Journal of Ambient Intelligence and Humanized Computing
https://doi.org/10.1007/s12652-020-01706-8

ORIGINAL RESEARCH

An improved memory adaptive up-growth to mine high utility itemsets from large transaction databases

D. Sathyavani[1] · **D. Sharmila**[2]

Received: 27 August 2019 / Accepted: 7 January 2020

Abstract
High utility itemset (HUI) mining identifies an interesting pattern from transactional databases by taking into consider the utility of each in the transaction for instance, margins or profits. Many candidates are generated for HUIs which reduces the mining performance. Frequent pattern-growth (FP-Growth) algorithm was widely used to discover the frequent itemsets using FP-Tree. But, UP-Tree was constructed to store high utility item set. The mining of UP-Tree by FP-Growth extracts high utility itemset and generates too many candidates. So, UP-Growth and UP-Growth$^+$ was proposed to shorten the candidate itemsets. In UP-Growth, two tactics such as discarding local unpromising items (DLU) and decreasing local node (DLN) were used in FP-Growth. In UP-Growth$^+$, two strategies such as DLU and its estimated node utilities (DNU) and DLN tools for the nodes of internal UP-Tree by evaluated the descendant nodes (DNN) were incorporated in FP-Growth. However, the UP-Growth and UP-Growth$^+$ has the problem of poor spatial and temporal localities. Initially, the UP-Tree is created in available main memory then extended to secondary memory when the transaction is large. The storage of data structure in secondary memory and accessibility is not clearly derived in existing algorithms. In this paper, the spatial locality issues of UP-Growth and UP-Growth$^+$ is solved by rearranging nodes of UP-Tree in a depth first manner and temporal locality issues of UP-Growth and UP-Growth$^+$ is solved by page blocking technique which reorganizes the execution part of UP-Growth and UP-Growth$^+$. The computational part is rearranged. In addition to, the memory management strategy is introduced to minimize the space requirement of high utility itemsets. Thus the proposed Improved Adaptive UP-Growth (IAUP-Growth) and Improved Adaptive UP-Growth$^+$ (IAUP-Growth$^+$) overcomes the spatial and temporal locality problem and effectively reduces the memory usage.

Keywords Frequent pattern growth · Utility pattern tree · High utility itemset mining · Memory management · Utility pattern growth

1 Introduction

Data mining techniques such as pattern mining and rule mining (Gan et al. 2018) are used for making crucial decision through extracting interesting patterns from database. Two important processes named as Association Rule Mining (ARM) (Wang et al. 2018) and frequent itemset mining (FIM) (Memar et al. 2012) are used to find out interesting relationship between items in transactional databases. ARM is intended to obtain the inviolable rules in database by using some interestingness measures. In FIM, items are called frequent when the expected support is larger than user specified threshold minimum support (minsupp) value. ARM reveals the relationship between objects by finding the strongest rules for binary transaction information. But some important features are not considered as profit on the goods and quantity purchased. Such problems are somewhat solved by quantitative association rule mining (QARM), which reveals the relationship between information items by taking into account the quantity value that each item in a transaction may contain. Frequent type-development (equivalent party-development) the ARM algorithm is widely used to find objects with actionable information, often with the help of the Party-Tree. It often finds items without candidate generation, so it requires less memory.

✉ D. Sathyavani
sathyavanid@yahoo.com

[1] Faculty of Computer Science and Engineering, United Institute of Technology, Coimbatore, Tamil Nadu, India

[2] Bannari Amman Institute of Technology, Sathyamangalam, Tamil Nadu, India

Published online: 06 February 2020

But, the existing algorithms were not considered other features such as cost, importance, price, weight or utility of items. In order to analyze and predict customer purchase behavior, the above mentioned factors are more important. These algorithms focus on discovering frequent patterns without taking into account of profitability. High Utility Itemsets mining (HUIM) used to handle the limitations in existing rule mining such as ARM and QARM.

The HUIM technique (Joshi and Patel 2014) determined all itemsets in transaction database which are having utility beyond a user defined threshold minimum utility value. Here, the utility represents the profit or quantity value of items in the transaction. The utility value depends on two important factors are internal utility and external utility. It represents the significance of distinct items as well as the utility of items in transactions. In order to discover the HUIs in transactional database, candidate itemsets are generated by determining the utilities and calculates accurate utilities of generated candidate itemsets through database scans. Different approaches were used for HUIM. However, these approaches generated large number of candidate itemsets. It has two problems that are extra memory requirement to keep candidate itemsets also requires more execution time to generate candidates and estimating its exact utilities. If the candidate itemsets are large in numbers, then the main memory is not enough to store the items. Performance of these algorithms is degraded due to thrashing.

