A Comparative Study of FP-growth Variations

Aiman Moyaid Said[A], Dr. P D D. Dominic[B], Dr. Azween B Abdullah[C]

Department of Computer and Information Sciences

Universiti Teknologi PETRONAS

Summary
Finding frequent itemsets in databases is crucial in data mining for purpose of extracting association rules. Many algorithms were developed to find those frequent itemsets. This paper presents a summarization and a comparative study of the available FP-growth algorithm variations produced for mining frequent itemsets showing their capabilities and efficiency in terms of time and memory consumption.

Keywords: Data mining, frequent itemsets, FP-growth.

1. INTRODUCTION

An association rule is defined as the relation between the itemsets, since its introduction in 1993 [1] the process of finding the association rules has received a great deal of attention. Today the extracting of association rules is still one of the main popular pattern discovery techniques in knowledge discovery and data mining (KDD).

The process of extracting the association rules can be viewed as two-phases: the first phase is to mine all frequent patterns; each of these patterns will happen at least as frequently as preset minimum support count (min_sup). The second phase is to produce strong association rules from the frequent patterns; these rules must assure minimum support and minimum confidence. The performance of discovering association rules is largely determined by the first phase, [8].

A lot of algorithms were proposed to optimize the performance of the FP-growth algorithm. In this paper we mainly restrict ourselves to study the performance of the FP-growth's Variations in term of running time and the memory usage.

2. RELATED WORKS

In fact, a broad variety of efficient algorithms for mining frequent itemsets have been developed. Agrawal et al in [1], introduced Apriori algorithm to find the frequent itemsets from market basket dataset. The Apriori algorithm adopts candidates' generations-and-testing methodology to produce the frequent itemsets. In the case of the long itemsets the Apriori approach suffer from the lack of the scalability, due to the exponential increasing of the algorithm's complexity.

FP-growth approach for mining frequent itemsets without candidate generation was proposed by Han in [2] . Its scalable frequent patterns mining method has been proposed as an alternative to the Apriori-based approach. The pattern growth approach adopts the divide-and-conquer methodology to produce the frequent itemsets.

This algorithm creates a compact tree-structure, FP-Tree, representing frequent patterns, which moderates the multi-scan problem and improves the candidate itemset generation. This algorithm is faster than others in the literature, this reported by the authors of this algorithm.

Several algorithms implicate the methodology of the FP-growth algorithm. In [3] Pei used the same approach for Mining closed frequent itemsets and max-patterns. Likewise, Pei suggested to Mining sequential patterns in [4].

Further Improvements of FP-growth Mining Methods were introduced. [5],[6],[7], adapted the similar approach of Han et al [2] for mining the frequent itemsets from the transactional database. The authors reported that these algorithms are more efficient than FP-growth.

Manuscript received May 5, 2009
Manuscript revised May 20, 2009

3. FP-GROWTH ALGORITHM REVISIT

FP-growth algorithm is an efficient method of mining all frequent itemsets without candidate's generation. The algorithm mine the frequent itemsets by using a divide-and-conquer strategy as follows: FP-growth first compresses the database representing frequent itemset into a frequent-pattern tree, or FP-tree, which retains the itemset association information as well.

The next step is to divide a compressed database into set of conditional databases (a special kind of projected database), each associated with one frequent item. Finally, mine each such database separately.

Particularly, the construction of FP-tree and the mining of FP-tree are the main steps in FP-growth algorithm.

For the explanation of the algorithm, we will use the following example. To find the frequent itemsets from transactional database DB (see Table 1).First, a scan of the database DB derivers a set of frequent 1-itemsets (L) which also include their support count. The set L is sorted in the order of descending support count, this ordering is important since each path of FP-tree will follow it.

Let the minimum support count be 3,then the set L={(f,4),(c,4),(a,3),(b,3),(m,3),(p,3)}.

Table 1: The transactional database DB

TID	Items
T1	f,a,c,d,g,i,m,p
T2	a,b,c,f,l,m,o
T3	b,f,h,j,o
T4	b,c,k,s,p
T5	a,f,c,e,I,p,m,n

Second, an FP-tree is constructed as follows: The root of the tree, labeled Null, is created. The database DB is scanned for the second time. The items in each transaction are processed in L order, and a branch is created for each transaction.

