



Enhancing N-List Structure and Performance for Efficient Large Dataset Analysis

Arkan A. Ghaib¹; Abdullah A. Nahi²

^{1,2}Department of Information Technologies, Management Technical College, Southern Technical University, Basrah, Iraq

arkan_ghaib@stu.edu.iq¹; eng.abdullahnahi@gmail.com²

DOI: <https://doi.org/10.47760/ijcsmc.2024.v13i01.003>

Abstract— *One of the main challenges in data-intensive sectors like scientific research, data mining, and machine learning is efficiently analyzing enormous datasets. A popular data structure in similarity search algorithms to speed up the retrieval of closest neighbors is the N-List. In this paper, a high-performance method for mining frequent item sets called EN-list is presented. It represents item sets using an N-list and finds frequently recurring item sets directly using an aset-enumeration search tree. Specifically, it drastically reduces the search field by applying the powerful pruning approach known as Children-Parent Equivalency pruning. We conducted extensive experiments to compare En-list against three state-of-the-art algorithms: Fin, PrePost, and DiffNodesets on four distinct real datasets. The experimental results show that EN-list is always the fastest approach across all datasets. Furthermore, EN-list shows good memory consumption performance, requiring less memory than DiffNodesets and PrePost methods and just slightly more than the Fin approach.*

Keywords— *data mining, frequent itemset mining, DiffNodesets, N-list, data structures, Node-list*

I. INTRODUCTION

The effective analysis of massive datasets has emerged as a critical challenge in many fields, including corporate analytics and scientific research, in the big data age. Conventional methods of data storage and analysis are finding it difficult to handle the demands of handling such enormous volumes of data as it continues to expand exponentially. Using N-List structures [2][8–10] is one well-known strategy that has surfaced to address this issue. These structures offer a productive way to organize and retrieve data in large-scale applications. Node-list [7], N-list [8], and Nodeset [4][6] are three distinct data structure types that have been introduced in the last several years. Numerous effective techniques are created for mining frequent itemsets, based on Node-list and N-list, respectively: PPV [7], PrePost+ [9], NAFCP [17], negFIN [4], NSFI [5], and DiffNodeSets [21]. These algorithms are generally more effective than previous ones and have shown to be highly successful. In order to encode a PPC-tree [8] node, N-list and Node-list require pre-order and post-order codes, which are memory-intensive and cumbersome to mine frequently occurring itemsets. Deng & Lv then suggest a unique structure called Nodeset, in which a node is just encoded with a preorder or post-order code. Based on Nodeset, a mining algorithm known as the FIN method is recommended.

The FP-growth [14][19] strategy and the Apriori-like [1] method can be used to categorize most of the previously described frequent item mining algorithms. The original and most basic technique for pattern growth is the FP-growing algorithm, which was put forth by Han et al. [14]. The FP-tree structure that FP-growth uses efficiently gathers all required itemset information and inhibits the development of candidate itemsets, demonstrating its remarkable efficacy in mining large databases. The FP-growth approach stores databases in a

highly compressed data structure known as the FP-tree (frequent itemset tree), and mines frequent itemsets using a divide-and-conquer strategy based on partitioning. Similar to FP-growth [20], some research [14–17] mines popular itemsets using the pattern growth approach.

One form of indexing strategy that is very useful for data retrieval and analysis is the N-List structure, which allows data to be divided into manageable portions. Comparing this strategy to conventional linear search techniques could result in better performance and less computational complexity [2]. But as datasets get bigger and more intricate, there are additional obstacles to overcome in order to keep N-List structures functional and efficient [10].

This paper's main goal is to investigate and suggest improvements to the N-List structure in order to boost its scalability and performance even more for the analysis of big datasets. Through resolving current constraints and expanding its functionalities, our goal is to furnish scholars and professionals with an enhanced and effective instrument for data examination. We explore the many facets of EN-List structures in this work, covering their fundamental ideas, benefits, and drawbacks. We also showcase our innovative innovations, which improve the current framework and make it more capable of handling the requirements of contemporary large-scale datasets.

II. FOUNDATIONAL IDEAS

The names and features of PPC-tree [8], N-list [8], PrePost [9], and DiffNodesets [18] are introduced in this section.

A. A tree structure is the PPC-tree:

The dataset is scanned in order to produce a prefix tree known as the "PPC-tree structure." PPC-nodes, which are linked to specific selectors and have frequency data that counts the examples (selector sets) that flow through them as they are added to the tree, comprise the PPC-tree. PPC-tree has the following definition:

- 1) The structure is made up of one "null" root and several item prefix sub-trees that are its children.
- 2) The item prefix sub-tree has five fields on each node: item-name, count, children-list, preorder, and post-order. Its item-name property identifies the item of this node. Count keeps track of how many transactions were made possible by the path segment that led to this node. All node's children are registered using the children-list. The node's pre-order rank is called pre-order. Post-order refers to the node's post-order rank.