Tseng et al. (2013) proposed the efficient algorithms UP-Growth and UP-Growth$^+$ to extract HUIs. It also reduces the number of candidate generations. By scanning the transaction two times, a UP-global tree was constructed. Then, from the constructed UP-global tree, a Potential High Utility Itemsets (PHUIs) was recursively generated by applying proposed UP-Growth algorithm. The UP-Growth was generated by including two strategies such as DLU and DLN into FP-Growth algorithm. The UP-Growth$^+$ reduced over estimated utilities effectively by incorporating two strategies DLU items, and their estimated Node Utilities and DLN tools for the nodes of local UP-Tree by evaluated tools of descendant Node(DNN) in FP-Growth. After the determination of all PHUIs, the re-arranged transactions were scanned to find out the high utility itemsets (Joshi and Patel 2014). A branch of the UP-Tree is placed in the secondary memory when the tree is unable to fit into the main memory (Manike and Hari 2014). So the multiple disc and primary memory swaps may emerges during the mining process which leads to poor spatial locality of the tree nodes (Adnan and Reda 2011). When the mining algorithm tries to construct the conditional pattern bases, the UP-Tree in secondary memory needs to be loaded into primary memory. This needs to be done for every frequent item in the header table. It loads the same data block multiple times which leads to poor temporal locality of the data. Hence, the UP-Growth and UP-Growth$^+$ has the problem of poor spatial and localities.

In this paper, the problem of spatial and temporal localities of UP-Growth and UP-Growth$^+$ is addressed and these problems are solved by proposing Improved Adaptive UP-Growth (IAUP-Growth) and IAUP-Growth$^+$ algorithms. But the temporal locality and spatial locality issues of UP-Growth and UP-Growth$^+$ are resolved by the same strategy. The strategy is applied in the UP-Growth algorithm, then it is called as IAUP-Growth and the strategy is applied in UP-Growth$^+$ algorithm, then it is called as IAUP-Growth$^+$. In IAUP-Growth and IAUP-Growth$^+$, the spatial locality problem is solved by reorganizing the nodes of UP-Tree based on depth first manner. Another problem, temporal locality is solved by page blocking technique which reorganizes the execution part by UP-Growth and UP-Growth$^+$. Moreover, memory management strategy is described in this paper which reduces the space requirement of high utility itemsets. In the memory management strategy, UP-Tree is constructed in the available allocated primary memory. When the structure exceeds out of the allocated memory, the data is mandatory to save on secondary memory as separate. This structure is accessed by block-by-block basis so that both the spatial and temporal localities of I/O access are optimized Adnan and Reda (2011). Hence the UP-Growth and UP-Growth$^+$ are improved by the proposed IAUP-Growth and IAUP-Growth$^+$ by resolving the spatial and temporal locality problem and by using memory management strategy.

The article is structured as follows: Sect. 2 describes the literature survey about HUIM. Section 3 describes the proposed methodology. Section 4 illustrates experimental results of the proposed approach. Finally, Sect. 5 gives conclusion of the research work.

2 Related works

The efficient algorithms (Tseng et al. 2016), mining Top-K Utility itemsets (TKU) and mining Top-K utility itemsets in One phase (TKO) were proposed to mine top k HUI in transaction database. The main advantage of these algorithms were mined HUI without user defined minimum utility threshold value. By adopting UP-Tree, the proposed TKU was maintained details about the transaction and utilities of each item. TKO used data structure to gather the details of itemset utilities from the transaction database. In a single phase, it found out the top k utility HUIs by using vertical data representation techniques. These algorithms were effective for mining HUIs. However, these algorithms were unable to integrate with supplementary utility mining steps for finding out the various kinds of Top-K high utility algorithms.

An efficient method (Krishnamoorthy 2018b) was proposed with various minimum utility threshold values. In a single phase, this method generated HUI without expensive candidate generation. It utilized vertical database representation which stores information about the itemset and mines HUIs effectively. Moreover, a theory of multiple minimum utility was introduced and presented the generalized pruning methods to mine HUIs efficiently. However this method is quite challenging.

A Generalized High Utility Mining (GHUM) method (Krishnamoorthy 2018a) was proposed for mining HUIs. The advantage of this method was considering both the negative and positive unit profits to mine HUIs. While mining process, the proposed method had a compact data structure which is used to store the utility information of itemsets. In addition to that, the performance of HUI was enhanced by introducing a new anti-monotonic property based on utility. An evaluation was performed during the mining process. Furthermore it minimizes the evaluation steps by using N-prune strategy. As well as enhanced the performance of HUIM. However, the total item utility values of each transaction were negative by using GHUM.

An efficient mining algorithm with multi objective (Zhang et al. 2018) was proposed to mine HUIs and frequent itemsets. This algorithm considered two measures are utility and support in a unified structure from a multi perspective aspect. The system of frequent with high utility structure mining was designed as a multi perspective issue, and the proposed algorithm helps as decision maker by providing multiple itemsets recommendation. The main benefit of this algorithm is no need to define minimum support threshold value and minimum utility value in advance.

A HUI-DTP algorithm (Lin et al. 2016) was proposed for mining HUI. This algorithm was considered discount strategies of items. This algorithm comprised of two phase algorithm which were based on a vertical Transaction Identifier (TID) list structure and novel downward closure property. In addition to, the HUI was discovered effectively without generation of candidates by using a HUI DMiner algorithm. It depends on properties of compact data structure and pruning techniques. In order to minimize the computation process, the relationship between two itemsets was stored by using Estimated Utility Co-occurrence strategy (Lin et al. 2016) which applied on the improved HUI DMiner algorithm.

