International Journal of Electrical and Computer Engineering (IJECE)
Vol. 9, No. 6, December 2019, pp. 5471~5478
ISSN: 2088-8708, DOI: 10.11591/ijece.v9i6.pp5471-5478

Frequent itemset mining: technique to improve eclat based algorithm

Mahadi Man, Masita Abdul Jalil
School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu, Malaysia

Article Info

Article history:

Received Dec 12, 2018
Revised Apr 18, 2019
Accepted Jun 12, 2019

Keywords:

Association rule mining
Data mining
Eclat based algorithm
Frequent itemset mining

ABSTRACT

In frequent itemset mining, the main challenge is to discover relationships between data in a transactional database or relational database. Various algorithms have been introduced to process frequent itemset. Eclat based algorithms are one of the prominent algorithm used for frequent itemset mining. Various researches have been conducted based on Eclat based algorithm such as Tidset, dEclat, Sortdiffset and Postdiffset. The algorithm has been improvised along the time. However, the utilization of physical memory and processing time become the main problem in this process. This paper reviews and presents a comparison of various Eclat based algorithms for frequent itemset mining and propose an enhancement technique of Eclat based algorithm to reduce processing time and memory usage. The experimental result shows some improvement in processing time and memory utilization in frequent itemset mining.

Corresponding Author:

Mahadi Man,
School of Informatics and applied Mathematics,
Universiti Malaysia Terengganu,
21030 Kuala Terenganu, Terengganu, Malaysia.
Email: hadie02@yahoo.com

1. INTRODUCTION

Nowadays, there is a numerous number of transaction recorded in any organization. In order to process this data and find useful information, data mining is used [1]. Data mining is the process of extracting useful information from various sources by extracting information from a data set and transform it into an understandable structure for further use [2, 3]. Association Rule Mining (ARM) is one of the most prominent areas in detecting the interesting pattern. With the aims to extract interesting correlations, frequent patterns, association or casual structures among a set of items in the transaction databases or other data repositories [4]. ARM must be emphasized to find out the association rules that satisfy the predefined minimum support and confidence from a given database [5]. Association rule can create analyzing data for frequent pattern using the criteria Support and Confidence. Support is indicating of how frequently the item appears in the database. Confidence indicates the number of time has been found [6].

The concept of frequent itemset was first introduced for mining transaction databases [7]. Let I = {i1, i2, ..., in} be a set of all items. A k-itemset α, which consists of k items from I, is frequent if α occurs in a transaction database D no lower than θ|D| times, where θ is a user-specified minimum support threshold (called min_sup in our text), and |D| is the total number of transactions in D [8]. The task of finding the frequent pattern in large databases is very essential and has been studied on a huge scale in the past few years. Unfortunately, it is computationally expensive, especially when a huge number of patterns exist [9].

Eclat was introduced by Zaki *et al.* in 1997. Eclat is an acronym for Equivalence Class Clustering and bottom-up Lattice Traversal [10]. Referring to W.A.B.W.A. (2016), Eclat uses prefix-based equivalence relation, θ_1 along with bottom-up search. It enumerates all frequent itemsets. There are two main steps: candidate generation and pruning.

In candidate generation, each k-itemset candidate is generated from two frequent (k-1)-itemsets and its support is counted, if its support is lower than the threshold, then it will be discarded, otherwise, it is frequent itemsets and used to generate (k+1)-itemsets [4]. The first scan of the database builds the transaction id (tids) of each single items. Starting with a single item (k = 1), then the frequent (k + 1)-itemset will grow from the previous k-itemset will be generated with a depth-first computation order. The computation is done by intersecting tids of the frequent k-itemsets to compute the tidsets of the corresponding (k+1)-itemsets. The process is repeated until no more frequent candidate itemsets can be found [4].

Eclat starts with prefix {} and the search tree is actually the initial search tree. To divide the initial search tree, it picks the prefix {a}, generate the corresponding equivalence class and does frequent itemset mining in the sub tree of all itemsets containing {a}, in this subtree it divides further into two sub trees by picking the prefix {ab}: the first subtree consists of all itemsets containing {ab}, the other consists of all itemsets containing {a} but not {b}, and this process is recursive until all itemsets in the initial search tree are visited. The search tree of an item base {a, b, c, d, e} is represented by the tree as shown in Figure 1. Figure 2 illustrates how data in a horizontal layout is transformed by a set of transaction ids or tidset in vertical layout [4].

