using Oracle 11g PL\SQL to be supported with all the previously mentioned DBMS properties.

Some Oracle 11g Database Features

ORACLE DBMS is preferred to implement the maximal itemsets embedded miner because it has the following privileged features:

1. ORACLE supports external stored procedures and PL/SQL, the complete, block-structured programming language for data manipulation (DML), which excludes the need for host language.
2. ORACLE supports stored procedure within database that combines the DB with the user-developed programs to manipulate the data.
3. ORACLE supports Open Database Connectivity (ODBC) mechanism that is used to access the various DBs of DBMSs.
4. ORACLE supports Large Object Block (LOB), Character LOB CLOB, and Multi CLOB (MCLOB), data type, which allows us to store a very large number of transactions for one item or very long frequent patterns itemsets.
5. ORACLE supports compound object and nested tables.
6. ORACLE supports LONG data type, which holds up to 2 GB of data as one field.
   The last three features of ORACLE are very important to represent dense databases prepared for mining.
7. Subprogram inlining in 11g, ORACLE has presented the PL/SQL Function Result Cache, which eliminates the overhead of function calls by re-organizing the source code during compilation.
8. PL/SQL enhancements in ORACLE 11g of the following features: Easy and simplified PL/SQL native compilation, Improved PL/SQL stored procedure invalidation mechanism, Stored Procedure Named Notation, Virtual columns in 11g, Query result cache in ORACLE 11g, PL/SQL function result cache, and Regular expression enhancements.

Proposed Miners' Architecture

The architecture of the proposed miner is composed of many modules each of which is responsible for a specific task such that they collectively perform the task of finding the MFIs. These modules are initialization module, support counter module, frequent item selector module, m and e selector module, itemset-m selector module, new itemset generator module, maximal itemset selector module, and reporter module. (See Figure 2).

Proposed Miners' Modules

Frequent item selector module (level 1) In order to discover MFIs, frequent items, 1-itemsets, must be found firstly. In this module, only frequent items will be selected from TDB and stored in a table which this module will construct it. The table is named "frequent items DB". It consists of three columns: freq_item, sup, and item_len. The fields' data types used for all columns of modules tables are CLOB and Number as follows:

Freq_item	CLOB	--- stores frequent itemsets, 1-itemsets, 2-itemset, etc.
Sup	number	--- stores itemsets' supports.
Item_len	number	--- stores the length of itemset.

Itemsets may have huge size, therefore the most suitable data type to hold an itemset is CLOB data type. These FIs are encoded numerically such that each FI has given a unique code (integer number) which represents it during mining process. These unique sequential codes are stored in table called "coding" table. Coding table has the following column: *item, code* . Coding step has two advantages:

1) It simplifies the completely mining process, because it is easier to deal with item, feature, or property in DB as numbers than as names (sequence of characters) in terms of storing and retrieving FIs and performing union operations.
2) Because MMIBA relies on randomization to explore search space (itemsets space), it uses randomization function to generate random numbers ranging between min and max values to form an itemset. These min and max values are min and max code of that FI. To decide whether the item

is frequent or not, the support of the item must be calculated and compared with the input value defined by the user which represents the threshold between frequent and infrequent items, i.e. min_sup.