For example, the scan of the first transaction, "T1:f,a,c,d,g,I,m,p" which contains five items (f,c,a,m,p in L order).Only those items that are in the list of frequent itemsets L ,leads to constructions of the first branch of the tree with tree nodes {<f,1>,<c,1>,<a,1>,<m,1>,<p,1>} where <f,1> is linked as a child of the root. <c,1> is linked to <f,1>,<a,1> is linked to <c,1> ,<m,1> is linked to <a,1>, and <p,1> is linked to <m,1>.

The second transaction, because it shares items f, c and a, it shares the common prefix {f,c,a} with the previous branch and extends to the new branch {<f,2>,<c,2> , <a,2>,<m,1>, <p,1>} .increasing the count of the common prefix by 1.The new intermediate version of FP-tree, after adding two transactions from the database ,is given in Fig. 1.for the remaining transactions can be inserted in the same way (see Fig. 2).

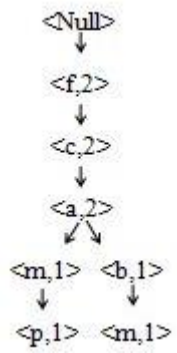

Fig.1 FP-tree for two transactions

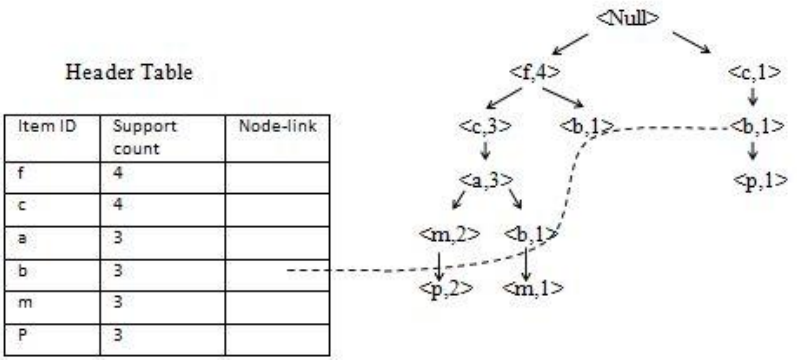

Fig. 2 Final FP-tree

To ease tree traversal, header table is built so that each item points to its occurrences in the tree via chain of node-link.

Using the compact tree structure (or FP-tree), the FP-growth algorithm mines all the frequent itemsets. The FP-tree is mined as follows. Begin from each frequent-1 pattern (as an initial suffix pattern), construct its conditional pattern base (a "subdatabase" which consists of the set of prefix paths in the FP-tree co-occurring with suffix pattern), then build its (conditional) FP-tree, and do mining recursively on such a tree. The patterns growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

In our example, according to L ,the complete set of frequent itemsets can be divided into subsets (6 for our example) without overlapping, first, frequent itemsets having items p (as an initial suffix pattern) ,which is the last item in L, rather than the first item. The reason for starting at the end of the list will become clear as we explain the FP-tree mining process. Second, the itemsets having item m but not p; third, the itemsets that have item b without both m and p; we continue this process to the end. Therefore, the last set will be the large itemsets only with f.

The item p occurs in two branches of the FP-tree of Fig.2.The occurrences of p can easily found by starting from the header table of p and following p's node-links. The paths formed by these branches are {<f,4>,<c,3>,<a,3>,<m,2>,<p,2>}and{<c,1>,<b,1>,<p,1} where samples with a frequent item p are {<f,2>,<c,2>,<a,2>,<m,2>,<p,2>}and{<c,1>,<b,1>,<p,1>}, which form its conditional pattern base ,these samples are the transactions that contain the branch of the tree with the existing of item p. Its conditional FP-tree contains only {<c, 3>} ,the other items are not included because its support count is less than 3.The generated frequent itemset that satisfy the minimum support count is {<c,3>,<p,3>},all the other itemsets are below the minimum support count.

The next subsets of frequent itemsets are those with m item and without p. The FP-tree recognizes the paths {<f,4>,<c,3>,<a,3>,<m,2>}and {<f,4>,<c,3>,<a,3>,<b,1> ,<m,1>},or the related accumulated samples {<f,2>,<c,2> ,<a,2>,<m,2>} and {<f,1>,<c,1>,<a,1>,<b,1>,<m,1>} . Analyzing the samples we find the frequent itemset {<f,3>,<c,3>,<a,3>,<m,3>}.