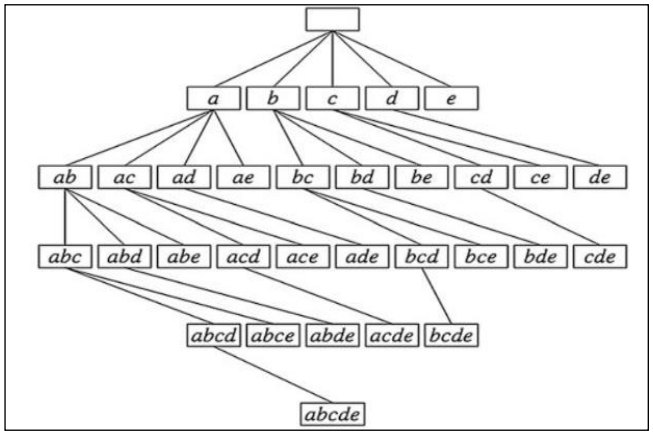


Figure 1. Prefix tree for 5 items {a, b, c, d, e}

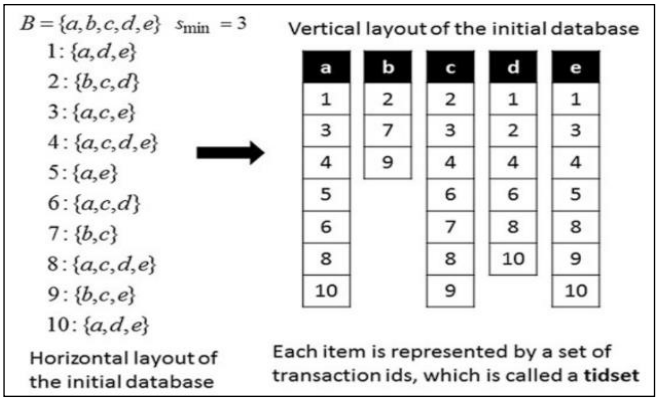


Figure 2. Transformation from horizontal to vertical layout

This paper starts with a formal introduction of Eclat based algorithms. Moving forward, this paper presented related works in section 2. Then it will continue with a comparative analysis based on the advantages and disadvantages of each algorithm in section 3.Section 4 discussed the enhancement of the Incremental Eclat algorithm. In section 5, this paper comes out with experimental results between previous works and the proposed enhancement and to conclude the discussion, there will be a conclusion and future works in section 6.

2. RELATED WORKS

Finding frequent itemset is a time-consuming process, many versions of frequent itemset mining algorithms have been proposed by many researchers that aim at reducing the time and space complexities [11].

2.1. Traditional eclat(tidset) algorithm

Tidset algorithm uses vertical dataset and a bottom-up approach for searching items in a database [2]. Figure 3 depicted the pseudocode for Tidset algorithm.

```
INPUT: A file 𝒟 consisting of baskets of items, a support threshold σ, and an item prefix I, such that I ⊆ 𝒥.
OUTPUT: A list of itemsets ℱ[I](𝒟, σ) for the specified prefix.
METHOD:
 ℱ[i] ← {}
for all i ∈ 𝒥 occurring in 𝒟 do
     ℱ[I] := ℱ[I] ∪ {I ∪ {i}}
#  Create 𝒟_i
     𝒟_i ← {}
     for all j ∈ 𝒥 occurring in 𝒟 such that j > i do
          C ← cover({i}) ∩ cover({j})
          if |C| ≥ σ then
               𝒟_i ← 𝒟_i ∪ {j, C}
# Depth-first recursion
     Compute ℱ[I ∪ i](𝒟_i, σ)
     ℱ[I] := ℱ[I] ∪ ℱ[I ∪ i]
```

