Hierarchical Adaptive Gap-Run TID Compression for Large-Scale Frequent Itemset Mining

XIN DAI[1], CHENJIAO LIU[2], XUE HAO[3], QICHEN SU[4]
Faculty of Computing, Universiti Teknologi Malaysia (UTM), Jalan Iman, 81310 Skudai, Johor Bahru, Malaysia[1,3,4]
School of Mathematics and Big Data, Guizhou Education University, 115 Gaoxin Road, Guiyang, Guizhou 550018, China[1]
Guizhou Key Laboratory of Artificial Intelligence and Brain-inspired Computing,
Guizhou Education University, 115 Gaoxin Road, Guiyang, Guizhou 550018, China[2]

***Abstract*—Frequent itemset mining faces the prominent problems of high storage space requirements and low efficiency in a large-scale transaction data environment.The traditional Eclat algorithm usually uses bitmap or sparse array to represent a single transaction identifier (TID), which is difficult to adapt to the changes of dense and sparse transaction data at the same time; Although the existing hybrid representation schemes can partly alleviate this problem, the additional computational overhead caused by frequent data structure switching and the inherent space waste of bitmap structure have not been fundamentally solved. Therefore, this article proposes a HiAGL-FIM algorithm based on Hierarchical Adaptive Gap Run Transaction Identifier List (HAGL-TID). This algorithm adaptively selects Gap List or Run List for transaction identifier encoding through continuity ratio, and designs an efficient TID intersection operation method, completely eliminating dependence on bitmap structure and effectively reducing memory consumption and intersection calculation overhead. The experimental results show that HiAGL-FIM has significant advantages in terms of running time, memory usage, and data scalability compared to classical algorithms such as Eclat, FP Growth, and dEclat. Especially when the transaction data scale reaches millions, it shows a more significant performance improvement, demonstrating the effectiveness and practical value of our method.**

***Keywords*—Frequent itemset mining; pure Eclat; Hierarchical Adaptive Gap-Run List (HAGL-TID); large-scale transaction data**

I. Introduction

With the rapid development of big data technology and artificial intelligence, frequent itemset mining (FIM), as one of the core problems in the field of data mining, has received widespread attention and in-depth research in recent years [1]. The goal of frequent itemset mining is to identify frequently occurring combinations of items from transactional datasets, which plays an important role in many application fields such as market basket analysis, recommendation systems, bioinformatics, and anomaly detection[2]. Among the existing frequent itemset mining methods, Apriori algorithm and FP Growth algorithm are widely used due to their clear theoretical basis and simple implementation [3]. However, the Apriori algorithm has low efficiency due to its multiple scans of the database and frequent generation of candidate sets; Although FP Growth avoids frequent database scans, it may incur significant memory overhead when building FP trees [4]. The Eclat algorithm records itemset support information through a Transaction Identifier List (TID), significantly reducing the number of database scans and achieving good efficiency [5]. However, traditional Eclat algorithms usually only use a single TID representation method (such as sparse arrays or bitmaps), which fails to simultaneously consider the sparse and continuous features of transaction data, resulting in low memory efficiency and significant optimization space for performance when applied in real datasets [6].

In recent years, researchers have proposed various hybrid TID representation strategies around "density adaptation". The typical approach is to dynamically switch between bitmap, array, or N-list/NegNodeset structures based on local transaction density, such as Tseng et al.'s FPL/TPL adaptive framework [7], Poovan et al.'s multi-threaded NegNodeset+N-list hybrid framework [8], and Bashir and Baig's HybridMiner algorithm [9]. These methods have achieved a good trade-off between time and space on small and medium-sized datasets, but there are still two major bottlenecks: firstly, the threshold for structure switching depends on empirical settings, which requires repeated re-encoding of TID in the face of frequent fluctuations in transaction density in streaming scenarios, resulting in significant CPU and cache overhead; secondly, even though bitmap is efficient in dense scenes, its storage requirements still increase linearly with transaction volume, resulting in a large amount of empty space waste when sparsity increases. The Ramp algorithm has conducted a detailed analysis on this [10]. Furthermore, the closed set miner CICLAD, which targets data streams of millions or more, also reported the issue of high peak memory consumption in traditional bitmap/node-set methods [11]. Therefore, how to completely eliminate the reliance on bitmap representations while enabling finer-grained continuity modeling, and how to maintain stable mining efficiency in environments with sparse remains a major gap in current mixed TID research. This issue is mainly reflected in two aspects: first, existing hybrid approaches depend heavily on frequent structure switching and empirically set thresholds, which often require repeated re-encoding under fluctuating transaction densities and lead to additional CPU and cache overhead; second, the storage cost of bitmap structures increases linearly with data size, causing severe space waste in sparse regions and making it difficult to meet the performance demands of large-scale transactional datasets.