An incremental mining algorithm (Lin et al. 2012a, b) was proposed to mine HUIs. Two phase algorithm was used for mining HUIs. The problems of processing all transactions in batch wise were solved by incremental mining algorithm. It was derived from the conception of fast update (FUP) approach. It doesn't focus the problem of utility based mining, when the modification is performed in original database.

A new approach called as short-period high utility itemset mining (SPHUIM) (Lin et al. 2017) was projected to mine the HUIs from transactional databases by taking into consideration of both the period constraint and utility measures. In this framework, three algorithms were designed. First algorithm called $SPHUI_{TP}$ mined short period HUIs by level wise process. Candidates were obtained by the first algorithm and selected the SPHUIs through multiple database scans. In third algorithm, the process of SPHUIs mining was speed up by using two pruning strategies which reduced the search space as well as pruned unpromising candidates from the transaction database. This framework has excellent scalability. In order to mine more specific patterns, along with the utility measures and period constraints could be considered.

A new approach called high Utility Mining using the Maximal Itemset property (UMMI) was proposed (Lin et al. 2012a, b) to mine HUIs. Initially, this algorithm reduced the number of potential itemsets. Then, an effective tree structure called as lexicographic tree structure was constructed and determines HUIs effectively. There are two phases in UMMI, Maximal phase and utility phase. They reduced search space and identified the HUIs respectively. This algorithm still needs an improvement in terms of memory.

A new approach, high utility pattern tree (HUP-Tree) and HUP-Growth algorithm (Lin et al. 2011) was proposed to mine HUIs in two phases. The proposed method was incorporated the conventional two phase system for HUIM. The compact tree structured called HUP-tree behaves like FP-tree except the HUP-tree maintains quantity of items in an array that attached with each node for utility mining. The candidate itemsets and also specific items were generated at a time using HUP-Growth algorithm. However, in this approach the database is considered as static. But the database is generally dynamic in real world applications. So this method was not able to process the utility mining in dynamic database.

A novel algorithm called as temporal high utility itemsets (THUI) (Chu et al. 2008) was proposed for mining sequential HUIs from large databases. This method utilized maximal possible error and bucket boundaries to delete or update itemsets with frequency in transaction database. The itemsets were deleted whose occurrence counts in memory were greater than utility threshold. The occurrence count was calculated by using cumulative filter (CF). However, the performance of THUI method was depends on the value of minimum utility threshold.

For mining HUIs, a novel algorithm called as efficient HUIs mining (EFIM) (Zida et al. 2015) was proposed. The search space is reduced by using upper bounds in it. Sub tree utility and local tree utility are the two upper bounds. The utility values were calculated in linear space and time by using fast utility counting. In addition to, two algorithms were proposed in EFIM were named as transaction merging

technique and efficient database projection technique. These techniques were reduced the database scans costs (Anitha and Kaarthick 2019). Identifying intruders using data mining approach in recent trend provides better detection rate when compared with other classical systems. In this paper we introduced Oppositional based Laplacian grey wolf optimization algorithm for clustering the class of attacks based on the similarity and active learning of SVM classification using this optimization algorithm.

3 Proposed methodology

In this section, the spatial and temporal locality problem in UP-Growth algorithm and UP-Growth$^+$ algorithm is described. Then, the proposed IAUP-Growth as well as IAUP-Growth$^+$ are described in detail. The UP-Growth and UP-Growth$^+$ methods exhibit poor temporal and spatial localities. To get better spatial and temporal localities in UP-Growth and UP-Growth$^+$ by identifying strategies by which UP-Growth and UP-Growth$^+$ can be made I/O conscious and by using cache conscious prefix tree. Initially, UP-Tree is created to retain the details of HUIs in the transactions. It avoids frequent scanning of original database and increases the mining performance. UP-Growth is very similar to FP-Growth with two strategies are DLU and DLN. With FP-Growth two strategies such as DNU and DNN was incorporated to get UP-Growth$^+$. So the problems of FP-Growth are reflected in UP-Growth and UP-Growth$^+$. The problems associated with existing algorithms. It causes poor temporal and spatial localities. This is due to its access behavior and construction process of the mining algorithm. The spatial and temporal locality problem in UP-Growth and UP-Growth$^+$ is solved by the same strategies used in proposed algorithms.

3.1 Problem definition

3.1.1 Poor spatial locality

A leaf node is inserted into the branch during construction of UP-Tree and it also maintains the property of prefix path in the UP-Tree. In UP-Tree, prefix path are formed and their nodes are placed in the linear memory depending upon the sequential order by which transaction were read and included into the file structure. While constructing the UP-Tree, the nodes of prefix path in the UP-Tree are not located linear to each other. This may leads to infrequent cache misses when the algorithm determines the prefix paths during the UP-Tree construction or traversing UP-Tree. Furthermore, when the branch of UP-Tree is located in the disc by using virtual memory management system, multiple discs swaps and primary memory may be explored during the process of HUIM. Since regular page faults, leads lack of memory space allocation for the nodes. Therefore the process of high utility mining requires maximum space and time.