Figure 3. Pseudocode for tidset algorithm

2.2. dEclat algorithm

dEclat performs a depth-first search of the subset tree. Zaki and Gouda (2003) show that diffset allow it to mine on much lower supports than the base Eclat method. The input to the procedure is a set of class members for a subtree rooted at P frequent itemset are generated by computing diffset for all distinct pairs of itemsets and checking the support of the resulting itemsets. A recursive procedure call is made with those itemsets found to be frequent at the current level. This process is repeated until all frequent itemsets have been enumerated [12]. Figure 4 depicted the pseudocode for dEclat algorithm.

```
INPUT: A file 𝒟 consisting of baskets of items, a support
Threshold σ, and an item prefix I, such that I ⊆ 𝒥
OUTPUT: A list of itemsets ℱ[I](𝒟, σ) for specified prefix
METHOD:
Start ( [P] )
For all X_i ∈ [P] do
R=X_i ∪ X_j;
d(R)=d(X_j) – d(X_i);
if σ(R ) ≥ min_sup then
T_i=T_i ∪ {R};//T_i initially empty
if T_i then start (T_i)
```

Figure 4. Pseudocode for dEclat algorithm

2.3. SortDiffset algorithm

SortDiffset was created by Trieu and Kunieda in 2014. In general, diffset in an equivalence class should be sorted in descending order according to size to generate new itemsets represented by diffset with smaller sizes [13]. Figure 5 depicted the pseudocode for SortDiffset algorithm.

```
1: INPUT: A file 𝒟 consisting of baskets of items, a support
   Threshold σ, and an item prefix I, such that I ⊆ 𝒥
2: OUTPUT: A list of itemsets ℱ[I](𝒟, σ) for specified prefix
3: METHOD:
4: Start ( [P] )
5: sort data by Itemset with descending order of dataset value
6: For all X_i ∈ [P] do
7:  get interest data for column[i] with column[i+1];
8:  save to db
9: add next transection data;
```

Figure 5. Pseudocode for sortdiffset algorithm

2.4. Postdiffset algorithm

The initial objective of Postdiffset is to handle the issues of big and dynamic data. Postdiffset tends to reduce a memory and spaces requirement by implementing flushing of memory prior to each itemset being visited before intersecting the next itemsets. Figure 6 depicted the pseudocode for Postdiffset algorithm.

```
1:  INPUT: vertical tid-list database
2:  OUTPUT:
3:  METHOD:
4:  Start
5: Sort data by Itemset
6: data by itemsets with descending order of dataset value.
7: looping = numberofcolumn;
8:  For (i=0; i<looping; i++)
9:  get intersect data for column[i] with column [i + 1]
10: get diffset data for column[i] with column[i+1];
11: save to DB
12: add next transaction data;
13: flush value for current/last transaction data;
14: End For
28: End For
```

Figure 6. Pseudocode for postdiffset algorithm

Compared with Postdiffset and Tidset algorithm, Tidset algorithm is more prone to memory scarcity. The situation becomes worse if the mining process involved a bigger dataset. As memory utilization increased, the machine's dependency on virtual memory increases. The more a computer has to depend on virtual memory, the less efficiently that machine will run [14]. As a result, the performance of the computer is compromised. Hence to prevent the problem of memory scarcity, Postdiffset included an additional step to flush the memory.

3. COMPARATIVE ECLAT BASED ALGORITHM

In summary, the review and analysis of Eclat based algorithms are presented in Table 1. From the comparative analysis, it shows that the Eclat based algorithm is challenged by memory utilization and processing time (depends on the size of data used). Eclat algorithms need more memory to process the data. Moving on to dEclat, even though its drastically cut down the size of memory required, but degrades with a sparse database. The introduction of Sortdiffset has reduced running time and memory usage but its cost to do the sorting. Finally, Postdiffset was introduced with the ability to reduced memory utilization. However, Postdiffset has disadvantage on the processing time that needs to be improved in the future works.

Table 1. Eclat based algorithm comparison

Approach	Techniques	Advantages	Disadvantages
Eclat [15, 16]	Vertical intersection of tidlist	The size of Tidsets represents support	Difficult in pruning technique. The longer tidset, more time and memory needed
dEclat [12]	Only keeps track of differences in tidsets.	Drastically cut down the size of memory required to store intermediate results	Suitable for a dense database but degrades with sparse database Need to switch between tidset and diffset for a sparse database.
Sortdiffset [17]	Combine Tidset + diffset, then sort tidset in ascending order and diffset in descending order	- No need for switching condition - Reduce running time and memory usage	Cost to do sorting
Postdiffset [4]	Perform tidset for first level looping and diffset for second level looping + Introducing flushing of memory for itemset being visited.	Better performance	Reduced memory utilization. However, suffer from processing time.