In response to the above challenges, this article proposes a new frequent itemset mining algorithm — HiAGL-FIM. This algorithm innovatively introduces a transaction-ID representation method based on Hierarchical Adaptive Gap Run List (HAGL-TID). The HiAGL-FIM algorithm does not rely on traditional bitmap structures; instead, it adaptively selects Gap or Run lists for compression storage and intersection operations according to

the continuity of transaction data, thereby significantly reducing memory consumption and improving the efficiency of frequent itemset mining. HiAGL-FIM avoids repeated re-encoding in hybrid approaches through one-time adaptive compression modeling based on continuity ratio after a single scan. It also introduces structure-specific intersection operators and a unified support calculation formula for Run–Run, Gap–Gap, and Run–Gap cases, which improve time efficiency and memory compactness under the premise of completely removing bitmap dependency.This article will provide a detailed introduction to the theoretical basis, specific implementation methods, and experimental verification results of the HiAGL-FIM algorithm in subsequent chapters, in order to demonstrate its excellent performance and wide applicability on different types of datasets.

The remainder of this paper is organized as follows: Section II reviews related work on frequent itemset mining and hybrid TID representations. Section III details the proposed HiAGL-FIM algorithm, including its structure, generation process, and complexity analysis. Section IV presents experimental results and performance comparisons with state-of-the-art algorithms. Section V concludes the paper and discusses future research directions.

II. Related Work

Frequent itemset mining algorithms have undergone decades of development, and researchers have proposed a series of classic algorithms and optimization strategies, mainly divided into two categories: horizontal data formats and vertical data formats. The algorithms based on horizontal data formats are represented by the classic Apriori [12] and FP Growth [3]. Among them, the Apriori algorithm first proposed a strategy of generating candidate itemsets layer by layer and pruning them, effectively reducing the search space. However, in large-scale datasets, a large number of candidate sets will be generated, leading to serious performance bottlenecks; FP Growth compresses data through a tree structure (FP Tree), avoiding the generation of candidate sets and improving mining efficiency. However, in the process of building and maintaining FP Tree, it still faces significant memory and time overhead, especially in high-dimensional data and high workload scenarios [13].

The vertical data format algorithms are represented by Eclat [5], Charm [14], and dEklat [15]. The Eclat algorithm represents the transaction support information of each itemset through a transaction identifier list (TID), greatly reducing the number of database scans and suitable for fast mining of large-scale transaction data; Charm further utilizes the characteristics of closed itemsets to optimize the vertical mining process and obtain closed frequent itemsets in a more efficient way; DEklat, on the other hand, reduces the overhead of intersection operations between TID lists through an extended Diffset structure, significantly improving mining efficiency [16]. However, these classic vertical formatting methods all default to using a single type of data structure to represent transaction support information, making it difficult to simultaneously meet performance requirements under different data densities.

To address this issue, researchers have recently begun exploring adaptive data representation structures to adapt to changes in data distribution. Tseng et al. proposed the FPL/TPL framework, which dynamically switches between sparse array and bitmap representations based on the local density of transaction data. Although it improves computational efficiency, the additional overhead caused by data structure transformation operations is particularly prominent in scenarios with frequent changes in data density [7]. Similarly, the NegNodeset+N-list hybrid storage method proposed by Poovan et al. also faces the burden of structural transformation in density changing scenarios, especially with poor performance when the transaction data scale expands [17]. In addition, the HybridMiner algorithm developed by Bashir et al. uses a combination of bitmap and array representations. Although it has improved mining efficiency, it still cannot fundamentally solve the problem of space waste caused by the linear growth of bitmap storage with transaction size [9].