Similar to subset 3 to 6 the same process is done in our example, additional frequent itemsets can be mined.These are itemsets {f,c,a} and {f,c},but they are already subset of frequent itemsets {f,c,a,m}.Therefore ,the final set of frequent itemsets is {{c,p},{f,c,a,m}}.

4. FP-GROWTH VARIATIONS

Several optimization techniques are added to FP-growth algorithm. In this paper, we investigate the performance of three algorithms, namely AFOPT Algorithm, Nonordfp algorithm and Fpgrowth* algorithm .Our goal is not to go into many details about the algorithms but show the basic optimization ideas and the different of the performance in term of running time and memory consumption. In the following we will illustrate what are the main optimization ideas in each algorithm.

- **AFOPT ALGORITHIM**

Liu et al in [5] investigated the algorithmic performance space of the Fpgrowth algorithm. They specified the problem of conditional databases construction (particularly the number of the conditional databases constructed and the mining cost of each individual conditional database) in Fpgrowth algorithm, which have direct effect on the performance of the algorithm. They studied the problem of enhancing the Fpgrowth algorithm from four perspectives to come with the best strategy for mining the frequent itemsets. These perspectives are the item search order (in what order the search space is explored), conditional database representation, conditional database construction strategy and tree traversal strategy.

For the first part of the problem, the number of the conditional databases constructed can differ very much using different items search orders. The dynamic ascending order is able to minimize the number and /or the size of the conditional database constructed in subsequent mining, AFOPT algorithm adapt this kind of items search order which is also used by Fpgrowth.

For the second part of the problem, the mining cost of each individual conditional database is heavily depends on its representation (tree-based or array-based).AFOPT algorithm use adaptive representation ,tree-based structure in the case of dense dataset and array –based representation in the case of sparse dataset. In additions to the conditional database representation the size and the conditional database construction strategy have effect on the mining cost of each individual conditional database, two type of

the conditional database construction strategy (physical construction or pseudo-construction).
The dynamic ascending frequency search order can make the subsequent conditional databases shrink rapidly. As a result, it is useful to use the physical construction strategy with the dynamic ascending frequency order. The traversal cost of a tree us minimal using the top-down traversal strategy, AFOPT algorithm uses dynamic ascending frequency order for both the search space exploration and prefix-tree construction, it uses the top-down traversal strategy. as a summery AFOPT algorithm utilizes dynamic ascending frequency for the item search space ,adaptive representation for the conditional database format ,physical construction for the conditional database construction, and top-down traversal strategy for the tree traversal.

- **NONORDFP**

The running time and the space required for the Fpgrowth algorithm were the motivation for Nonordfp algorithm. Rácz in [9] dealt with the implementation issues, data structures, memory layout, I/O, and library functions.
A compact, memory efficient representation of an FP-tree by using Trie data structure, with memory layout that allows faster traversal was introduced, to deal with the running time and space requirement problem. This compact representation of FP-tree allows faster allocation, traversal, and optionally projection. It contains less administrative information about the items in the database (no labels for the items are stored in the node, no header lists and children are required), and allows more recursive steps to be carried out on the same data structure, with no need to rebuild it.

- **FPGROWTH* ALGORITHM**

Depending on a numerous experiments were done by Grahne et al [10], they found that 80% of the CPU time was used for traversing FP-trees. Consequently, they employed the array-based to reduce the traversal time of the FP-trees. Fpgrowth* algorithm uses FP-tree data structure in combination with the array-based and incorporates various optimization techniques.

In the case of sparse data set the array-based technique work very well, the array save traversal time for all items and the next level of FP-trees can be initialized directly. While in the case of dense data set, the FP-tree is more compact. To deal with this problem they proposed optimizing technique that help the algorithm to estimate if the data set is sparse or dense, by counting the number of the nodes in each level of the tree which done during the construction of each FP-tree. If the data set turns to be dense data set then no need to calculate the array for the next level of the FP-tree. In the case of sparse data set, the calculation of the array for the next FP-tree is required.

5. COMPARISON OF THE ALGORITHMS

To verify the efficiency of the FP-growth variation algorithms a lot of experiments were conducted. All the experiments are conducted on Core 2 Duo 2.00 GHZ CPU, 2.00 GB memory and hard disk 160 GB. The operating system is ubuntu 8.10.We test the AFOPT algorithm, Nonordfp Algorithm, Fpgrowth* algorithm [11] and the original Fpgrowth algorithm [12].To evaluate the behavior of the four algorithms different datasets and different support threshold were used, in the following subsections the type of the data sets ,the running time and the memory consumption are illustrates:

5.1 Datasets

The data is challenging due to the number of characteristics which are the number of the records, and the sparseness of the data (each records contains only small portion of items).