4. IMPROVEMENT FOR INCREMENTAL ECLAT ALGORITHM

Previous works use a Relational Database Management System (RDBMS) for its transaction process to store transaction data gained in the data mining process. RDBMS is one that presents information in tables with rows and columns. A table is referred to as a relation in the sense that it is a collection of objects of the same type (rows). Data in a table can be related according to common keys or concepts, and the ability to retrieve related data from a table is the basis for the term relational database [18]. There is two types of SQL statement can be used to store data into the database using a single row INSERT or LOAD DATA INFILE. According to (Benjamin Morel, 2017) LOAD DATA INFILE is the preferred solution when looking for raw performance on a single connection [19]. It requires to prepare a properly formatted file first before the LOAD DATA INFILE can be executed.

In current algorithms, transaction data is stored in the database using a single row INSERT. Therefore, the statement needs to be executed multiple times based on the n-transaction of data available. Figure 7 shows the algorithm for single row INSERT. To improve on the single row INSERT into the database system, LOAD DATA INFILE statement has been used. By using INFILE LOAD statement, only one statement is executed to insert data into the database system. Since only one statement executed, it minimized the used of memory for data transaction to the database. Figure 8 shows how the improved algorithm is executed.

```
For (i=0; i<looping; i++)
get frequent itemset
save data to DB//using INSERT INTO Statement
add next transaction data
End For
```

Figure 7. Pseudocode for a single row insert into database

```
For (i=0; i<looping; i++)
get frequent itemset
save data to Flat File
add next transaction data
End For
Export from Flat File into DB using INFILE LOAD
```

Figure 8. Pseudocode for multiple rows insert into database

5. EXPERIMENT

5.1. Application platform

The experiment is performed on a Dell LATITUDE 7280, Intel ® Core(TM) i7-6600U CPU 2.60 GHz with 8 GB RAM in a Win 10 64-bit platform. The software used is MySQL version 10.1.34-MariaDB database server, Apache/2.4.17 (Win32) OpenSSL/1.0.2d PHP/5.6.21as a web server. Benchmark datasets have been retrieved from http://fimi.ua.ac.be/data/ and the being imported into the MySQL. For the initial test, we start with a simple dataset in order to produce faster results. Chess and Mushroom datasets are divided into several benchmark trained datasets.

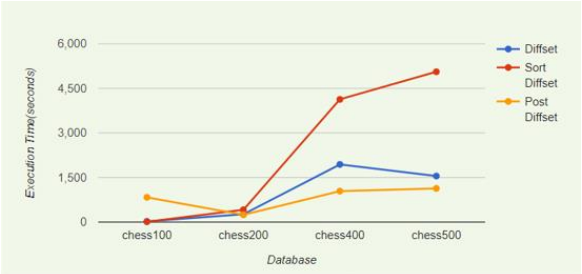
5.2. Experimental result

The experiment has been carried out for dEclat (Diffset) algorithm, SortDiffset algorithm and Postdiffset algorithm. We focused on the performance of processing time and memory usage between old algorithms and the proposed algorithm as shown in Table 2.

Table 2. Database characteristic

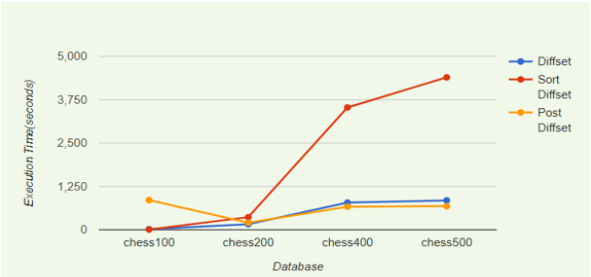
Dataset	Size (KB)	Length (Attribute)	Records
Chess100	7.48	12	100
Chess200	14.85	12	200
Chess400	202.5	12	400
Chess500	202.5	12	500
Mushroom100	213	12	100
Mushroom200	213	12	200
Mushroom400	256	12	400
Mushroom500	256	12	500

Figures 9 and 10 demonstrate the running time between the old algorithm and the new proposed algorithm using chess datasets at a support threshold of 10%. For all three algorithms, both running time is increasing in line with the number of records. The new proposed algorithm shows a better performance in processing time. By average, Diffset reduced by 42.48%, SortDiffset by 15.88% and Postdiffset by 22.3%. On the memory usage, the usage is reduced by 3.66% for Diffset,1.26% for SortDiffset and 3.5% for Postdiffset. Overall result shows an improvement for all algorithm.



