

MiSA-Miner: An Anchor-Driven Missing-Set Framework for Maximal and Closed Frequent Itemset Mining

Liyuan Wang and Jing Yang

College of Artificial Intelligence, Taiyuan University of Technology, Taiyuan, China
Corresponding author: Jing Yang; e-mail: yangjing01@tyut.edu.cn

Abstract. Frequent itemset mining often produces a very large and redundant output space, making maximal frequent itemsets (MFIs) and closed frequent itemsets (CFIs) useful compact representations. Although existing MFI/CFI algorithms restrict enumeration to target patterns, they still retain many intermediate states and can incur substantial maximality or closure-checking overhead. To reduce this cost, we introduce MiSA-Miner, an anchor-driven framework built on a missing-set representation, in which an itemset is characterized by the transactions that do not contain it. With this representation, support is computed as N minus the size of the union of missing-sets, giving a unified formulation of frequency testing and closure checking. For MFI mining, MiSA-Miner organizes depth-first search around anchors derived from minimal missing-sets and performs anchor-driven expansion. For CFI mining, it conducts exact depth-first enumeration in the closure space using missing-set-based closure computation. Experiments on dense benchmark data verify that MiSA-Miner matches baseline outputs exactly while delivering substantial speedups in the target regime. On chess.dat, MiSA-Miner achieves 2.0x to 36.3x speedups over FPMMax and GenMax for MFI mining, and it successfully completes instances where the baselines time out under a 3-hour limit. Taken together, these results suggest that missing-set modeling provides a unified and practically effective foundation for MFI/CFI mining.

Keywords: Maximal Frequent Itemsets; Closed Frequent Itemsets; Missing-Set Representation; Anchor-Driven Search

1 Introduction

Frequent itemset mining (FIM) remains a central data-mining task and supports association analysis, pattern discovery, and recommendation modeling [1-10]. When support thresholds are low or data distributions are favorable, the complete set of frequent itemsets can grow rapidly, so compact output representations are often preferred in practice. Two widely used choices are maximal frequent itemsets (MFIs), which retain

only non-extendable frequent patterns, and closed frequent itemsets (CFIs), which retain support-preserving representatives [11-20].

FIM efficiency is shaped by the search strategy and by how support-related information is represented and updated during enumeration. Classical paradigms include candidate-generation methods such as Apriori [6, 21, 22], pattern-growth methods such as FP-Growth/FPMax [7, 13, 23, 24], and vertical methods that represent an itemset by the transaction ID set (tidset) of supporting transactions [15, 25, 26]. Difference-based variants such as diffsets further show how a complementary perspective can change the cost profile of support propagation [27]. For MFI mining, strong baselines include FPMax, GenMax, MaxMiner, MAFIA, CarpenterMax, and Charm-MFI [11-13, 28, 34, 35]. For CFI mining, CHARM, LCM, CLOSET/CLOSET+, DCI_Closed, Carpenter, and Grafci+ are representative methods [15-18, 20, 29, 30]. Even so, direct MFI/CFI enumeration still often retains many intermediate states and repeatedly pays maximality or closure-checking costs.

We revisit MFI/CFI mining through a missing-set representation. Rather than describing an itemset by the transactions that contain it, the representation records the transactions that do not contain it. This perspective differs from tidsets, which operate on supporting transactions, and from diffsets, which propagate relative differences along a traversal path. A missing-set is instead an absolute complement with respect to the full transaction universe. Under this formulation, support computation becomes N minus the cardinality of a missing-set union, while frequency testing and closure-related checks use the same primitives: unions and subset tests on missing-sets.

Building on this view, we develop **MiSA-Miner**. For MFI mining, MiSA-Miner organizes depth-first search around an anchor set induced by inclusion-minimal single-item missing-sets and performs anchor-driven expansion under the same representation. For CFI mining, it conducts depth-first enumeration in the closure space with exact missing-set-based closure computation. Thus, the two tasks share the same support and closure backbone but use different search-space organizations.