To reduce the high memory requirements of bitmap representation structures, the Ramp algorithm has designed an efficient bit vector projection technique, which to some extent reduces the redundant storage of bitmaps. However, it still generates a large amount of redundant space when transaction data density is sparse [10]. For data stream mining scenarios, the CICLAD algorithm proposed by Martin et al. utilizes compact data structures to improve memory usage efficiency. However, in high-density and volatile scenarios, this algorithm still suffers from reduced computational performance due to frequent data structure adjustments [11]. The existing frequent itemset mining algorithms have more or less the following bottlenecks: 1) a single data representation structure is difficult to accommodate the different density features of transaction data, resulting in insufficient adaptability of the algorithm. 2) Although mixed data representation methods can partially solve adaptability problems, the additional computational burden brought by data structure transformation limits their application effectiveness when transaction density frequently changes. 3) Although traditional bitmap representation is efficient in dense scenarios, it still faces significant memory consumption issues when processing datasets with transaction scales of tens of millions or even billions. Therefore, it is particularly important to design a frequent itemset mining algorithm that does not require bitmap and can automatically adapt to transaction data of different densities. The HiAGL-FIM algorithm proposed in this article adopts a hierarchical adaptive Gap Run transaction identifier representation structure, which completely avoids the shortcomings of bitmap representation, effectively balances the algorithm's space occupation and computational efficiency, and further promotes the practical application of frequent itemset mining in large-scale transaction data.

III. HiAGL-FIM Algorithm

A. Algorithm Framework

The design goal of HiAGL-FIM is to introduce an adaptive transaction identifier compression structure in the modeling stage, combined with structure-specialized intersection operations and layered scheduling mechanisms, to maintain efficiency and stability in large-scale environments with mixed dense and sparse transactions. To achieve this objective, the overall process is divided into five tightly connected phases.

1) Phase 1: Transaction scanning and inverted TID construction: The transaction database $D = \{\tau_1, \ldots, \tau_n\}$ is

scanned linearly to construct the transaction identifier sequence $T_i = \{t_1 < \cdots < t_{|T_i|}\}$ for each item $i \in \mathcal{I}$, while also performing global support counting. This phase completes the initialization of the inverted structure, laying the foundation for subsequent compression and pruning.

2) Phase 2: Continuity-driven adaptive compression: The structural choice is determined by the continuity ratio $\phi(T_i)$:

$$\phi(T_i) = \frac{\sum_{k=1}^{|T_i|-1} \mathbf{1}[\,t_{k+1} - t_k = 1\,]}{|T_i| - 1} \tag{1}$$

When $\phi(T_i) \geq \rho_0$, T_i is compressed into a Run-List; otherwise, it is compressed into a Gap-List. The compression operation is performed once during the modeling phase and remains stable throughout mining, avoiding the overhead of frequent structure switching in mixed-density scenarios.

3) Phase 3: Frequent 1-itemset extraction and buffer partitioning: For each T_i, support is calculated as the sum of segment lengths if stored as a Run-List, or as the element count if stored as a Gap-List. Items with $\sup(i) \geq \theta$ are identified as frequent 1-itemsets and included in the initial result set. Simultaneously, frequent 1-itemsets are partitioned into high-, medium-, and low-frequency buffers, establishing a layered scheduling mechanism to enhance cache locality and prepare for subsequent intersection operations.

4) Phase 4: Structure-specialized intersection and pruning: Candidate itemset expansion adopts a depth-first strategy, and support calculation relies on structure-specialized operators:

$$\operatorname{supp}(A \cap B) = \begin{cases} \sum_{(s,\ell)\in(R_A \cap R_B)} \ell, & \text{Run} \times \text{Run}, \\ |G_A \cap G_B|, & \text{Gap} \times \text{Gap}, \\ |\, G_{\text{gap}} \cap \operatorname{Expand}(R_{\text{run}})\,|, & \text{Run} \times \text{Gap}. \end{cases} \tag{2}$$

Here, $R_\star$ denotes the Run-List and $G_\star$ denotes the Gap-List. During the intersection process, the upper bound of support is used for short-circuit pruning. If the upper bound falls below the threshold θ, the computation is terminated immediately to reduce unnecessary overhead.

5) Phase 5: Recursive expansion and termination: Candidate itemsets that meet the support threshold are recursively expanded, with their TID structures retained as new prefixes. The depth-first expansion strategy ensures a low memory footprint and strong path locality, enabling the algorithm to scale effectively on large datasets. Recursive traversal also allows heuristic pruning methods to be combined, further improving overall execution efficiency.