Database	Execution time (seconds)			Memory Usage (MB)		
	Diffset	Sort Diffset	Post Diffset	Diffset	Sort Diffset	Post Diffset
chess100	24.36	11.13	832.51	3.57	3.59	3.59
chess200	263.27	414.32	244.11	3.8	3.99	3.77
chess400	1939.65	4127.11	1041.37	4.19	4.33	4.15
chess500	1548.5	5057.06	1131.63	4.38	4.62	4.33

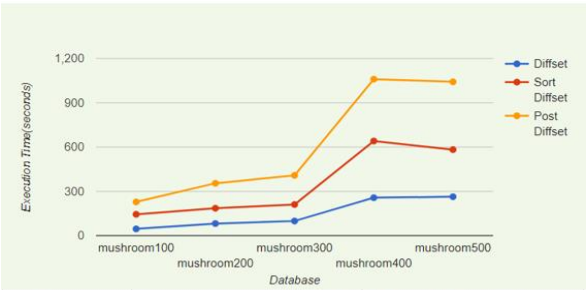
Figure 9. Experimental result for old algorithm using chess datasets



Database	Execution time (seconds)			Memory Usage (MB)		
	Diffset	Sort Diffset	Post Diffset	Diffset	Sort Diffset	Post Diffset
chess100	17.91	8.52	856.24	3.43	3.29	3.45
chess200	161.4	363.09	204.92	3.65	3.85	3.63
chess400	782.93	3525.78	667.81	4.04	4.49	4.01
chess500	846.84	4393.06	680.66	4.24	4.77	4.2

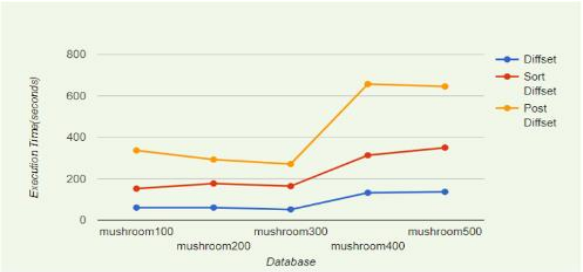
Figure 10. Experimental result for proposed algorithm using chess dataset

Figures 11 and 12 presented the experimental using mushroom datasets at 10% support threshold. Comparing with the old algorithm, on average, the proposed algorithm reduced the execution time by 22.23% for Diffset, 22.54% for SortDiffset and 11.59% for Postdiffset. On memory usage, the new algorithm shows good performance by reducing 3.65% for Diffset, 2.85% for SortDiffset and 3.51% for Postdiffset.



Database	Execution time (seconds)			Memory Usage (MB)		
	Diffset	Sort Diffset	Post Diffset	Diffset	Sort Diffset	Post Diffset
mushroom100	45.84	144.69	228.67	3.59	3.75	3.62
mushroom200	82.05	185.66	354.7	3.79	4.08	3.82
mushroom300	99.1	211.1	408.2	3.98	4.41	4.01
mushroom400	257.12	641.06	1060.12	4.18	4.74	4.21
mushroom500	263.58	582.88	1042.22	4.37	5.06	4.39

Figure 11. Experimental result for old algorithm using mushroom datasets



Database	Execution time (seconds)			Memory Usage (MB)		
	Diffset	Sort Diffset	Post Diffset	Diffset	Sort Diffset	Post Diffset
mushroom100	60.99	152.53	336.66	3.45	3.61	3.48
mushroom200	61.14	177.14	292.72	3.65	3.95	3.68
mushroom300	52.12	164.59	271.14	3.84	4.28	3.87
mushroom400	132.55	313.33	656.96	4.03	4.63	4.07
mushroom500	136.98	350.41	645.41	4.22	4.95	4.25

Figure 12. Experimental result for proposed algorithm using mushroom dataset

6. CONCLUTION AND FUTURE WORKS

Our proposed technique shows some improvement in reducing memory usage and processing time. By reducing the number of transaction to the MySQL database, memory usage is reduced. It also affects the processing time. From the experimental result, the occurrences of the itemset would be one of the contributing factors of the algorithm performance. For future work, we can also deploy this technique for sparse datasets such as T10I4D100K or retail. Looking at the same domain, the technique would be implemented to mining infrequent itemset. Hence, we could discover either the result would be the same as frequent itemset or distinguished from the findings.