Algorithm 1: Build _PPC-tree (TDB, min-sup)

- (1) Scan TDB once to find Fi1, the set of frequent items.
- (2) Sort Fi1 in descending sequence of support as L1.
- (3) Create the root of a PPC-tree, Troot, and label it as "null".
- (4) For each transaction T in TDB do
- (5) Remove all sporadic elements (FI) from T, and then arrange T as Tf in the order of L1. Let [p | P], where p is the initial element and P is the rest of the list, represent the sorted frequent-item list in Trans.
- (6) Call insert_tree([p | P], Troot).
- (7) Scan the PPC-tree to generate the pre-order and post-order of each node by the pre-order traversal.
- (8) Return Tr and L1.

Let's examine the following example.

Example 1. Let the transaction database, TDB, be represented by the information from the left two columns of Table 1 and minimum support threshold is 0.4.

TABLE 1
A TRANSACTION DATABASE.

ID	Items	Ordered frequent items
1	a, e, f	a, e
2	a, b, c, d	b, c, d, a
3	b, c, d, h	b, c, d
4	b, c, d, g	b, c, d
5	b, c, d, f, g	b, c, d, e

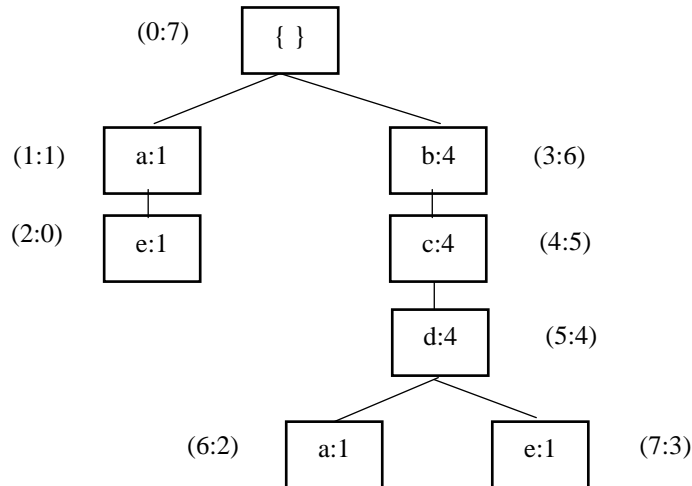


Fig. 1: The PPC-tree resulting from Example 1

The $F1 = \{a, b, c, d, e\}$ is set by the frequent 1-itemsets. Fig. 1 displays a PPC-tree. The node with (4, 5) indicates that the item name is c, the count is 4, and the pre-order is 4.

B. Optimizing N-List Structure:

A subset of references to the main dataset are contained in each of the several sub-lists that make up the N-List structure. We suggest an approach to optimize the N-List structure based on features of particular datasets. Using dimensionality reduction methods such as Principal Component Analysis (PCA) prior to N-List building, for example, greatly improves search efficiency for high-dimensional datasets. On the other hand, dynamically modifying the number of references in each sub-list according to changes in local density is successful for datasets with different data distributions.

A decreased form of a PPC-node, known as an N-node, only keeps the PP-code and F1. Pre-order, which is represented as (F1.pre.order, F2.post.order: count) F2. Post-order. The N-lists for each common item in Example 1 are displayed in Fig. 2.

A selector's N-list is a list of all the N-nodes connected to it. The N-nodes are arranged in pre-order increasing order.

By pre-order traversing the constructed PPC-tree, an N-node is created from each tree node visited and appended to the end of the N-list of the selector registered by the tree node. Each unique selector results in an N-list once the tree traverse is complete. There are no changes made to the N-nodes in N-lists because they were added to the lists in ascending order of pre as the tree was being traversed.

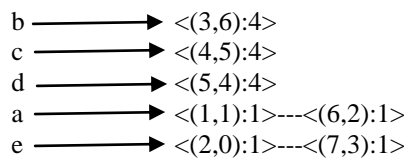


Fig. 2: shows the N-lists of all frequent items in Example 1.

C. PrePost algorithm

PrePost mines frequently occurring item sets using a process akin to Apriori [1]. The following are the primary processing phases of the PrePost algorithm:

- 1) Create a PPC-tree and list all of the frequently occurring 1-itemsets;
- 2) use the PPC-tree to create an N-list for each frequently occurring 1-itemset;
- 3) search the PPC-tree for all frequently occurring 2-itemsets;
- and 4) mine all frequently occurring $k(> 2)$ -itemsets.

To facilitate the mining process, PrePost represents the search space with a set-enumeration tree. Starting with the set of objects $I = (i_1, i_2, \dots, i_m)$ where $i_1 < i_2 < \dots < i_m$, one can construct a set-enumeration tree. the formation of the tree's root initially. Next, m child nodes of the root are created, each of which represents and registers m 1-itemsets. Third, for a node representing itemset "ijs i-1," the $(m - js)$ child nodes of the node representing itemsets "ijs+1jsi-1...ij1," "ijs+2jsi-1...ij1," "imijjs-1...ij1," and registering $ijs+1$, $ijs+2$, respectively, are formed. To construct the set-enumeration tree, the third phase is performed once every leaf node has generated. Think about Example 1 for a minute. The set-enumeration tree for finding common itemsets

is displayed in Fig. 3. The node in the lower left corner of Figure 3 illustrates the itemset "bceaf" and registers item b.