B. Principle

HiAGL-FIM (Hierarchical Adaptive Gap-Run List-based Frequent Itemset Miner) is an efficient frequent itemset mining algorithm for large-scale transaction data. Its core lies in constructing a hierarchical adaptive transaction identifier representation structure—HAGL-TID (Hierarchical Adaptive Gap-Run Transaction Identifier List). This structure breaks away from the reliance of traditional Eclat and its variants on bitmaps and static sparse arrays. It offers high compression capabilities and density adaptability, effectively improving mining efficiency and resource utilization in both dense and sparse transaction data environments.

The overall process of HIAGL-FIM follows the algorithm framework of "one-time modeling, adaptive compression, structure-specialized intersection, recursive deep expansion". First, the algorithm scans the transaction database $\mathcal{D} = \{t_1, t_2, \ldots, t_n\}$ in sequence, and constructs the transaction identifier list T_i for each item $i \in I$. Then, the algorithm measures the local density of the TID list by defining the continuity ratio $\phi(T_i)$ as in Eq. 1. If adjacent TIDs are continuous, the function evaluates to 1. When $\phi(T_i) \geq \rho_0$, the transaction sequence exhibits high continuity. The algorithm compresses it into a *Run-List* structure, recording the starting position and length of each continuous segment; otherwise, it preserves the sequence as an ascending sparse integer array (i.e. a *Gap-List*). This structural compression is completed during the initial database scan and remains static throughout the mining phase, eliminating the re-encoding overhead inherent in traditional dynamic hybrid structures.

To further improve access efficiency, HIAGL-FIM introduces the support-layered buffer mechanism, which divides all TID structures into high-frequency, intermediate-frequency, and low-frequency levels according to their support values, and manages them separately to strengthen memory locality.

During the mining phase, the algorithm uses a depth-first search (DFS) strategy to recursively expand candidate itemsets. In each expansion process, the pre-built inverted TID structure is used to locate the transaction sets between candidate itemsets, and the structure-specialized intersection operator is selected to efficiently calculate the intersection support according to the TID structure combination, including:

- Run × Run: sliding window matching for continuous segments;
- Gap × Gap: sparse array processing with double-pointer or vectorization;
- Run × Gap: fast positioning and screening through interval judgment and mask operations.

When the support of the intersection result is lower than the preset minimum threshold θ, the algorithm prunes immediately to avoid invalid expansion. Through this structure-driven recursive expansion mechanism, HIAGL-FIM maintains stable mining performance in dense and sparse mixed scenarios, while significantly reducing memory consumption and intersection computational complexity. The overall process is shown in Fig. 1.

C. Generation Process

The complete generation process of HIAGL-FIM algorithm includes five stages: transaction preprocessing and inverted structure construction, continuity compression modeling, frequent 1-itemset extraction and buffer partition, candidate itemset intersection support calculation, and recursive expansion to generate frequent itemsets. The process is built around the HAGL-TID structure, forming a seamless mapping from the original transaction to efficient frequent itemsets.

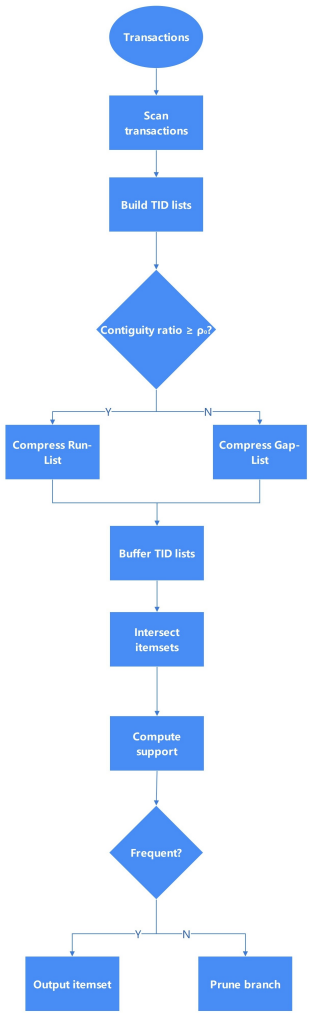

Fig. 1. The overall process of HIAGL-FIM.