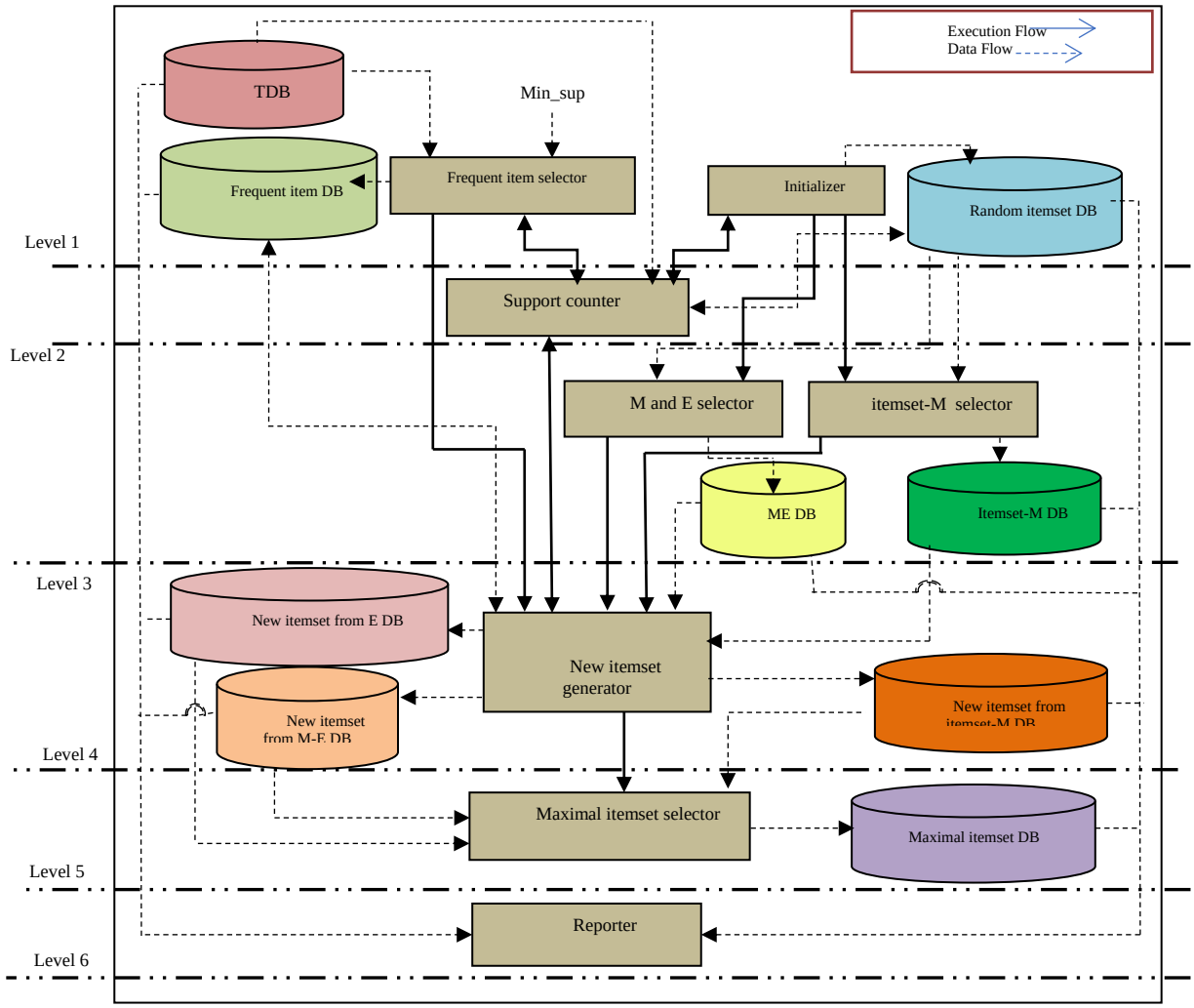

Figure 2: Presents the architecture of the proposed miner

Initializer module (level 1) The task of this module is generating the initial population. Each individual of the population is an itemset of length k, i.e. k-itemset. Initially, the module constructs a table named "random itemsets DB". The table consists of the following columns: N (population size), K (the length of itemset), Itemset (holds the itemset itself), and Sup (represents the support of the itemset). Initializer module generates N numbers randomly, each of which represents the length of itemset. Moreover, for each K, the initializer will generate a random number K-times, which represents the itemset. After that, it will invoke support counter module in order to count itemset support (see figures 3 and 4).

```
BEGIN
  loop
   rnum:=round (dbms_random.VALUE(min_value,max_value));
   SELECT count(*) INTO c FROM temp_itemset WHERE itemset1=rnum AND item=len_item;
   exit WHEN c=0;
  END loop;
END;
```

Figure 3 Abstracted PL/SQL code to generate K.