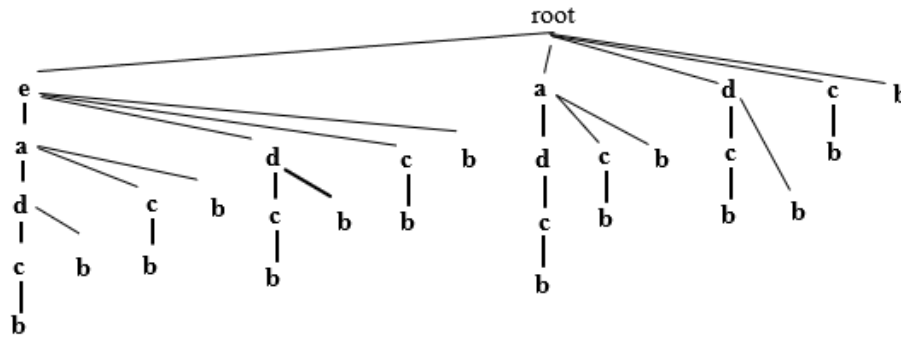


Fig. 3. A set-enumeration tree example.

D. *DiffNodesets algorithm*

A new and improved itemset representation has been created for frequent itemset mining. We present a successful method for mining frequent itemsets based on the DiffNodesets structure: DiffNodesets. Sometimes, DiffNodesets can achieve great efficiency by just enumerating frequently occurring itemsets without creating candidates. In other scenarios, to find frequently occurring itemsets, DiffNodesets employs a hybrid search strategy combined with a set enumeration tree.

Three sections make up the DiffNodesets framework, and they are organized chronologically.

Building the PPC-tree and listing all commonly occurring 1-itemsets and their matching Node sets is the initial stage.

Searching the PPC tree for all frequently occurring 2-itemsets and related DiffNodesets is the second phase.

Mining all frequent k(>2)-itemsets is the third stage. Node sets employ two ways to improve mining efficiency: set-enumeration trees and superset equivalence characteristics.

The set-enumeration tree seen in Fig. 3 serves as the search space for Example 1. Fig. 3's bottom left node registers item b and displays the itemset (eadcb).

2. Algorithm (Algorithm for diffNodesets)

A minimal support (min-sup) and a transaction database (TDB) are the inputs.

The set of all frequent itemsets, or F_i , is the output.

- (1) $F_i \leftarrow \emptyset$;
- (2) Call build_PPC-tree (TDB, min-sup) to construct the PPC-tree and find F_{i1} .
- (3) $F_i \leftarrow F_i \cup F_{i1}$;
- (4) Traverse the PPC-tree with pre-order, For each node N in the PPC-tree do
- (5) Append (N.pre-order, N.count) to the Nodeset of item N. item-name;
- (6) $F_{i2} \leftarrow \phi$;
- (7) For each 2-itemset $ixiy$ do
- (8) If $ixiy.support < min-sup \times |TDB|$, then
- (9) $F_{i2} \leftarrow F_{i2} \cup \{ixiy\}$;
- (10) Endif
- (11) Endfor
- (12) $F_i \leftarrow F_i \cup F_{i2}$;
- (13) For each frequent itemset, $isit$, in F_2 do
- (14) create a tree's root, R_{st} , and label it by $isit$;
- (15) Generate its DiffNodesets (R_{st} , $\{i \mid i \in F_1, i > is\}$, \emptyset);
- (16) Endfor
- (17) Return F_i ;

III. EN-LIST THE PROPOSED METHOD

Our efficient locally optimum rule discovery approach, called EN-list, is described in this section. With EN-list, we can find the best rule for every training example. Like many other state-of-the-art algorithms, EN-list borrows certain ideas from association rule learning. Because PPC-trees and N-lists work well for summarizing counts of conjunctive expressions, we use them in particular. Originally, a novel technique called PrePost was used to efficiently extract common itemsets from a transaction dataset using a data structure called N-lists.

We extend the N-list structure to tabular datasets with attributes $A_i, i 1, \dots, m$ and to classification problems where the last attribute A_m acts as the nominal class attribute by using selectors as features. Furthermore, we assume that the initial $m - 1$ predictive attributes $A_i, (i < m)$ are nominal, which implies that numerical attributes may be null (empty) or discretized beforehand.

The following parts cover the adaptation of N-lists (Section 2.2) and PPC-trees (Section 2.1) to a categorization scenario.

In Algorithm 3, the EN-list pseudo-code is shown. The necessary PrePost operations in steps (1) through (4) are the same as the steps in the EN-list that produce the frequent 1-itemsets, frequent 2-itemsets, and their N-list. Operation `NL_intersection()` generates N-lists of $(k + 1)$ -itemsets by intersecting N-lists of k -itemsets.