1) Transaction scanning and inverted TID structure construction: Firstly, the algorithm performs a single round linear scan on the input transaction database $D = \{\tau_1, \tau_2, \ldots, \tau_n\}$. For each transaction τ_j, write the transaction number j of all items i into its corresponding inverted TID list T_i. At the same time, the algorithm counts the global support of each item to prepare for the subsequent compression modeling and pruning.

2) Modeling of transaction ID compression driven by continuity: After the TID list is built, the algorithm sets the transaction number sequence for each item $T_i = \{t_1, t_2, \ldots, t_{|T_i|}\}$ and conducts continuity assessment by calculating its continuity ratio as in Eq. 1. This ratio describes the intensity of transaction distribution.

According to the continuity threshold ρ_0 (e.g. 0.25), the algorithm determines the structural representation:

- If $\phi(T_i) \geq \rho_0$, it will be compressed into a Run-List structure, recording the starting position and length of each continuous transaction segment.
- Otherwise, the Gap-List structure is retained, representing all TID numbers as an ascending sparse array.

The compression operation is performed only once, and the structure type remains unchanged throughout the mining process. This avoids the problem of heavy recoding of hybrid structures in density fluctuation scenarios, improving stability and execution efficiency.

- Frequent 1-itemset extraction and hierarchical buffer partition: The algorithm then calculates the support of TID structure T_i of each item i:
  - For Run-List: support is the sum of the lengths of all sections.
  - For Gap-List: support is the number of elements in the array.

If support $\sup(i) \geq \theta$ (minimum support threshold), item i is determined to be a frequent 1-itemset. All frequent 1-itemsets are included in the result set and serve as the starting point for subsequent candidate extensions.

At the same time, the algorithm divides its TID structure into high-frequency, intermediate-frequency and low-frequency buffers according to the size of each support, and uses a hierarchical scheduling mechanism to improve access locality and cache hit rate, providing structured support for the intersection calculation stage.

- Intersection calculation and support determination of candidate itemsets

Starting from frequent 1-itemsets, the algorithm uses depth-first search (DFS) to recursively construct candidate itemsets. For two frequent k-itemsets A and B with the same prefix, if their first $k-1$ items are identical, the connection extension is performed to generate the candidate itemset $C = A \cup B$.

To calculate the support of candidate itemset C, the algorithm calls the structure-specific intersection operator, following the unified formula given in Eq. 2.

The formula provides the mathematical basis for support determination and subsequent pruning strategies and recursive expansion. To improve efficiency, the algorithm introduces a short-circuit mechanism during intersection—when the maximum possible support is known to be below θ before completing intersection, the calculation terminates immediately and skips subsequent extensions, effectively reducing unnecessary computational overhead.

- Recursive extension and termination of frequent itemsets.

For each candidate itemset C that passes the support determination, the algorithm recursively expands the next-level candidate set as a new prefix, and continues to perform connection, intersection and pruning until no more itemsets meeting the support requirements can be generated. The depth-first expansion strategy has low memory footprint and path locality, and shows good scalability in large-scale transaction data. In addition, recursive traversal allows the combination of heuristic pruning methods to further improve execution efficiency.

Through the above five stages, HIAGL-FIM not only ensures the high compression rate and structural stability of

the algorithm, but also realizes the efficient recursive mining of frequent itemsets. Especially in the aspect of support calculation, a unified and scalable mining engine is constructed based on the structure-specific intersection strategy of Run-List and Gap-List structures and the formal support determination formula. This algorithm does not need to rely on the bitmap structure, and avoids the frequent switching of hybrid structures, comprehensively improving adaptability in large-scale dense-sparse mixed transaction environments. The pseudo-code is as follows, and experimental verification will be carried out in the following chapters.