In our experiments we chose different dataset with different prosperities, to prove the efficiency of the algorithms, Table 2 shows the datasets and the characteristics.

Table 2: The Datasets

Data set	#Items	Avg. Length	#Trans	Type	Size
T10I4D100k	1000	10	100,000	Sparse	3.93 MB
T40I10D100K	1000	40	100,000	Sparse	14.8 MB
Mushroom	119	23	8,124	Dense	557 KB
Connect4	150	43	67557	Dense	8.89 MB

5.2 Running Time: The running time is real time, system time and user time. Figs 3, Fig 4, Fig 5, and Fig 6 depict the time needed in seconds for each one of the algorithms.

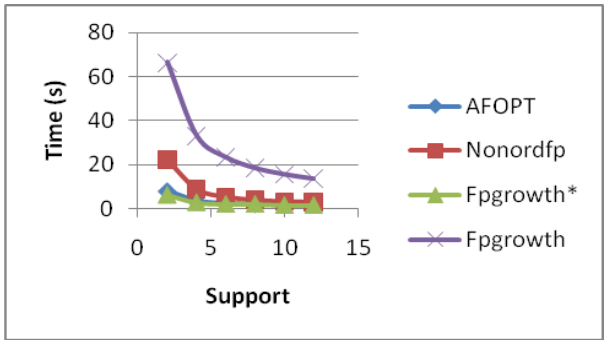

Fig 3 Execution time at various support levels on T10I4D100k

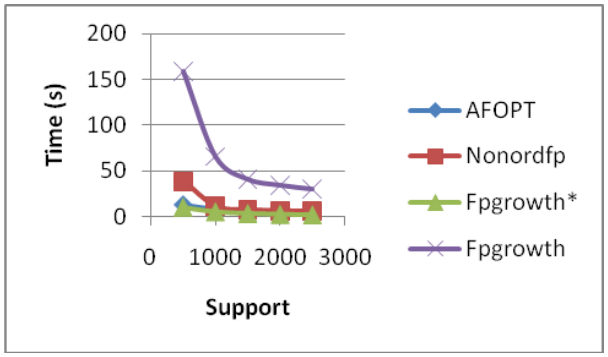

Fig 4 Execution time at various support levels on T40I10D100K

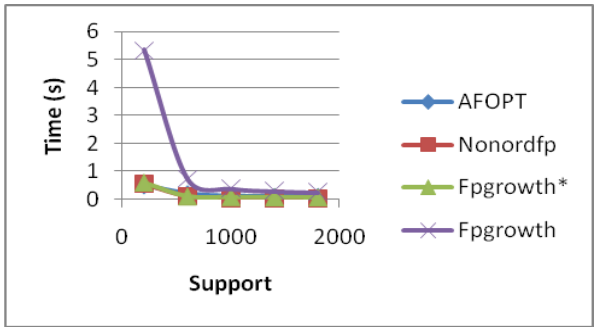

Fig 5 Execution time at various support levels on Mushroom

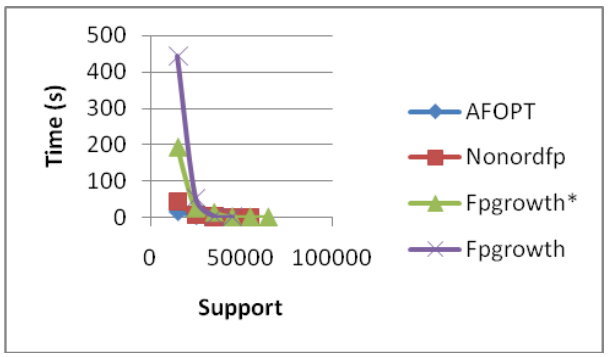

Fig 6 Execution time at various support levels on Connect4

It is clear that with the T10I4D100k data set Fpgrowth* algorithm outperforms all the other algorithms. On T40I10D100K data set there is obvious performance competition among both Fpgrowth* algorithm and AFOPT algorithm. The running times for the AFOPT algorithm, Nonordfp algorithm, and Fpgrowth* algorithm are near in the case of mushroom data set. For the connect4 data set, we should mention that some algorithms had problem, segmentation fault, with some values of support due to the huge number of the frequent itemsets satisfy those thresholds values and some took long time to find the frequent itemsets.