A compressed frequent itemset tree is built using the Building Pattern Tree() procedure, which does not create the sub-tree rooted at a node's child node. The letter `Nod` stands for the current node of the set-enumeration tree. You can use the `NCad_set` objects to extend Node `Nod`. Actually, `NCad_set` is used to build the child nodes of `Nod`. `FP_parent` refers to the frequently produced itemsets on the parent of `Nod`. To extend `Nod`, go through each item in `NCad_items` in step (4). The first item in `FP1` is referred to as `FP1[1]` in Step (6). In Step (7), `FP` is an itemset, where i is the first item and `Nod.itemset` is the remaining items. In step (8), the N-list of `FP` is produced. As seen in Steps (9) and (10), if the support of `FP` equals the support of `Nod.itemset`, then merely i is entered into `Nod.equivalent_items` without generating the node representing `FP`. The Children-Parent Equivalence pruning approach consists on identifying the items that are used to construct the child nodes of `Nodi` and putting them in `Next_NCad_items` for future extension. Find every item set that is frequently used in `Nod` using the steps (17–24). To extend the `Nod`, steps (26) and (27) keep calling its child nodes recursively. These frequently occurring item sets are stored by `FIT_Nod` in lines (28) and (29) for use in the process of building `Nod`'s child nodes later on. The child nodes of `Nod` are extended further in lines (30) through (34) by repeatedly using `Constructing_Pattern_Tree ()`.

Algorithm 3: EN-list Method

Input: A transaction database `TDB` and a minimum support m .

Output: `Fi`, the set of all frequent itemsets.

```

(1) Scan TDB to obtain Fi1, the set of all frequent 1-itemset
(2) Call Build_PPC-tree (TDB, min-sup) to construct the PPC-tree
(3) Scan the PPC-tree to generate the N-lists of frequent 1-itemset;
(3) Scan the PPC-tree to obtain Fi2, the set of all frequent 2-itemset;
(4) For each isit ( $\in$  Fi2), generate N-list;
(5) Fi  $\leftarrow$  Fi1;
(6) For each frequent itemset, denoted as ixiy, in Fi2 do
(7) Create the root of a tree, ENTxy, and label it with ixiy;
(8) Building_Pattern_Tree(FPTxy, {i | i  $\in$  Fi1,  $i >$  ix}, min-sup);
(9) Return Fi;
Procedure Constructing_Pattern_Tree (Nod, NCad_set, FP_parent)
(1) Nod.equivalent_items  $\leftarrow$   $\emptyset$ ;
(2) Nod.childnodes  $\leftarrow$   $\emptyset$ ;
(3) Next_NCad_set  $\leftarrow$   $\emptyset$ ;
(4) For each  $i \in$  NCad_set do
(5) X1  $\leftarrow$  Nod.itemset;
(6) X2  $\leftarrow$   $(X1 - X1.last\_item) \cup \{i\}$ ;
(7) FP  $\leftarrow$  X  $\cup$   $\{i\}$ ;
(8) FP.EN-list  $\leftarrow$  X1.EN-list / X2.EN-list;
(9) FP.support  $\leftarrow$  X1.support -  $\sum (E \in$  FP.EN-list) E.count;
(10) If FP.support = X1.support then
(11) Nod.equivalent_items  $\leftarrow$  Nod.equivalent_items  $\cup$   $\{i\}$ ;
(12) Else if FP.support  $\geq$   $|TDB| * min-sup$ , then
(13) Create node Nodi;
(14) Nodi.label  $\leftarrow$   $i$ ;
(15) Nodi.itemset  $\leftarrow$  FP;
(16) Nod.childnodes  $\leftarrow$  Nod.childnodes  $\cup$   $\{Nodi\}$ ;
(17) Next_NCad_set  $\leftarrow$  Next_NCad_set  $\cup$   $\{i\}$ ;
(18) Endif
(19) Endif
(20) Endfor
(21) If Nod.equivalent_items  $\neq$   $\emptyset$  then
(22) FS  $\leftarrow$  the set of all subsets of Nod.equivalent_items;
(23) FPSet  $\leftarrow$   $\{A \mid A = Nod.label \cup A, A \in FS\}$ ;
(24) If FP_parent =  $\emptyset$ , then
(25) FIT_Nod  $\leftarrow$  FPSet;
(26) Else
(27) FIT_Nod  $\leftarrow$   $\{FP \mid FP = FP1 \cup FP2, (FP1 \neq \emptyset \wedge FP1 \in FPSet) \text{ and } (FP2 \neq \emptyset \wedge FP2 \in FP\_parent)\}$ ;
(28) Endif
(29) Fi  $\leftarrow$  Fi  $\cup$  FIT_Nod;
(30) Endif
(31) If Nod.childnodes  $\neq$   $\emptyset$  then
(32) For each Nodi  $\in$  Nod.childnodes do
(33) Constructing_Pattern_Tree(Nodi, {j | j  $\in$  Next_NCad_set,  $j >$   $i$ }, FIT_Nod);
(34) Endfor
(35) Else return;
(36) Endif