Algorithm 1 HiAGL-FIM: Hierarchical Adaptive Gap-Run List Frequent Itemset Miner

Require: Transaction database T, minimum support fraction $minsup\_fr$
Ensure: Frequent itemsets F
Load T from file
Initialize *item_to_tidlist* as empty dictionary
for each transaction T_i in T **do**
  for each item x in T_i **do**
    Append i to *item_to_tidlist*[x]
  end for
end for
for each item x in *item_to_tidlist* **do**
  Convert TID list to NumPy array
  if continuity_ratio $\geq \rho_0$ **then**
    Compress to Run-List format
  else
    Compress to Gap-List format
  end if
end for
Compute $minsup \leftarrow \lceil minsup\_fr \times |T| \rceil$
Filter itemsets with support $\geq minsup$ into L_1
$F \leftarrow L_1$, $level_k \leftarrow L_1$, $k \leftarrow 1$
while $level_k \neq \emptyset$ **do**
  $next\_level \leftarrow \emptyset$
  Group $level_k$ by common $(k-1)$-prefix
  for each group G **do**
    Sort G lexicographically
    for $i \leftarrow 0$ to $|G|-1$ **do**
      for $j \leftarrow i+1$ to $|G|-1$ **do**
        $A \leftarrow G[i]$, $B \leftarrow G[j]$
        $C \leftarrow A \cup B$
        if $C \notin F$ **then**
          Compute $TID(C) \leftarrow TID(A) \cap TID(B)$ using Eq. 2
          if $support(C) \geq minsup$ **then**
            Store $TID(C)$ (auto-adaptive)
            Add C to F and $next\_level$
          end if
        end if
      end for
    end for
  end for
  $level_k \leftarrow next\_level$
  $k \leftarrow k+1$
end while
return F

D. Structural Optimization and Complexity Analysis

To comprehensively improve the performance of HiAGL-FIM in large-scale transaction data processing, the algorithm introduces various optimization mechanisms in the structure design and execution process. This section systematically describes these optimization strategies, analyzes their time and space complexity across different modules, and highlights the algorithm's scalability and practicability.

1) Compression coding and pruning acceleration mechanism: HiAGL-FIM compresses all TID lists at once through the continuity ratio $\phi(T_i)$ as defined in Eq. 1 during the modeling stage, avoiding the recoding overhead caused by frequent hybrid structure switching in traditional methods. The compression process has linear time complexity $O(n)$, where n is the number of TIDs.

During candidate itemset expansion, the algorithm introduces support upper-bound estimation combined with structure-specialized intersection operators (Eq. 2) to enable "intersection short-circuit" judgment. If the current intersection cannot reach the minimum support threshold θ, immediate pruning avoids subsequent invalid calculations.

2) Buffer hierarchical scheduling: TID structures with different support levels are allocated to high-frequency, intermediate-frequency, and low-frequency buffers for batch management and cache-affinity scheduling. This mechanism significantly reduces random TID structure accesses during intersection operations, improving memory locality and reducing access latency.

- Time and space complexity analysis
  - Initialization phase (build TID table): Single transaction scan with time complexity $O(n \cdot L)$ and space complexity $O(n)$, where n is the transaction count, L is the average transaction length, and m is the number of distinct items.
  - Structure compression stage: Calculate $\phi(T_i)$ for each item and classify structures. Complexity $O(n \cdot m)$, where m is the average support.
  - Intersection calculation phase:
    - `Run × Run`: optimal complexity $O(p+q)$ with p, q segment counts;
    - `Gap × Gap`: complexity $O(m+n)$, reducible to $O(\min(m,n))$ with hash optimization;
    - `Run × Gap`: approximately $O(n \log p)$ using binary/mask matching.
  - Recursive expansion stage: DFS approach prevents candidate explosion. Actual depth depends on frequent itemset density. Worst-case remains exponential $O(2^n)$, but pruning mechanisms effectively control expansion scale.

IV. Result and Discussion

All experiments were conducted in the same computing environment. The experimental platform was a personal computer equipped with 2.2 GHz Intel Core i5 processor, 8 GB memory and 64-bit Windows 10 Professional operating system. To ensure comparability and consistency of results, this paper implements all algorithms using Python without loading any GPU acceleration components during experiments, guaranteeing authenticity and reliability of measured performance metrics.

For experimental validation, we selected the classic sparse transaction database `T10I4D100K` as test dataset, containing 100,000 transactions with an item set size of 870. Each transaction averages approximately 10 items, exhibiting typical low-density characteristics. This dataset is widely adopted in frequent itemset mining research and particularly suitable for evaluating algorithm efficiency and resource consumption in sparse data scenarios.

To comprehensively assess HiAGL-FIM's performance advantages, we compare against two mainstream algorithms: FP-Growth and Eclat. FP-Growth avoids explicit candidate generation by constructing compressed FP-Tree structures, representing one of the most widely-used horizontal format algorithms. Eclat employs vertical TID representation and computes support through intersection operations, offering good execution efficiency but with performance highly dependent on underlying TID structure design. For fair comparison, all algorithms run independently under identical input data and parameter configurations, with key performance metrics recorded at five different support thresholds.