3.1.2 Poor temporal locality

Virtual memory manager (VMM) is like memory managers which manages the memory requests made by the system and its application. The UP-Tree needed to load into primary memory while the remaining part of UP-Tree is resides in the secondary storage. It needs to be performed for all itemsets in the header table. While performing this step, the UP-Growth algorithm loads several times backward and forward from hard disc to main memory. This leads to poor temporal locality of the data.

3.2 Resolving spatial and temporal locality issue

3.2.1 Resolving spatial locality issues

By rearranging the UP-Tree nodes in depth first manner, the spatial memory allocation is improved. The reorganized version of original UP-Tree called conscious prefix tree is used in this task. Initially, a contiguous block of memory is allocated which is equivalent to the UP-Tree size. Then, UP-Tree is navigated in the approach of depth first. In order to guarantee the improved spatial locality of nodes in the same prefix path, UP-Tree nodes are copied linearly into new assigned contiguous memory. The mining algorithm wants to go across the tree several times by bottom up approach. By rearranging the tree traversal on cache conscious prefix tree, it enhances the performance in spatial locality of nodes. The Fig. 1 shows an example for UP-Tree with node construction and allocation in the memory. The nodes are numbered in red color based on the nodes allocation or creation time. An allocation of node in the memory is represented by array like manner. The spatial problem in UP-Tree is solved by rearranging the nodes as well as traversed by depth first manner. The rearranged UP-Tree nodes are depicted in Fig. 2.

3.2.2 Resolving temporal locality issue

The UP-Growth shows poor temporal locality since UP-Tree is read multiple times in the header table for each itemsets. In this process, to determine the conditional pattern bases, need to collect the counts in the first step then builds the conditional UP-Tree from collected patterns. The system goes slow down massively due to thrashing process. When the virtual memory management system accumulates a part of tree in secondary memory, then it leads to constant page faults. This problem is overcome by rearranging the computational part of the UP-Growth through the process of page blocking. The computational part is rearranged by