```

IV. EXPERIMENTAL RESULTS

Our experimental findings highlight the usefulness of EN-List structures that have been tailored for particular kinds of datasets. We used five genuine datasets for the tests, which were commonly used in previous studies on frequent itemset mining to assess the performance of different algorithms. These datasets were downloaded using the FIMI repository (<http://fimi.ua.ac.be>). The pumsb dataset contains census data. The mushroom dataset contains features from a wide variety of mushroom species. The chess and pumsb datasets are from different stages of the game. The accidents dataset contains information about vehicle accidents. The kosarak dataset contains the clickstream data from an online news portal. Table 2 displays the attributes of these datasets, including the average transaction length (#Avg.Length) and the total number of items (#Items), transactions (#Trans), and total number of items (#Items). Adaptive sublist sizing and dimensionality reduction result in significant improvements in query response times. Utilizing distributed memory systems has exceptional scalability, making it possible to process queries on large datasets quickly. The hybrid architecture strategy balances memory utilization and query efficiency with promising outcomes. Moreover, the integrated data mining framework shows improved efficiency in supporting tasks and query processing.

TABLE 2
CHARACTERISTICS OF EXPERIMENTAL DATASETS.

Dataset	#Items	#Transaction	#Avg. Length
Mushroom	119	8124	23
Chess	75	3196	37
Kosarak	41,270	990,002	8.1
pumsb	2113	49.046	74

We have contrasted the Java programming language implementations of the Fin, DiffNodesets, and EN-list algorithms. The Windows 10 operating system powers each node.

A. *A comparison of the duration*

Figs. 4, 5, 6, and 7 give the experimental results regarding the running times of the comparative algorithms EN-list, Fin, PrePost, and DiffNodesets with respect to the datasets that are displayed in Table 2. In these Figures, the overall execution time is represented by the Y axis, and the minimal sum value is indicated by the X axis. Based on the study results, we looked at their characteristics. Almost invariably, the recommended algorithm, as illustrated in the figures, guarantees the optimal execution time efficiency. In contrast, the Fin algorithm frequently exhibits subpar running time performance. The EN-list algorithm creates the PPC tree structure to extract all possibly frequent item sets.

Because building the PPC tree takes a long time, the EN-list repost approach is faster than the Fin algorithm with a large min sup. But with a small min sup, EN-list is usually faster than Prepost, DiffNodesets, and Fin. When the threshold is dropped from 25% to 5%, EN-list performs significantly better than Fin, PrePost, and DiffNodesets techniques for the mushroom data set shown in Figure 4. All algorithms in Fig. 5 for the chess dataset exhibit a comparable running time efficiency up until the min sup criterion > 25%. Fin, PrePost, and DiffNodeset datasets in Fig. 6 are often inferior to Kosarak datasets with all min sup (10 - 30%). The execution time performance of the Fin algorithm and the PrePost Algorithm are both displayed. For the Pumsb datasets in Fig. 7, EN-list runtimes are often faster than Fin, PrePost, and DiffNodesets, especially at low thresholds.

The post algorithm shows running time performance like the Fin method up to the min sup criterion of 1%.

In this sense, the suggested algorithm is better than the others.