The generated itemset is unordered and in form of one length itemset. To order it and making it of K-length itemset; a view is created and the order by Clause is used. The items of the generated itemset are ascendingly ordered and connected by ";" symbol. After all itemsets have been generated, support counter module is invoked in order to calculate the support of each itemset and store the result in "random itemset DB" table.

```
BEGIN
 loop
  rnum:=ROUND (DBMS_RANDOM.VALUE(min_value,max_value));
  SELECT count(*) INTO c FROM initpop WHERE k=rnum;
  -- check the uniqueness of K
  exit WHEN c=0; -- if not found insert it in the table
 END loop; -- if rnum previously found in the table
END;
```

Figure 4 Abstracted PL/SQL code to generate the itemset

Support counter module (level 2). The task of this module is calculating the support of the itemset that is generated by initializer module. The support of 1-length item is gotten from frequent table and there is no need to scan the database to ch reduces time cost. If itemset's length is k-itemset, then its support will be calculated through count of tidset resulted from intersection among sets of tidset of each item that forming the itemset in TDB. After counting the tidsets resulted from intersection operation, random itemset DB will be updated and the resulted number will be inserted in sup column (consider figure 5).

The longer (M) and longest (E) Selector Module (level 3) The job of this module is selecting M and E random numbers from N population size. They represent itemsets that are chosen according to fitness criterion such that M < N and E < M. M represents the higher fitness itemset where E is the highest fitness itemset. Initially, M and E selector module constructs a table named "ME DB". The table consists of M_selected and Elite attributes. Then the module selects E and (M-E) and stores them in "ME DB" table (see figure 6).

```
IF item_record.item=1 THEN --if item length=1
    V_ITEM:=1;
   SELECT sub into v_sup FROM frequent WHERE
    freq_item=(SELECT itemset1 FROM    temp_itemset
               WHERE item=1);
ELSE   ------ if item length is more than 1
  SELECT  COUNT(tidset),item_record.item
     INTO   v_sup,v_item FROM
     (SELECT  tidset FROM  trans WHERE item IN
        (SELECT itemset1 FROM temp_itemset   WHERE
          item=item_record.item)GROUP BY  tidset
      HAVING COUNT (DISTINCT item)= (SELECT
        COUNT(*) FROM
           temp_itemset WHERE
           item=item_record.item));
END IF;
```

Figure 5: Abstracted PL/SQL code of support generation

```
Mnum:= ROUND (DBMS_RANDOM.VALUE(2,lcount));
   insert into t(m_selected) SELECT * FROM
      (select itemset from initpop
       where sup is not null group by itemset
       order by max(length(itemset)) desc);
   Enum:=ROUND (DBMS_RANDOM.VALUE(1,Mnum-1));
   insert into me(m_selected,elite)
    (SELECT LEAD (m_selected, Enum) OVER (ORDER BY  LPAD
    (m_selected,100) desc)     m_selected,
      CASE
      WHEN m_selected< NTH_VALUE (MAX (m_selected),Enum)
      OVER (ORDER BY  LPAD (m_selected, 100) desc) THEN NULL
       ELSE m_selected
       END
             Elite FROM t GROUP BY m_selected);
```

Figure 6. Abstracted PL/SQL code for selecting M and E

Itemset−M Selector Module (level 3). This module is in charge of selecting the remaining itemsets after selecting M better itemsets and Elite itemsets from random itemset DB. These itemsets are referred to as"itemset-M" itemsets. Initially, Itemset-M selector module constructs a table named "itemset_M" to carry the selected itemsets. The table consists of two columns, which are ID1 and Itemset_M. Then it selects the remaining itemsets from random itemset DB that are frequent and neither Elite nor M_selected (not within the itemsets in ME table) (see figure7).

```
insert into itemset_m
 select rownum id1,  itemset
 from (select * from initpop
       where sup>=min_sup
       AND ITEMSET NOT IN (
SELECT M_SELECTED FROM ME WHERE m_selected IS NOT NULL ) AND
ITEMSET NOT IN (
SELECT ELITE FROM ME WHERE elite IS NOT NULL)
```

Figure 7. Abstracted PL/SQL code for selecting itemset-M.