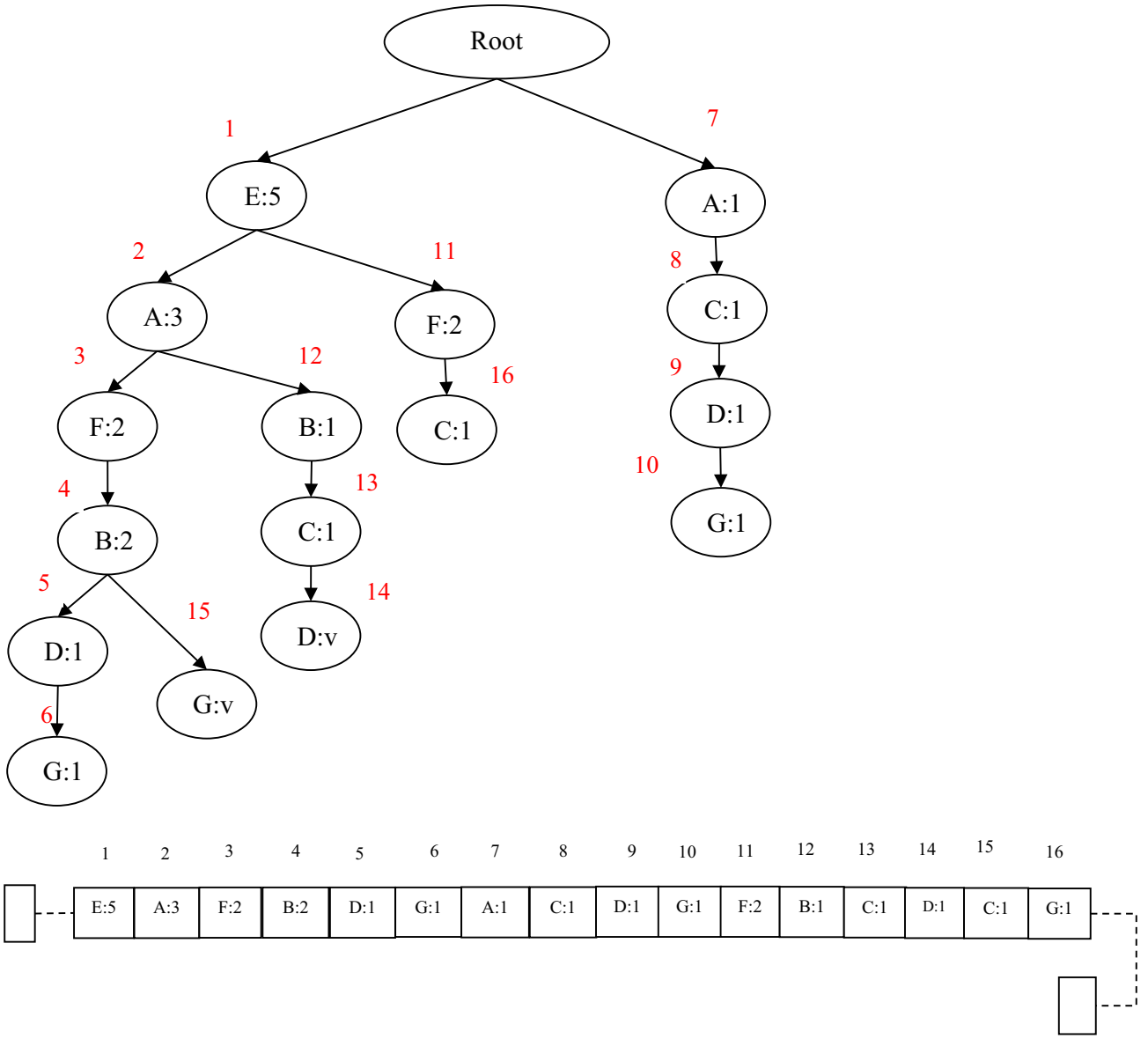

Fig. 1 UP-Tree node creation and memory allocation

exhausting the specific block in I/O conscious UP-Tree. Accordingly, when the system is necessary to load the blocks from the disk to main memory, then the block is loaded only once which reduces the page faults dramatically. This is briefly explained in the following algorithm which resolves the spatial and temporal locality problem in UP-Growth.

In the following IAUP-Growth (IAUP-Growth$^+$) algorithm, UP denotes the UP-Tree, x denotes the high utility itemsets, b denotes the block, H denotes the header table, s denotes the user specified minimum utility threshold, agg-count-list denotes the aggregated count list and is a binary operation on a set. It is calculated that combines two elements in the set. The UP-Growth and UP-Growth$^+$ algorithm are briefly explained in.

IAUP-Growth (IAUP-Growth$^+$) Algorithm

Initialize s, H

Step 1: IAUP-Growth(UP, H, s, alg)

{

Step 2: for each block b in UP-Tree do

{

Step 3: for each high utility item x in H do

{

Step 4: To traverse the nodes in b, follow the x node links from H

Step 5: Calculate the support count and set the criteria of minutil threshold.

}

}

Step 6: for each x in H do

{

Step 7: Aggregate conditional pattern base then collect for block b to produce agg-count-list

}

Step 8: for each block b UP do

{

Step 9: for each x in H do

{

Step 10: if all entries of $agg - count - list_x$ arenot lesser than s

{

Step 11: Build the conditional UP-Tree $UP_{x,s}$

}

}

}

Step 12: for each x in H do

{

Step 13: if $UP_{x,s}$ exists then

{

Step 14: if (alg==UP-Growth())

{

Step 15:UP-Growth$\left(UP_{x,s}, H_{UP_{x,s}}, s \triangleright H_{UP_{x,s}}\right)$ is the header table for $UP_{x,s}$

Step 16: else

Step 17: UP-Growth$^+\left(UP_{x,s}, H_{UP_{x,s}}, s \triangleright H_{UP_{x,s}}\right)$