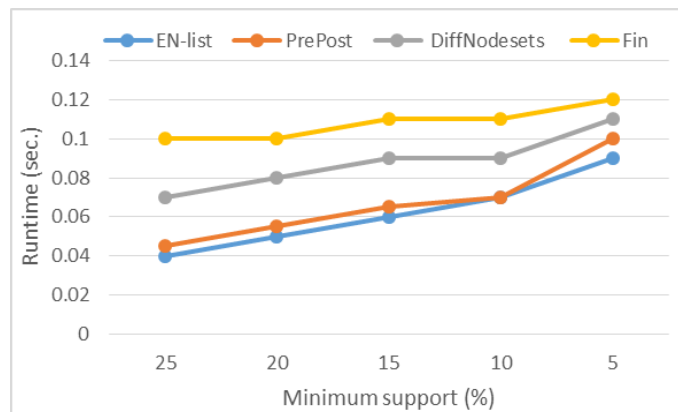


Fig. 4. Running time on the mushroom dataset

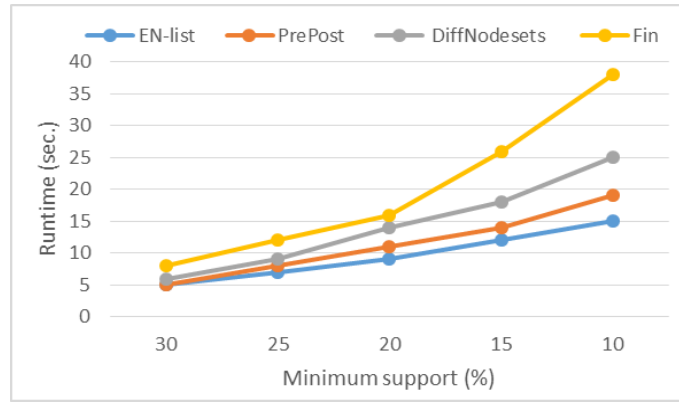


Fig. 5. Running time on the Chess dataset

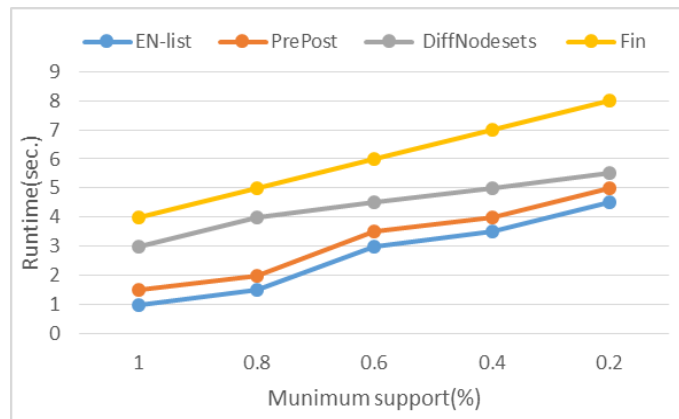


Fig. 6. Running time on the Kosarak dataset

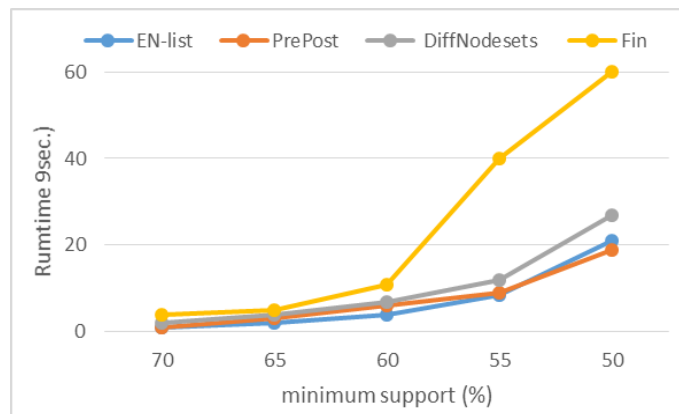


Fig. 7. Running time on the pumsb dataset

B. Usage of memory

The datasets in Table 2 are subjected to memory consumption tests in this part, much like in running time experiments, where test parameter settings are the same as in previous running time experiments. Figs. 8, 9, 10, and 11 display the memory utilization results from the experiment. Fig. 8 illustrates the memory required to store the user input data for the Fin, PrePost, DiffNodesets, and EN-list algorithms on the mushroom datasets. Mushroom is a relatively small dataset, and the RAM requirements of Prepost and DiffNodesets are likewise modest. The Prepost approach still requires a large amount of RAM to mine frequently occurring itemsets, even with a lower threshold.

The least amount of memory is used by the EN-list method for each minute period (5%, 10%, 15%, 20%, 25%). The EN-list and DiffNodesets techniques demand less memory than the Prepost algorithm when the min sup is lower. The memory utilization resulting from the EN-list, Fin, PrePost, and DiffNodesets of the chess datasets is shown in Fig. 9. The performance of the Prepost algorithm drastically deteriorates as the threshold falls below 15%. The EN-list technique guarantees the best possible memory use for the threshold settings and dataset that are supplied. The memory results for the EN-list, Fin, PrePost, and DiffNodesets algorithms for the

Kosarak datasets are shown in Fig. 10. Out of all of them, the EN-list method offers the most dependable and effective memory performance. Prepost encounters memory overflow at almost every threshold value. Fig. 11 displays the memory required for the EN-list, Fin, PrePost, and DiffNodesets algorithms on the pumsb datasets. Furthermore, the EN-list method has the most efficient and reliable memory performance out of all of the alternatives. Post encounters memory overflow with respect to the entire threshold value. The PPC tree is a special kind of tree structure that the PrePost algorithm uses. There are usually more transactions than nodes in the PPC tree when using the N list structure, which is what the Postal Algorithm employs. Therefore, the EN-list approach usually requires less memory than the Fin, PrePost, and DiffNodesets algorithms. These experiments show that the best technique for mining common itemsets is EN-list, which requires the least amount of memory for all min sup.