5.3 Memory Consumption: In this section, we calculate the total number of memory consumption for each algorithm .All the experiments are done on the same sets of data. As shown in Fig 7,Fig 8,Fig 9,and Fig 10 the support values and the amount of memory for each one. We observe that, Nonordfp algorithm remains stable over the whole range of support values on T10I4D100k.The stability in memory consumption is also observe for Fpgrowth* algorithm and AFOPT algorithm for the high values of support.

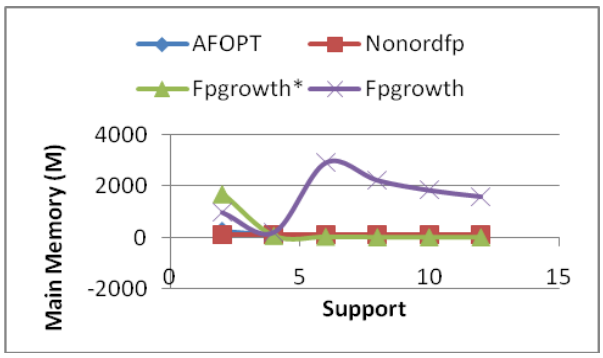

Fig 7 Memory usage on T10I4D100k

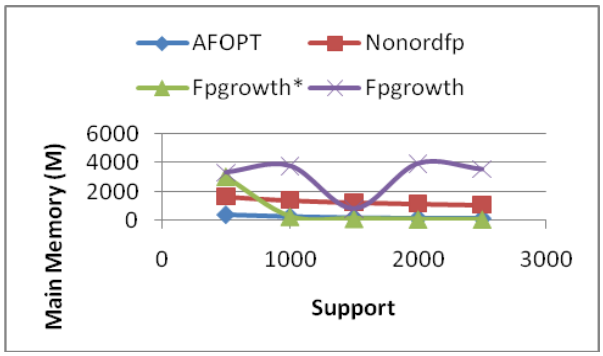

Fig 8 Memory usage on T40I10D100K

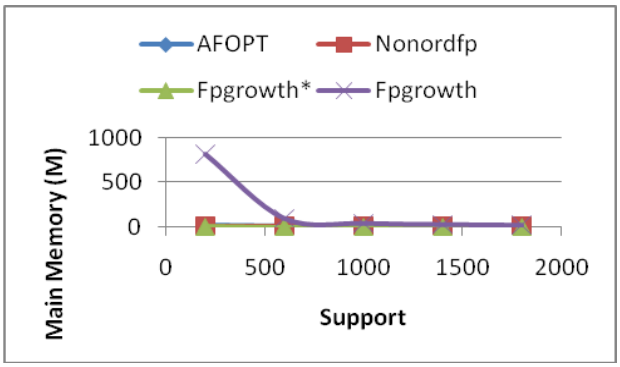

Fig 9 Memory usage on Mushroom

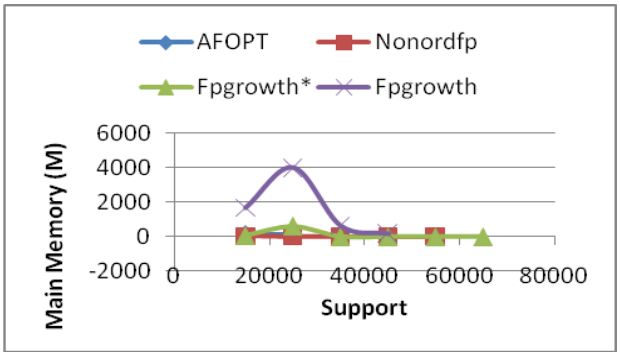

Fig 10 Memory usage on Connect4

AFOPT algorithm keeps its stable consumption of the memory on T40I10D100K. It further confirmed the fact that AFOPT algorithm is stable with sparse data sets. In Fig 9 the competition between the three algorithms is clear, the memory usage is competitive.On Connect4 data set, Fpgrowth* shows stability in the case of the high support thresholds while Nonordfp algorithm remain stable for the low values of support.