}

}

}

}

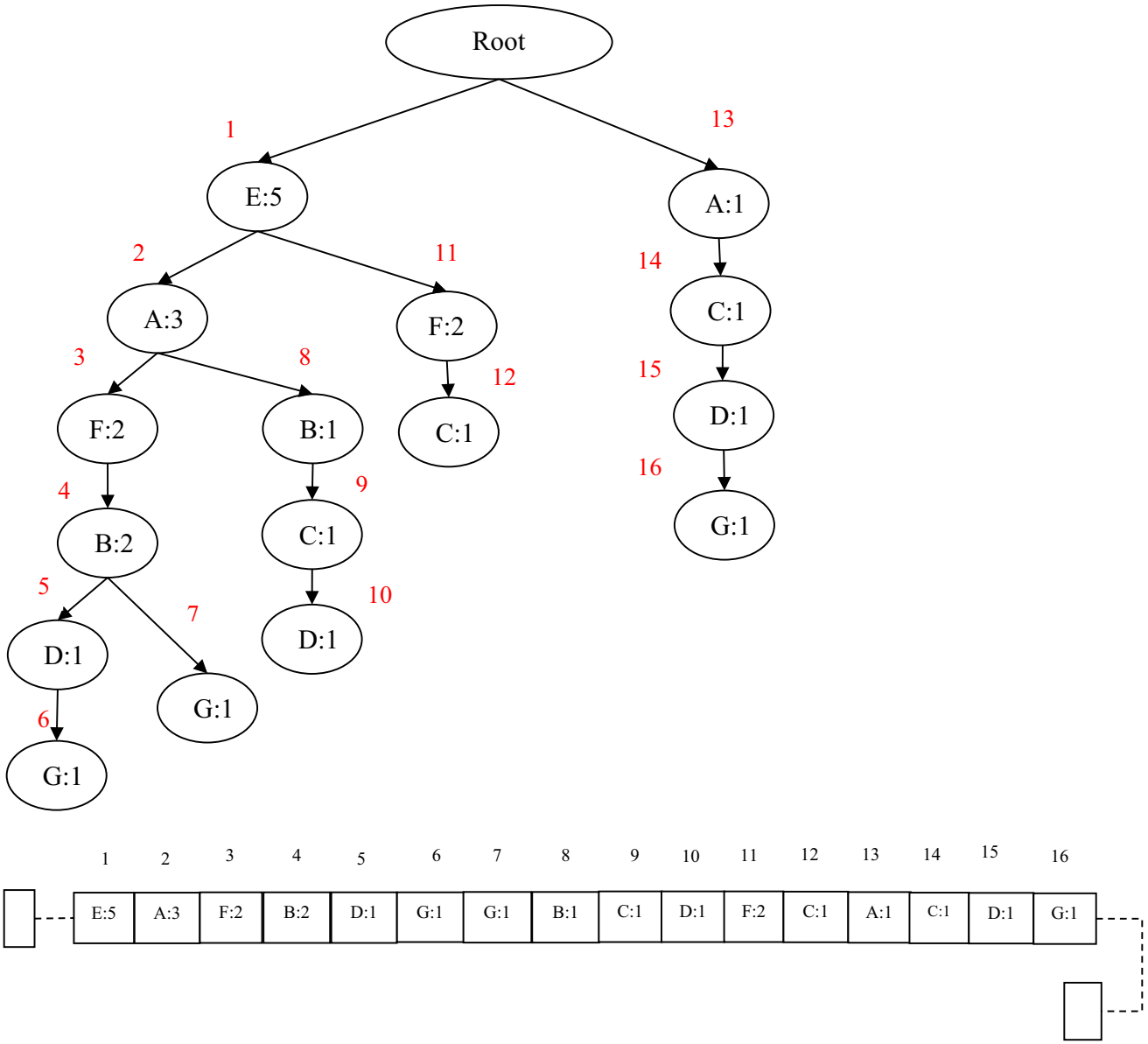

Fig. 2 Reorganizing UP-Tree in depth first order

3.3 Memory management strategy

The memory management is the core of the proposed IAUP-Growth. The memory management unit does all translation need to store the over flowed memory-based compact data structure like UP-Tree into secondary memory as sequential based file structures. Also it loads the data from file structure to preferred memory-based structure when the translation is necessary. In proposed memory management strategy, the tree translation unit is used as memory management unit in HUIM. This method transforms the memory-based UP-Tree which contains the UP-node list into secondary storage-based version of I/O conscious prefix tree. Initially, UP-Tree nodes are generated by searching UP-Tree in depth first manner with exclusive spatially aware node-Id. Then, the UP-Tree was split into blocks. The size of block must be lesser than or equal to block size of file for each node in the tree. The divided blocks are translated to I/O conscious prefix tree. Then it is stored in the disk as sequential file structures followed by node-IDs. Each node-Id is matches with unique pointer position in the file. Moreover, the tree node-Ids are generated I a single traversal and the tree blocks are translated into disc. Hence, it does not require any extra memory to store node Ids. The translated disc-based prefix tree and memory based UP-Tree is corresponds with individual Ids in the tree. UP-Tree node-Ids are saved into Disc

Table 1 Comparison of memory usage (MB) (MinU = 0.05%)

No. of transactions	T10I4d100k				Mushroom			
	UP-Growth	UP-Growth$^+$	IAUP-Growth	IAUP-Growth$^+$	UP-Growth	UP-Growth$^+$	IAUP-Growth	IAUP-Growth$^+$
2000	600	510	400	320	520	480	400	312
4000	720	590	510	450	624	514	450	380
6000	800	710	600	520	735	627	501	412
8000	950	812	703	652	800	722	652	567

Tree location Table (DTLT) structure which is performed in the final translation process. Thus, the boundary of blocks is stored in the file structure which is used for identification of particular prefix-tree. Furthermore, the whole header table and node-link Ids are mapped into Item Location Table (ILT) structure by memory management strategy.

4 Result and discussion

The effectiveness of proposed method for HUIM is performed based on memory utilization and execution time for HUIs. For the experimental purpose, there are two datasets are used called as T10I4d100k and mushroom dataset. The T10I4d100k came from IBM Data Generator and mushroom dataset taken from FIMI repository. The T10I4d100k dataset has 300,000 transactions, 1000 number of different items and 33.8 mean length of all transaction itemsets. Another dataset called mushroom dataset has 8124 numbers of transactions, 119 numbers of distinct items, 23 mean lengths of all transaction itemsets.

4.1 Memory usage

Memory usage refers the total amount computational storage required for mining HUIs from transaction database. The following Table 1 shows the memory usage with fixed minimum utility (0.05%) value for existing UP-Growth, UP-Growth$^+$ and proposed IAUP-Growth, IAUP-Growth$^+$ under different number of transactions.

From the Table 1, it is clear that the proposed IAUP-Growth, IAUP-Growth$^+$ algorithms has low memory usage for different number of transactions than the existing UP-Growth, UP-Growth$^+$ algorithms.