New Itemset Generator (level 4). this module is responsible for generating new itemset. Initially, the module constructs number of table to generate new itemsets resulted from union operation of Elite, M_selected, and itemset-M with frequent items in frequent item DB.The first table is "me_result" table, which consists of the following columns: ID1, ELITE, FREQ_ITEM, COMBINED_STR, and SUP. The generated itemsets are saved in new_gen_item table, which represents the population of new generation (see figure 8).

```
insert into new_gen_item
  select COMBINED_STR,SUP,REGEXP_COUNT(combined_str,';')
  from (
  SELECT id1,COMBINED_STR,SUP,REGEXP_COUNT(combined_str,';')
       from (
              SELECT id1,ELITE,FREQ_ITEM,COMBINED_STR,SUP,
              ROW_NUMBER() over (PARTITION BY ELITE order by id1) RN FROM ME_RESULT
              WHERE ( EITE,REGEXP_COUNT(combined_str,';')
                    ) IN
                         (
        SELECT ELITE,MAX(REGEXP_COUNT(combined_str,';')
                         )
                          FROM me_result where sup>=min_sup GROUP BY ELITE
            ) and sup>=min_sup
    ) where rn=1
order by id1
```

Figure 8. Abstracted PL/SQL code for generating new itemset (elite union freq_item)

Maximal Itemset Selector (level 5). This module is responsible for selecting the best new generated itemsets from new_gen_item, new_gen_item2, and new_gen_item3 tables as candidate maximal itemset.

```
Loop
   ,item_counts as
               (
                 SELECT DISTINCT a.item
                ,COUNT(DISTINCT a.tidset) over() total_tidset
                ,COUNT(a.item) over(PARTITION BY a.item)item_occurrence_count
                FROM (
                       SELECT DISTINCT item ,tidset
                       FROM all_items
                     ) a
               )
  SELECT listagg(item, ';') within GROUP(ORDER BY to_number(item))common_items
  into cc FROM item_counts
  WHERE total_tidset = item_occurrence_count;
End loop;
```

Figure 9: Saving Elite maximal candidates in maximal table

The module constructs a table named “tidtid” table to hold tidsets of each item that form an itemset. The table has the following column: ID1, TIDSET, and GROUP_ALL. Figure.9 depicts the skeleton of this module through very abstracted PL/SQL code.

The Reporter Module (level 6). The reporter module shows Maximal itemsets (MI)s of each generation. Each generation displays, the contents of TDB, random itemset DB, frequent item DB, ME DB, itemset-M DB, new generated itemset from Elite U half no. of freq_item DB, new generated itemset from M-E U quarter no. of freq_item DB, new generated itemset from itemset-M U freq_item DB, maximal itemset DB. The reporter module exhibits for each item the set of tidsets that support this item

in TDB. For random itemset DB, it exhibits population size N, the length of each randomly generated itemset K, k-itemset itemsets, and the support of each k-itemset sup. For frequent item DB, it exhibits each itemset item with its support sup and length Len. For ME DB, it displays number of best M selected itemsets m_selected and number of best E selected itemsets Elite. For new generated itemset from Elite U half no. of freq_item DB, it displays Elite, half no. of freq_item,combined_str which represents new generated itemset, and support sup. For new generated itemset from itemset-M U freq_item DB, it displays itemset-M, freq_item,combined_str which represents new generated itemset, and support sup. For maximal DB, it exhibits new generated itemsets with their tidsets that support them.