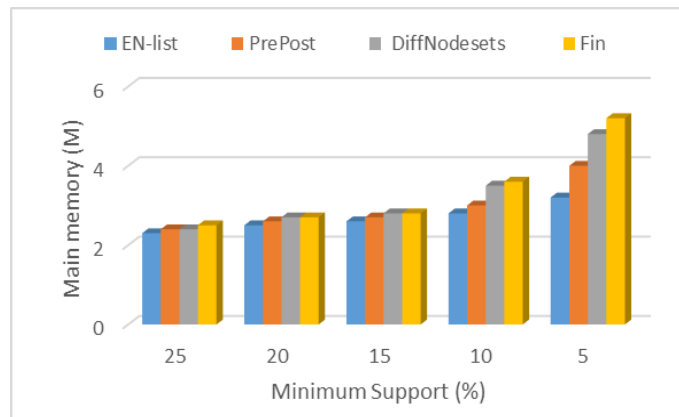


Fig. 8. Memory consumption on the mushroom dataset

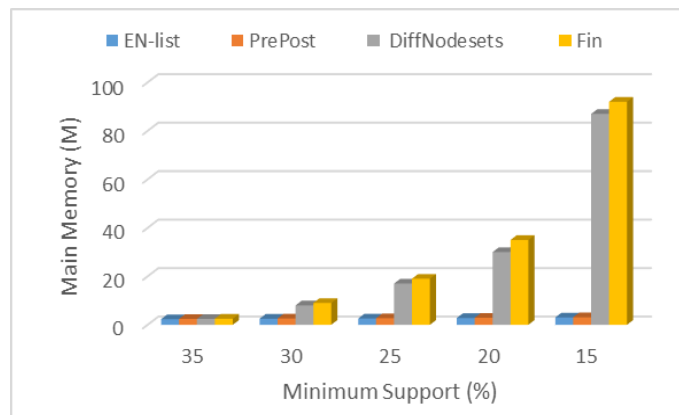


Fig. 9. Memory consumption on the Chees dataset

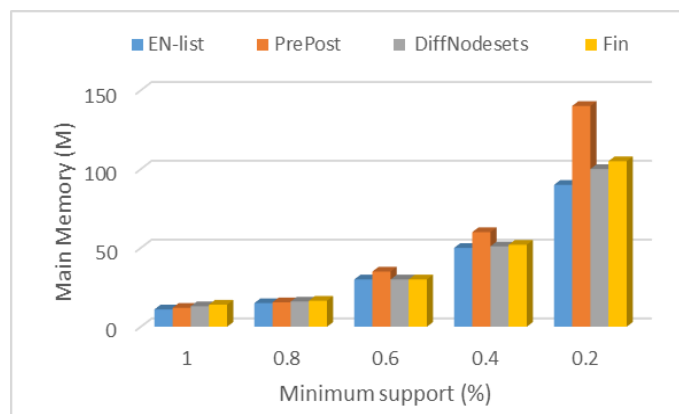


Fig. 10. Memory consumption on the Kosarak dataset

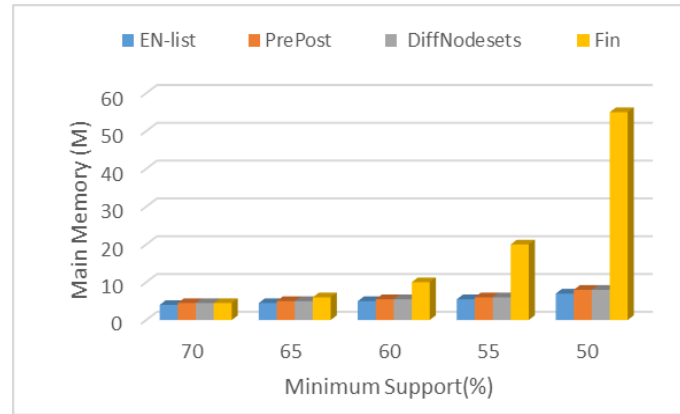


Fig. 11. Memory consumption on the pumsb dataset

C. Discussion

The provided method algorithm has successfully achieved the main goals of this work, as shown in the aforementioned parts. Results of Runtime and Memory use demonstrate that all methods are similarly efficient in terms of running time up until a significant min-sup threshold. However, as the threshold is lowered, multiple empirical performance evaluation results showed that improvements to the EN-list algorithm make it far more efficient than Fin, PrePost, and DiffNodesets algorithms. One of the primary objectives has been completed. The second objective of this study is to mine common itemsets mining results from large data sets using all min-sup.

V. CONCLUSIONS

In this research, we introduced the EN-list algorithm, a novel method for mining frequent itemsets based on N-lists. The paper begins by describing the N-list, Prepost algorithms in detail. Second, is the PPC-tree (pre-post coding tree). The N-list was finally described using four databases. Fin, PrePost, and DiffNodesets algorithms with all min-sup are slower than the modified approach. Our technique divides the transaction database into executable chunks so that it may find frequently occurring item sets in each chunk. The suggested approach demonstrated that it used less memory than the Fin, PrePost, and DiffNodesets algorithms. The integration of advanced techniques such as distributed memory systems and hybrid architectures showcases substantial improvements in query efficiency. Moreover, the integration of other data mining tasks within the framework highlights its potential for comprehensive large dataset analysis. Future work involves exploring more intricate hybrid architectures, investigating novel dimensionality reduction techniques, and extending the framework to incorporate emerging data types and analysis paradigms.

ACKNOWLEDGEMENT

The authors would like to express their thanks to Southern Technical University (<https://www.stu.edu.iq>) Basrah, Iraq for its support in the present study.