Figures 3 and 4, shows the comparison between proposed approaches of IAUP-Growth, IAUP-Growth$^+$ with existing UP-Growth, UP-Growth$^+$ algorithms based on memory usage for T10I4d100k dataset and mushroom dataset respectively. X axis denotes minimum utility (MinU %) and Y axis denotes the memory usage in terms of MB. A large number of utility itemsets are extracted as HUIs, while minimum utility value is low. So a lot of memory is required to store those HUIs and vice versa. In order to analysis, the minimum utility is taken in the X axis as well as memory usage is taken in the Y axis. From the figures, it is proven that the proposed IAUP-Growth$^+$ have less memory usage than the other methods.

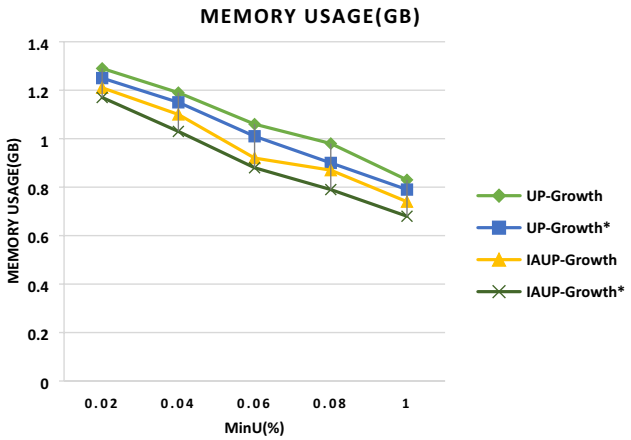

Fig. 3 Evaluation of memory usage for T10I4d100k

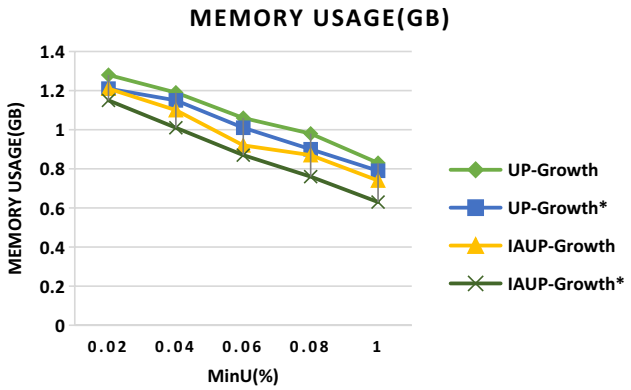

Fig. 4 Evaluation of memory usage for mushroom

Table 2 Comparison of running time (s) (MinU = 0.05%)

No. of transactions	T10I4d100k				Mushroom			
	UP-Growth	UP-Growth$^+$	IAUP-Growth	IAUP-Growth$^+$	UP-Growth	UP-Growth$^+$	IAUP-Growth	IAUP-Growth$^+$
2000	1.7	1.5	1.2	0.8	1.5	1.3	1	0.7
4000	2.3	2.1	2	1.8	2.2	1.9	1.7	1.4
6000	2.9	2.7	2.4	2.1	2.7	2.5	2.3	2
8000	3.4	3.1	2.8	2.6	3.2	2.9	2.7	2.4

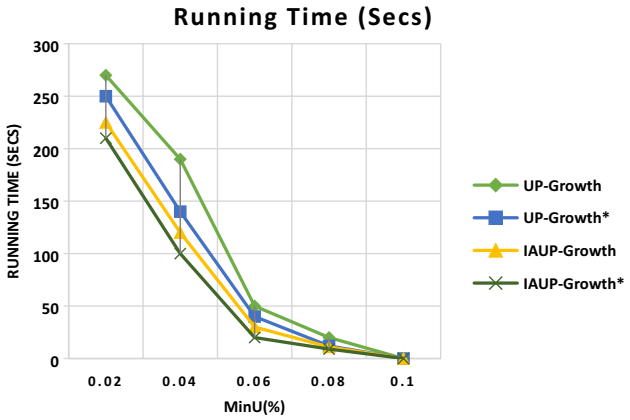

Fig. 5 Evaluation of Running Time for T10I4d100k dataset

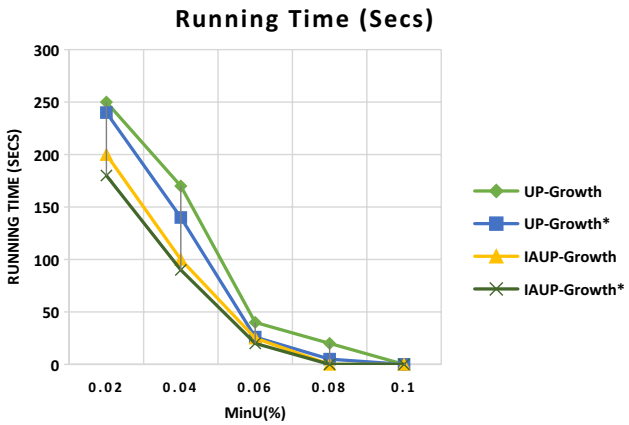

Fig. 6 Evaluation of Running Time for mushroom dataset

4.2 Running time

Running Time denotes the amount of time taken for mining the HUIs from the transaction database. The following Table 2 shows the running time with fixed minimum utility (0.05%) value for existing UP-Growth, UP-Growth$^+$ and proposed IAUP-Growth, IAUP-Growth$^+$ under different number of transactions.