5. Experiments

Very low levels of Min-Sup values were adopted to test the speed and scalability of the proposed implementation, PL/SQL-MMIBA, and prove its ability to execute in spite of these values of Min-Sup. The experiments indirectly pointed to the memory utilization according to the used database and Min-Sup values. It is well known that the lower value of Min-Sup leads to largest number of FIs and MIs in addition to the increment of the execution time and memory consumption.

There are many real databases used for the experiments such as (Chess, Mushroom, Cancer Cells, Census Data, and Dense Census) in addition to synthetic databases. The following tables present a comparison among PL/SQ-MMIBA, Loosely -coupled implementation of MMIBA (LC-MMIBA), and MAFIA. As it is explained previously, MMIBA starts the execution with random values for its population, therefore its execution time is affected by the closeness or farness of these values from actual results to be mined, i.e., MIs. Therefore, to get fair comparisons, LC-MMIBA and PL/SQL-MMIBA were executed 10 times for each Min-Sup value for a specific database and the average of these executions has been calculated and presented in the tables of result section.

6. Results and Discussion

Table (1) presents the execution times of the MAFIA, LC-MMIBA and PL/SQL-MMIBA using Chess DB and different values of Min-Sup ranged from 1% to 10% of the database size. It is obvious that MAFIA outperforms LC-MMIBA when the values of Min-Sup ranged from 1% to 7.5%. In addition, there is unexpected execution time for LC-MMIBA when the Min-Sup value equals 6%. Indeed, one of the ten executions of LC-MMIBA for Min-Sup=6% was considerably higher than the rest nine executions which affects the value of the presented average, i.e., 50:20. PL/SQL-MMIBA outperforms MAFIA and LC-MMIBA in all aspects of the Min-Sup and its execution time has regular decrement according to the increased value of Min-Sup.

Table 1

Execution time of MAFIA, Loosely-Coupled-MMIBA, and PL/SQL-MMIBA using Chess DB with various MIN-Sup values.

	MIN-SUP VALUES											
Alg.	1%	2%	2.5%	3%	4%	5%	6%	7%	7.5%	8%	9%	10%
MAFIA	78:02	60:55	50:00	40:57	40:21	40:00	40:00	40:00	40:00	30:45	30:42	30:42
LC-MMIBA	100:01	90:03	90:00	90:00	60:23	50:15	50:20	40:27	40:02	30:34	30:02	20:55
PL/SQL-MMIBA	64:33	50:55	47:58	40:30	40:07	40:00	30:25	20:18	20:12	20:06	20:00	10:50

Table (2) presents the execution times of the MAFIA, LC-MMIBA and PL/SQL-MMIBA using Mushroom DB and different values of Min-Sup ranged from 1% to 10% of the database size. MAFIA does better than LC-MMIBA when the values of Min-Sup ranged from 1% to 4%. The execution time of MAFIA for many values of Min-Sup was not changed, for example consider the execution time when Min-Sup values are 7.5%, 8%, and 9%. This case can be observed for LC-MMIBA for some values of Min-Sup. PL/SQL-MMIBA outperforms MAFIA and LC-MMIBA for all values of Min-Sup. As a

general observation, the nature of the database has imposed a slight and smooth decline for the execution time values when the values of Min-Sup increase, this observation can be considered for the three algorithms.

Table 2

Execution time of MAFIA, Loosely-Coupled-MMIBA, and PL/SQL-MMIBA using **Mushroom** DB with various MIN-Sup values.