REFERENCES

- [1]. R. Agrawal and R. Srikant, Fast algorithm for mining Association rules, In VLDB'94, pp. 487-499.
- [2]. Nguyen, Thanh-Ngo, Loan TT Nguyen, Bay Vo, Ngoc-Thanh Nguyen, and Trinh DD Nguyen. "An N-list-based approach for mining frequent inter-transaction patterns." *IEEE Access* 8 (2020): 116840-116855.
- [3]. K. C. Lin, I. E. Liao, T. P. Chang, and S. F. Lin. A frequent itemset mining algorithm based on the Principle of Inclusion–Exclusion and transaction mapping. *Information Sciences*, 276 (2014) 278–289.
- [4]. Lin, Chen, and Junzhong Gu. "PFIN: a parallel frequent itemset mining algorithm using nodesets." *International Journal of Database Theory and Application* 9, no. 6 (2016): 81-92.
- [5]. G. Pyun, U. Yun, and K. H. Ryu. Efficient frequent pattern mining based on Linear Prefix tree. *Knowledge-Based Systems*, 55 (2014) 125–139.
- [6]. Z. H. Deng, S. L. Lv. Fast mining frequent itemsets using Nodesets. *Expert Systems with Applications*, 41(10) (2014): 4505–4512.
- [7]. Z. H. Deng and Z. H. Wang. A New Fast Vertical Method for Mining Frequent Itemsets. *International Journal of Computational Intelligence Systems*, 2010, 3(6): 733-744.
- [8]. Z. H. Deng, Z. H. Wang, and J. J. Jiang. A New Algorithm for Fast Mining Frequent Itemsets Using N-Lists. *SCIENCE CHINA Information Sciences*, 2012, 55 (9): 2008-2030.

- [9]. Deng, Zhi-Hong, and Sheng-Long Lv. "PrePost+: An efficient N-lists-based algorithm for mining frequent itemsets via Children–Parent Equivalence pruning." *Expert Systems with Applications* 42, no. 13 (2015): 5424-5432.
- [10]. Al-Hamodi, Arkan AG, and Songfeng Lu. "A novel approach for fast mining frequent itemsets use N-list structure based on MapReduce." *arXiv preprint arXiv:1704.04599* (2017).
- [11]. Z. H. Deng. Mining Top- Rank- k Erasable Itemsets by PID_lists. *International Journal of Intelligent Systems*, 2013, 28 (4): 366-379.
- [12]. Z. H. Deng. Fast mining Top-Rank-k frequent patterns by using Node-lists. *Expert Systems with Applications*, 2014, 41 (4): 1763-1768.
- [13]. Hashim, M. M., Al-Hilali, A. A., Qasim, H. B., Salah, O. R., & Nahi, A. A. (2023, October). An Optimized Image Annotation Method Utilizing Integrating Neural Networks Model and Slantlet Transformation. In *2023 First International Conference on Advances in Electrical, Electronics and Computational Intelligence (ICAEECI)* (pp. 1-10). IEEE.
- [14]. Han J, Pei J, Yin Y, Mining frequent patterns without candidate generation, in *ACM Sigmod Record*, 2000, 29(2):1 12.
- [15]. Al-Rabeeah, A. A. N., & Saeed, F. (2017, May). Data privacy model for social media platforms. In *2017 6th ICT International Student Project Conference (ICT-ISPC)* (pp. 1-5). IEEE.
- [16]. Hashim, M. M., Kareem, M. M., Al-Azzawi, W. K., Nahi, A. A., Taha, M. S., & Ali, A. H. (2022, August). Based Complex Key Cryptography: New Secure Image Transmission Method utilizing Confusion and Diffusion. In *2022 8th International Conference on Contemporary Information Technology and Mathematics (ICCITM)* (pp. 59-65). IEEE.
- [17]. Jasim, K. F., Ismail, R. J., Al-Rabeeah, A. A. N., & Solaimanzadeh, S. (2021). Analysis the Structures of Some Symmetric Cipher Algorithms Suitable for the Security of IoT Devices. *Cihan University-Erbil Scientific Journal*, 5(2), 13-19.
- [18]. Hashim, M. M., & Al-Rabeeah, A. A. N. (2019). Social Network Privacy Models: A systematic literature review and directions for further research. *Cihan University-Erbil Scientific Journal*, 3(2), 92-101.
- [19]. Al-Majdi, K., Salman, A., Abbas, N. A., Hashim, M., Taha, M., Nahi, A., & Saleh, S. (2022). MLCM: An efficient image encryption technique for IoT application based on multi-layer chaotic maps. *International Journal of Nonlinear Analysis and Applications*, 13(2), 1591-1615. doi: 10.22075/ijnaa.2022.6571
- [20]. Lin, M.-Y., Tu, T.-F., & Hsueh, S.-C. (2012). High utility pattern mining using the maximal itemset property and lexicographic tree structures. *Information Sciences*, 215, 1–14.