6. CONCLUSION

In this paper, we dealt with FP-growth's Variation algorithms. We restricted ourselves to the "Classic" frequent itemsets problem, which is the mining of all frequent itemsets that exist in market basket-like data with respect to support thresholds. The execution time and the memory usage were recorded to see which algorithm is the best. For the time consumption AFOPT algorithm took advantage for most of the data set even though it suffers from segmentation fault in the low support values on connect4 data set. However, for the memory consumption Nonordfp algorithm remains stable for almost all the type of the data set except for the high support values on Connect4. Unfortunately, there is no algorithm work for the all situations. The main restricted for the successful of the algorithm is statistical prosperities of the data set.

REFERENCES

[1] Agrawal , R. , Imieliński , T. , & Swami , A."Mining association rules between sets of items in large databases".In proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pages 207-216, Washington, DC, 1993.
[2] Han, J. , Pei, J. , & Yin, Y. "Mining frequent patterns without candidate generation". In Proc. ACM-SIGMOD Int. Conf. Management of Data (SIGMOD '96), Page 205-216, 2000.
[3] J. Pei, J. Han & R. Mao. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets", DMKD'00, 2000.
[4] J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. ICDE'01, 2001.
[5] Liu,G. , Lu ,H. , Yu ,J. X., Wang, W., & Xiao, X.. "AFOPT:An Efficient Implementation of Pattern Growth Approach", In Proc. IEEE ICDM'03 Workshop FIMI'03, 2003.
[6] Grahne, G. , & Zhu, J. "Fast Algorithm for frequent Itemset Mining Using FP-Trees",IEEE Transactions on Knowledge and Data Engineer,Vol.17,NO.10, 2005 .
[7] Gao, J. "Realization of new Association Rule Mining Algorithm" Int. Conf. on Computational Intelligence and Security ,IEEE, 2007.
[8] Han, J. , & Kamber, M. "Data Mining :concepts and techniques" , second edition, The Morgan Kaufmann Series in Data Management Systems, 2006.
[9] Balázes Rácz," nonordfp: An FP-Growth Variation without Rebuilding the FP-Tree**",** 2nd Int'l Workshop on Frequent Itemset Mining Implementations FIMI2004
[10] Grahne O. and Zhu J. "Efficiently Using Prefix-trees in Mining Frequent Itemsets", In Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining, 2004.
[11] http://fimi.cs.helsinki.fi/
[12] http://adrem.ua.ac.be/~goethals/software/

Aiman Moyaid Said received bachelor degree in computer information system from Yarmouk University, Jordan in 2007 .And currently he is doing his master degree in computer information sciences, at Universiti Teknologi PETRONAS. His research interests include Association Rule Mining, Clustering and application of data mining to problems in retailer industry.

Dr. P.D.D.Dominic obtained his M.Sc degree in operations research in 1985, MBA from Regional Engineering College, Tiruchirappalli, India during 1991, Post Graduate Diploma in Operations Research in 2000 and completed his Ph.D during 2004 in the area of job shop scheduling at Alagappa University, Karaikudi, India. Since 1992 he has held the post of Lecturer in the Department of Management Studies, National Institute of Technology (Formally Regional Engineering College), Tiruchirappalli- 620 015, India. Presently he is working as a Senior Lecturer, in the Department of Computer and Information Science, Universiti Teknologi PETRONAS, Malaysia. His fields of interest are Operations Management, KM, E-business and Decisions Support Systems. He has published technical papers in International, National journals and conferences.

Dr Azween Abdullah obtained his bachelors degree in Computer Science in 1985, Master in Software Engineering in 1999 and his Ph.d in computer science in 2003. His work experiences includes twenty years in institutions of higher learning in both the management and academic capacities, and fifteen years in commercial companies as Software Developer and Engineer, Systems Analyst and IT/MIS and educational consultancy and training. He has spent more than a decade with leading technology firms and universities as a process analyst, senior systems analyst, project manager, and lecturer. He have participated in and managed several software development projects. These have included the development of management information systems, software process improvement initiatives design and implementation, and several business application projects.

His area of research specialization includes computational biology, system survivability and security, autonomic computing and self-healing and regenerating systems, formal specifications and network modeling. His contributions include publishing several journal and refereed conference papers and in the development of programs to enhance minority involvement in bridging the ICT digital gap. Currently he is working on two projects funded by the Ministry of Science Technology and Innovation.