	MIN-SUP VALUES											
Alg.	1%	2%	2.5%	3%	4%	5%	6%	7%	7.5%	8%	9%	10%
MAFIA	100:02	100:05	90:00	60:57	60:05	50:13	50:11	50:11	50:00	50:00	50:00	40:42
LC-MMIBA	110:15	110:11	110:00	90:00	70:01	40:43	40:30	40:27	30:46	30:37	30:30	30:30
PL/SQL-MMIBA	80:03	80:22	50:47	40:30	40:22	40:00	30:25	20:18	20:12	20:06	20:01	10:50

Table (3) presents the execution times of the MAFIA, LC-MMIBA and PL/SQL-MMIBA using Cancer DB and different values of Min-Sup ranged from 1% to 10% of the database size. Cancer DB is dense database, which may justify the outperformance of LC-MMIBA and PL/SQL-MMIBA over the MAFIA algorithm. One can see that a considerable difference in the amount execution time between MMIBA in its two implementations from one side and MAFIA from another side.

Table 3

Execution time of MAFIA, Loosely-Coupled-MMIBA, PL/SQL-MMIBA using Cancer Cells DB with various MIN-Sup values.

	MIN-SUP VALUES											
Alg.	1%	2%	2.5%	3%	4%	5%	6%	7%	7.5%	8%	9%	10%
MAFIA	170:13	160:09	150:00	130:44	130:13	120:11	110:21	100:03	90:23	90:15	90:10	90:10
LC-MMIBA	170:01	150:13	140:10	130:09	120:00	110:00	100:03	80:01	80:01	70:25	60:50	60:00
PL/SQL-MMIBA	150:38	150:02	130:21	110:00	110:00	90:17	70:47	70:00	70:00	60:25	50:05	40:13

Tables (4) and (5) show the results using Census and Dense Census DB, they are dense database, but the latter is very dense in comparison with the former one. Some fields of the tables contain '*' which indicates an "insufficient memory space" error. These cases appeared in MAFIA and LC-MMIBA implementation. This problem did not emerge with PL/SQL-MMIBA depending on Census and Dense Census DBs.

Table 4

Execution time of MAFIA, Loosely-Coupled-MMIBA, and PL/SQL-MMIBA using Census DB with various MIN-Sup values.

	MIN-SUP VALUES											
Alg.	1%	2%	2.5%	3%	4%	5%	6%	7%	7.5%	8%	9%	10%
MAFIA	*	220:24	200:00	190:37	190:13	190:00	160:12	140:00	110:15	110:00	110:00	100:39
LC-MMIBA	260:03	210:50	180:03	180:02	170:05	160:45	120:45	100:32	90:09	90:35	80:03	70:30
PL/SQL-MMIBA	270:17	190:55	170:00	180:02	170:20	150:04	100:20	100:00	80:03	70:13	70:00	60:14

Table 5

Memory utilization of MAFIA, Loosely-Coupled-MMIBA, and PL/SQL-MMIBA using **Dense Census** DB with various MIN-Sup values.

	MIN-SUP VALUES											
Alg.	1%	2%	2.5%	3%	4%	5%	6%	7%	7.5%	8%	9%	10%
MAFIA	*	310:09	290:00	270:56	270:43	270:29	250:02	220:21	200:12	190:10	180:00	160:19
LC-MMIBA	*	320:02	250:39	260:00	220:20	200:16	190:07	180:00	170:08	160:00	160:00	140:45
PL/SQL-MMIBA	340:05	310:17	270:03	250:10	200:21	200:37	180:55	170:32	160:02	140:09	130:00	110:37

7. Conclusions

This research presents a new implementation of MMIBA using Oracle 11g PL\SQL. The aim of this implementation is to combine the mining system with the data to be mined. This approach maintains scalability to large DBs and data integrity with DBMS. In contrast with traditional mining approaches which use flat files, depend on limited facilities of programming languages, make extensive use of main memory and I/O traffics, embedded miner in DBMS can treat dataset larger than main memory. In addition, it avoids I/O traffics because all DBS, tables, programming language and other structures are in one environment (Oracle DBMS). This approach overcomes the loosely-coupled implementation in considerable amount of execution time and memory consumption. The proposed system was tested using many real and synthetic databases with wide range of properties including size, number of items, database sparseness or its density. Many values for minimum support and conflict were used in these tests to prove the robustness of the designed system.