In this paper, the support threshold σ is set as $\{0.006, 0.005, 0.004, 0.003, 0.002\}$ to simulate the actual mining demand change from high support to low support. Under each support setting, the running time (in seconds) and the peak physical memory usage (in MB) required by the algorithm to complete the mining of all frequent itemsets are measured respectively. The memory statistics are collected in real time through the psutil library. The evaluation results are shown in Fig. 2 and Fig. 3, respectively corresponding to the running time change and memory consumption trend under different support levels.

As can be seen from Fig. 2, as the support threshold σ decreases from 0.006 to 0.002, the running time of *HiAGL-FIM* gradually increases from 10.9 seconds to 18.7 seconds. Although the computational workload increases due to the exponential expansion of candidate itemsets at a lower threshold, *HiAGL-FIM* consistently outperforms Eclat and FP-growth across the entire threshold range. Specifically, the average acceleration of *HiAGL-FIM* is 31.7% over Eclat and 45.9% over FP-growth, demonstrating its superior time efficiency. This performance advantage stems from its layered compression mechanism, which enables fast and structure-aware TID intersection.

Fig. 3 further highlights the significant memory-saving advantage of the Hierarchical Adaptive Gap-Run List (HAGL). The memory footprint of FP-growth and Eclat remains relatively high—ranging from 383.58 MB to 421.43 MB and 136.85 MB to 142.02 MB respectively—while the memory usage of *HiAGL-FIM* remains compact, ranging only from 4.76 MB to 13.15 MB. At the lowest threshold $\sigma = 0.002$, the memory consumption

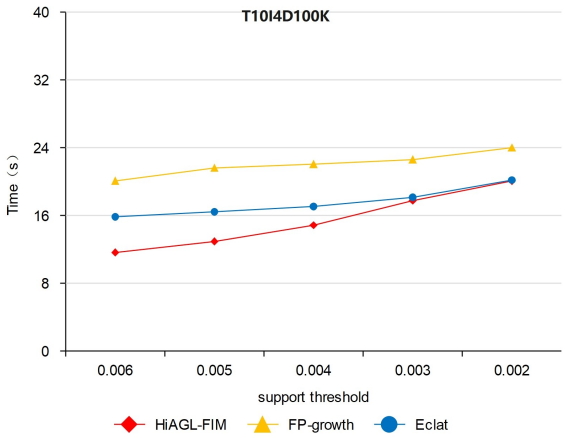

Fig. 2. Execution time comparison under different support thresholds on T10I4D100K.

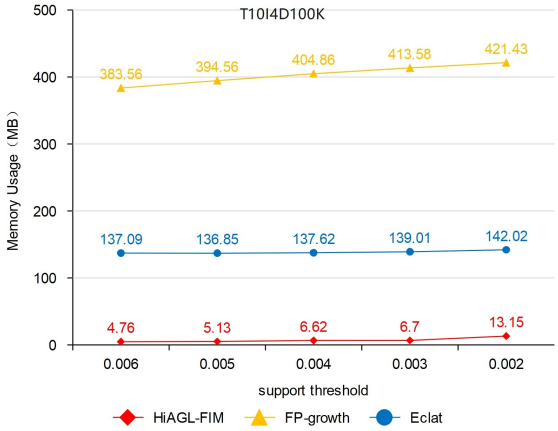

Fig. 3. Memory usage comparison under different support thresholds on T10I4D100K.

of *HiAGL-FIM* is just 13.15 MB, accounting for only 9.2% of Eclat and 3.1% of FP-growth.

This substantial improvement is attributed to the use of the HAGL-TID representation, which dynamically adapts to the density of TID sequences by switching between compact Run-List encoding for continuous intervals and sparse Gap-List encoding for discrete elements. In addition, the algorithm employs specialized intersection operators for each structure pair (Run-Run, Gap-Gap, Run-Gap), thereby minimizing redundant operations in the candidate generation process. As a result, the overall space complexity is effectively reduced to $\mathcal{O}(|G| + |R|)$, where $|G|$ and $|R|$ represent the size of the Gap and Run segments, respectively. This lightweight design enables *HiAGL-FIM* to efficiently mine frequent itemsets from large-scale transaction datasets in a single-threaded Python environment.