The main contributions are as follows:

1. First, we introduce a **missing-set representation** that is complementary to tidsets and fundamentally different from diffset propagation, giving a unified formulation for support computation, frequency pruning, and closure-related tests.
2. Second, we design **MiSA-Miner**, a unified framework for MFI and CFI mining that couples anchor-driven search for MFIs with exact missing-set-based closure enumeration for CFIs.
3. Third, we derive exact **maximality and closure criteria** under the missing-set representation and evaluate the framework on both MFI and CFI mining tasks.

2 Missing-Set Formulation and MiSA-Miner

2.1 Notation and problem setup

Let the original transaction database be $D = \{T_1, \dots, T_N\}$ over an item universe $\mathcal{I}$, and let σ denote the *absolute* minimum support. MiSA-Miner first removes infrequent single items and then removes empty transactions after item filtering, yielding a reduced database $D' = \{T'_1, \dots, T'_{N'}\}$ and the surviving frequent single items $\mathcal{F} \subseteq \mathcal{I}$. Since this preprocessing does not change the supports of itemsets over $\mathcal{F}$, all subsequent definitions and equations are with respect to D' . For simplicity, we omit the subscript D' and write $\text{sup}(X)$ and $\text{missing}(X)$.

Let $U = \{1, \dots, N'\}$ be the transaction index set of D' . An itemset $X \subseteq \mathcal{F}$ is frequent if $\text{sup}(X) \geq \sigma$. An MFI is a frequent itemset with no proper frequent superset, and a CFI is a frequent itemset with no proper superset of identical support [11, 14]. We define

$$\text{cap} = N' - \sigma, \quad (1)$$

which is the maximum allowed missing-set size for a frequent pattern.

2.2 Missing-set representation

For any item $x \in \mathcal{F}$, define its missing-set as

$$\text{missing}(x) = \{t \in U \mid x \notin T'_t\}. \quad (2)$$

For any itemset $X \subseteq \mathcal{F}$, define

$$\text{missing}(X) = \{t \in U \mid X \not\subseteq T'_t\}. \quad (3)$$

A transaction misses X if and only if it misses at least one item in X , so

$$\text{missing}(X) = \bigcup_{x \in X} \text{missing}(x). \quad (4)$$

Consequently, the supporting transactions of X are exactly $U - \text{missing}(X)$, and therefore

$$\text{sup}(X) = N' - |\text{missing}(X)|. \quad (5)$$

Hence,

$$X \text{ is frequent} \Leftrightarrow |\text{missing}(X)| \leq \text{cap}. \quad (6)$$

Under this formulation, support computation reduces to union-cardinality computation, and frequency testing becomes a threshold test on missing-set size.

2.3 Anchor-driven MFI mining

For MFI mining, MiSA-Miner performs depth-first search in the missing-set space. Each search node is represented by (P, M, tail) , where P is the current prefix, $M = \text{missing}(P)$, and tail is the ordered list of admissible extension items.

Anchor construction. MiSA-Miner builds an anchor set $\mathcal{A} \subseteq \mathcal{F}$ from inclusion-minimal single-item missing-sets. Candidate items are sorted by increasing $|\text{missing}(x)|$, and an item x is selected as an anchor only if no previously selected anchor a satisfies

$$\text{missing}(a) \subseteq \text{missing}(x). \quad (7)$$

These anchors act as minimal representatives that partition the MFI search space.

Item ordering. For each frequent item i , define its dominating anchor index as

$$\text{dom}(i) = \min\{k \mid \text{missing}(a_k) \subseteq \text{missing}(i)\}. \quad (8)$$

Items are globally ordered by $(\text{dom}(i), |\text{missing}(i)|, i)$, and the DFS block rooted at anchor a_k only extends with items whose dominating index is at least k .

Expansion, pruning, and absorption. When expanding a node by an item x , MiSA-Miner computes

$$M' = M \cup \text{missing}(x). \quad (9)$$

If $|M'| > \text{cap}$, the branch is infrequent and is pruned by Eq. (6). If $\text{missing}(x) \subseteq M$, then adding x does not change support, and MiSA-Miner immediately absorbs x into P .