ORACLE DBMS is preferred to implement the maximal itemsets embedded miner since it has the many privileged features listed previously. According to the implementation of MMIBA algorithm, several new features have made a difference in implementation effort and time. Such features provided in Oracle 11g PL/SQL don't exist in other programming languages. For example ORACLE's regular expression support comprised five functions (REGEXP_LIKE, REGEXP_SUBSTR, REGEXP_INSTR, REGEXP_REPLACE and REGEXP_COUNT). In addition, the availability of suitable data type for data mining such as CLOB to hold k-itemsets, plays important roles in supporting the efficiency of the proposed miner. We believe that such facilities in DBMSs make big difference in term of efficiency scalability, data integrity, execution time, memory consuming and security.

The practical significance of obtained results is that the implementation of the miners as stored procedures with the data to be mined produced significant results such as the scalability, robust memory management, and efficient utilization of computation resources which leads to increasing the throughputs. Such implementation is associated with correct selection of an algorithm and DBMS. Prospects for further research are to study the ability of implementing more algorithms in different data mining tasks.

8. Acknowledgements

The work is supported by Al-Rafidain University College.

9. References

[1] Mohamed A. Gawwad, Mona F. Ahmed et al (2017): Frequent Itemset Mining for Big Data Using Greatest Common Divisor Technique. Data Science Journal 16(25), PP. 1-10.

[2] Nicolas D., Mohamed Q. (2016), Frequent Itemset Border Approximation by Dualization. In: A. Hameurlain et al. (Eds.) TLDKS XXVI, LNCS, 9670, pp. 32-60. Springer, Verlag Berlin Heidelberg.

[3] Haifeng L., Yuejin Z. et al (2016), A Heuristic Rule Based Approximate Frequent Itemset Mining Algorithm. Procedia Computer Science 91, 324-333, (2016).
[4] C. Ganesh, B. Sathiyabhama et al (2016), Fast Frequent Pattern Mining Using Vertical Data Format for Knowledge Discovery. International journal of emerging research in management & Technology 5(5), pp. 141-148.
[5] Philippe F., Jerry C. et al (2017), Survey of Itemset Mining. Wiley, https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1207.
[6] Mais A. Mohammed, Hussein Al-Khafaji (2017): Maximal Itemsets Mining Algorithm Based On Bees' Algorithm. IEEEXPLORE, https://ieeexplore.ieee.org/document/7976130.
[7] Yunuscan K., Tansel O., et al, (2016), Utilizing Maximal Frequent Itemsets and Social Network Analysis for HIV Data Analysis. Journal of Cheminformatics 8(71), pp. 1-15.
[8] Zhengxin C. 2015, Towards Integrated Study of Data Management and Data Mining. (ITQM 2015) Procedia Computer Science 55 (2015) 1331 – 1339.
[9] Beibei Z., Xuesong M 2006, Data mining using Relational Database Management Systems. W.K. Ng, M. Kitsuregawa, and J. Li (Eds.): PAKDD 2006, LNAI 3918, pp. 657–667, Springer-Verlag Berlin Heidelberg.
[10] Lu H. (2001) Seamless Integration of Data Mining with DBMS and Applications. In: Cheung D., Williams G.J., Li Q. (eds) Advances in Knowledge Discovery and Data Mining. PAKDD 2001. Lecture Notes in Computer Science, vol. 2035. Springer, Berlin, Heidelberg.
[11] M. S. Sousa, M. L. Q. Mattoso et al: Data mining: a database perspective. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.8188&rep=rep1&type=pdf.
[12] Amir N., Surajit C. (2000), Integration of Data Mining and Relational Databases. Proceedings of the 26th International Conference on Very Large Databases, pp. 719-722 Cairo, Egypt.
[13] Tadeusz M., Marek W. et al (2000): Data Mining Support in Database Management Systems. In : 2nd International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2000) Proceedings, Pages 382-392, Springer-Verlag London, UK.