V. Conclusion and Future Work

In this paper, a new frequent itemset mining algorithm, HiAGL-FIM, is proposed. Its core lies in the design of a Hierarchical Adaptive Gap-Run TID (HAGL-TID) structure.

Through continuity-driven transaction compression modeling, structure-specialized intersection operations, and hierarchical buffer scheduling mechanisms, the algorithm achieves efficient processing of dense–sparse mixed environments in large-scale transaction databases. Experimental results demonstrate that, compared with the classical Eclat and its variants, HiAGL-FIM exhibits significant advantages in execution time and memory consumption, verifying the effectiveness and scalability of the proposed method in large-scale data scenarios.

However, the proposed method still has certain limitations. The current study mainly focuses on validation using static transaction databases and does not fully address dynamic updates in streaming data environments. In addition, the implementation is based on a single-machine architecture without distributed or parallel optimization, leaving room for improvement in handling ultra-large-scale datasets.

Future research can be carried out in the following directions: 1) extending HiAGL-FIM to streaming environments by incorporating sliding windows and incremental update strategies to achieve efficient mining of dynamic data; 2) introducing distributed computing frameworks and GPU acceleration to enhance the algorithm's processing capacity in large-scale scenarios; and 3) exploring the integration of frequent itemset mining with concept drift detection, pattern evolution, and other tasks, further expanding the application potential of the algorithm in intelligent decision-making and real-time data analysis.

REFERENCES

[1] W. I. D. Mining, *Data Mining: Concepts and Techniques*. San Francisco, CA: Morgan Kaufmann, 2006.

[2] C. C. Aggarwal, *Data Mining: The Textbook*, vol. 1. New York: Springer, 2015.

[3] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, 2000.

[4] G. Grahne and J. Zhu, "Efficiently using prefix-trees in mining frequent itemsets," in *Proc. FIMI*, 2003, p. 65.

[5] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, 2000.

[6] M. R. Al-Bana, M. S. Farhan, and N. A. Othman, "An efficient spark-based hybrid frequent itemset mining algorithm for big data," *Data*, vol. 7, no. 1, p. 11, 2022.

[7] F. C. Tseng, "An adaptive approach to mining frequent itemsets efficiently," *Expert Syst. Appl.*, vol. 39, no. 18, pp. 13,166–13,172, 2012.

[8] J. S. P. Poovan, D. A. Udupi, and N. V. S. Reddy, "A multithreaded hybrid framework for mining frequent itemsets," *Int. J. Elect. Comput. Eng.*, vol. 12, no. 3, pp. 3249–3264, 2022.

[9] S. Bashir and A. R. Baig, "HybridMiner: Mining maximal frequent itemsets using hybrid database representation approach," in *Proc. INMIC*, 2005, pp. 1–7.

[10] S. Bashir and A. R. Baig, "Ramp: Fast frequent itemset mining with efficient bit-vector projection technique," *arXiv:0904.3316*, 2009.

[11] T. Martin, G. Francoeur, and P. Valtchev, "CICLAD: A fast and memory-efficient closed itemset miner for streams," in *Proc. KDD*, 2020, pp. 1810–1818.

[12] L. Huang, H. Chen, X. Wang, and G. Chen, "A fast algorithm for mining association rules," *J. Comput. Sci. Technol.*, vol. 15, pp. 619–624, 2000.

[13] J. Lepping, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*. Hoboken, NJ: Wiley, 2018.

[14] M. J. Zaki and C. J. Hsiao, "CHARM: An efficient algorithm for closed itemset mining," in *Proc. SDM*, 2002, pp. 457–473.

[15] M. J. Zaki and K. Gouda, "Fast vertical mining using diffsets," in *Proc. KDD*, 2003, pp. 326–335.

[16] B. Vo, T. Le, F. Coenen, and T. P. Hong, "Mining frequent itemsets using the N-list and subsume concepts," *Int. J. Mach. Learn. Cybern.*, vol. 7, pp. 253–265, 2016.

[17] B. Wang, X. X. Fang, R. R. Lv, and J. J. Ma, "Weighted frequent itemset mining algorithm based on WNegNodeset structure," *J. Comput. Appl.*, vol. 37, no. 7, 2020.