From the Table 2, it is clear that the proposed IAUP-Growth, IAUP-Growth$^+$ requires less running time for different number of transactions than existing the UP-Growth, UP-Growth$^+$ algorithms.

Figures 5 and 6, shows the comparison of computational time between proposed IAUP-Growth, IAUP-Growth$^+$ and existing UP-Growth, UP-Growth$^+$ algorithms for T10I4d100k dataset and mushroom dataset respectively. The minimum utility (MinU %) represented by X axis and memory usage is represented by Y axis in terms of MB. From the figures, it is proved that the proposed IAUP-Growth$^+$ has less running time than the other methods.

5 Conclusion

IAUP-Growth and IAUP-Growth$^+$ are proposed to improve UP-Growth and UP-Growth$^+$ algorithm in HUIM. In UP-Tree, nodes are re-organized in depth first manner which resolves the spatial problem. Moreover, the computational part of UP-Growth and UP-Growth$^+$ rearranged by using a page blocking technique which resolves the temporal locality problem by proposed algorithms. A memory management strategy managed the memory of nodes of UP-Tree and by using this strategy there is no need to retain additional space for the node ID in each UP-node. The experiment is carried out in T10I4d100k and mushroom datasets and proves the effectiveness in terms of space requirement and execution time.

References

Adnan M, Reda A (2011) A bounded and adapted memory-based approach and adaptive memory-based approach to mine frequent patterns from very large databases. IEEE Trans Syst Man Cybern 41(1):154–172

Anitha P, Kaarthick B (2019) Oppositional based Laplacian grey wolf optimization algorithm with SVM for data mining in intrusion

detection system. J Ambient Intell Humaniz Comput. https://doi.org/10.1007/s12652-019-01606-6
Chu CJ, Tseng VS, Liang T (2008) An efficient algorithm for mining temporal high utility itemsets from data streams. J Syst Softw 81(7):1105–1117
Gan W, Lin JCW, Fournier-Viger P, Chao HC, Hong TP, Fujita H (2018) A survey of incremental high-utility itemset mining. Wiley Interdiscip Rev Data Min Knowl Discov 8(2) http://fimi.ua.ac.be/data/mushroom.dat
Joshi M, Patel M (2014) A survey on high utility itemset mining using transaction databases. IJCSIT International Journal of Computer Science and Information Technologies, ISSN, 0975-9646
Krishnamoorthy S (2018a) Efficiently mining high utility itemsets with negative unit profits. Knowl Based Syst 145:1–14
Krishnamoorthy S (2018b) Efficient mining of high utility itemsets with multiple minimum utility thresholds. Eng Appl Artif Intell 69:112–126
Lin CW, Hong TP, Lu WH (2011) An effective tree structure for mining high utility itemsets. Expert Syst Appl 38(6):7419–7424
Lin CW, Lan GC, Hong TP (2012a) An incremental mining algorithm for high utility itemsets. Expert Syst Appl 39(8):7173–7180
Lin MY, Tu TF, Hsueh SC (2012b) High utility pattern mining using the maximal itemset property and lexicographic tree structures. Inf Sci 215:1–14
Lin JCW, Gan W, Fournier-Viger P, Hong TP, Tseng VS (2016) Fast algorithms for mining high-utility itemsets with various discount strategies. Adv Eng Inform 30(2):109–126
Lin JCW, Zhang J, Fournier-Viger P, Hong TP, Zhang J (2017) A two-phase approach to mine short-period high-utility itemsets in transactional databases. Adv Eng Inform 33:29–43
Manike C, Hari O (2014) Sliding-window based method to discover high utility patterns from data streams. Comput Intell Data Min 3:173–184
Memar M, Deypir M, Sadreddini MH, Fakhrahmad SM (2012) An efficient frequent itemset mining method over high-speed data streams. Comput J 55(11):1357–1366
Tseng VS, Shie BE, Wu CW, Philip SY (2013) Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans Knowl Data Eng 25(8):1772–1786
Tseng VS, Wu CW, Fournier-Viger P, Philip SY (2016) Efficient algorithms for mining top-k high utility itemsets. IEEE Trans Knowl Data Eng 28(1):54–67
Wang L, Meng J, Xu P, Peng K (2018) Mining temporal association rules with frequent itemsets tree. Appl Soft Comput 62:817–829
Zhang L, Fu G, Cheng F, Qiu J, Su Y (2018) A multi-objective evolutionary approach for mining frequent and high utility itemsets. Appl Soft Comput 62:974–986
Zida S, Fournier-Viger P, Lin JCW, Wu CW, Tseng VS (2015) EFIM: a highly efficient algorithm for high-utility itemset mining. In: Mexican International Conference on Artificial Intelligence. Springer, Cham, pp 530–546