REFERENCES

[1] K. Maniktala, *et al.*, "Technique to Enhance Map Reduce ECLAT Algorithm," *Int. J. Technol. Comput.*, vol. 2, pp. 547-548, 2016.
[2] M. Kaur, *et al.*, "Advanced Eclat Algorithm for Frequent Itemsets Generation," *Int. J. Appl. Eng. Res.*, vol. 10, pp. 23263-23279, 2015.
[3] V. Shine, "Dataset Preparation in Datamining Analysis Using Horizontal Aggregations," *Int. J. Eng. Res. Appl.*, vol. 4, pp. 369-372, 2014.
[4] W. A. B. W. A. Bakar, *et al.*, "Advances in Machine Learning and Signal Processing," vol. 387, pp. 35-46, 2016.
[5] A. Meenakshi, "Survey of Frequent Pattern Mining Algorithms in Horizontal and Vertical Data Layouts," *Int. J. Adv. Comput. Sci. Technol.*, vol. 4, pp. 48-58, 2015.
[6] R. Ishita and A. Rathod, "Eclat with Large Database Parallel Algorithm and Improve its Efficiency," *Int. J. Comput. Appl.*, vol. 143, pp. 33-37, 2016.
[7] R. Agrawal, *et al.*, *"Mining Association Rules between Sets of Items in Large Databases,"* Proceedings of the 1993ACM-SIGMOD international conference on management of data (SIGMOD'93)), Washington, DC, pp. 207-216, 1993.
[8] J. Han, *et al.*, *"Frequent pattern mining : current status and future directions,"* Springer Sci. Media, LLC 2007, pp. 55-86, 2007.
[9] R. Krupali and D. Garg, "Survey on the Techniques of FP-Growth Tree for Efficient Frequent Item-set Mining," *Int. J. Comput. Appl.*, vol. 160, pp. 975-8887, 2017.
[10] B. Goethals, "Survey on Frequent Pattern Mining," Univ. Helsinki, pp. 1-43, 2003.
[11] J. Shana and T. Venkatachalam, "An Improved Method for Counting Frequent Itemsets Using Bloom Filter," *Procedia - Procedia Comput. Sci.*, vol. 47, pp. 84-91, 2015.
[12] M. J. Zaki and K. Gouda, *"Fast Vertical Mining Using Diffsets,"* Proceedings of the ninth ACM SIGKDD international, 2003.
[13] W. Bakar, *et al.*, "Postdiffset : An Eclat-like algorithm for frequent itemset mining Postdiffset : an Eclat-like algorithm for frequent itemset mining," *Int. J. Eng. Technol.*, vol. 54, pp. 2-5, 2018.

[14] B. Posey, "Understanding the Impact of RAM on Overall System Performance," 2006, [Online], Available: http://techgenix.com/understanding-impact-ram-overall-system-performance/.
[15] M. J. Zaki, "Scalable Algorithms for Association Mining," *IEEE Trans. Knowledge Data Eng.*, vol. 12, pp. 372-390, 2000.
[16] W. Li, *et al.*, "New Algorithms for Fast Discovery of Association," *KDD-97 Proceedings*, pp. 283-286, 1997.
[17] T. A. Trieu, *"An Improvement for dEclat Algorithm,"* Proceedings of The 6th International Conference on Ubiquitous In- formation Management and Communication (ICUIMC'12*)*, pp. 1-6, 2012.
[18] A. B. P. D. Raut, "NOSQL Database and Its Comparison with RDBMS," *Int. J. Comput. Intell. Res*., vol. 13, pp. 1645-1651, 2017.
[19] M. Benjamin, "High-speed inserts with MySQL," 2017, [Online] Available: https://medium.com/@benmorel/high-speed-inserts-with-mysql-9d3dcd76f723.

BIOGRAPHIES OF AUTHORS

Mahadi Man was born in Terengganu, Malaysia. He received his B.Sc. in Information System Engineering from Universiti Technology MARA (UiTM) on May 2007. His research work in the field of the database system. His area of interest is in the field of big data analytics, machine learning, data mining and data integration.

Dr. Masita Abdul Jalil is currently working as a Senior Lecturer in the School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu, Kuala Terengganu, Terengganu, Malaysia. Her interest is in the field of Software Engineering, Computer Science Education, Programming Languages and Mobile Learning.