Leaf maximality test. When no further frequent extension is accepted, MiSA-Miner performs a leaf-level maximality test: P is output as an MFI only if no frequent single item $y \notin P$ satisfies

$$|M \cup \text{missing}(y)| \leq \text{cap}. \quad (10)$$

This avoids maintaining a global superset-elimination structure during enumeration.

2.4 Missing-set-based CFI enumeration

For CFI mining, MiSA-Miner performs depth-first enumeration in the closure space. The key observation is that a frequent item x belongs to the closure of a node with missing-set M if and only if adding x preserves M , namely

$$\text{missing}(x) \subseteq M. \quad (11)$$

Therefore, the closure operator can be written as

$$\mathcal{C}(M) = \{x \in \mathcal{F} \mid \text{missing}(x) \subseteq M\}. \quad (12)$$

Given a candidate extension item x_i , the algorithm forms

$$M' = M \cup \text{missing}(x_i), \quad (13)$$

prunes the branch if $|M'| > \text{cap}$, computes the closed child $C(M')$, and applies the standard Close-by-One (CbO) canonicity rule under a fixed item order to avoid duplicates [16, 19]. Hence, MFI and CFI mining share the same missing-set representation and the same basic set operations, while differing in search-space organization.

3 Correctness Highlights

We summarize the key correctness properties used by MiSA-Miner.

Lemma 1 (Support equivalence in missing-set space). For any itemset $X \subseteq \mathcal{F}$, Eq. (4) holds and $\text{sup}(X) = N' - |\text{missing}(X)|$ as in Eq. (5).

Proof. By definition, a transaction misses X if and only if it misses at least one item in X , which gives Eq. (4). The supporting transactions of X are exactly $U - \text{missing}(X)$, so

$$\text{sup}(X) = |U - \text{missing}(X)| = N' - |\text{missing}(X)|.$$

Theorem 1 (Closedness criterion under missing-sets). A frequent itemset X is closed if and only if there is no item $y \in \mathcal{F} - X$ such that

$$\text{missing}(y) \subseteq \text{missing}(X). \quad (14)$$

Proof. If such a y exists, then

$$\text{missing}(X \cup \{y\}) = \text{missing}(X) \cup \text{missing}(y) = \text{missing}(X),$$

and thus $\text{sup}(X \cup \{y\}) = \text{sup}(X)$ by Eq. (5), so X is not closed. Conversely, if X is not closed, then some strict superset $Z \supset X$ satisfies $\text{sup}(Z) = \text{sup}(X)$. By Eq. (5), this implies $\text{missing}(Z) = \text{missing}(X)$. For any $y \in Z - X$, we then have $\text{missing}(y) \subseteq \text{missing}(X)$.

Theorem 2 (Anchor coverage and MFI completeness). Every maximal frequent itemset contains at least one anchor. Therefore, anchor-driven DFS together with exact frequency pruning, absorption, and leaf maximality testing enumerates every MFI.

Proof. Fix any frequent item $i \in \mathcal{F}$. By anchor construction, there exists an anchor $a \in \mathcal{A}$ such that $\text{missing}(a) \subseteq \text{missing}(i)$. Now let M be a maximal frequent itemset. If M contained no anchor, then for any $x \in M$ there would exist an anchor a such that

$$\text{missing}(a) \subseteq \text{missing}(x) \subseteq \text{missing}(M).$$

Hence,

$$\text{missing}(M \cup \{a\}) = \text{missing}(M),$$

and thus $\text{sup}(M \cup \{a\}) = \text{sup}(M) \geq \sigma$ by Eq. (5), contradicting maximality. Therefore, every MFI belongs to at least one anchor block. Along the corresponding DFS path, pruning is exact by Eq. (6), absorption preserves support because it only adds items x with $\text{missing}(x) \subseteq M$, and the leaf maximality test enforces the definition of maximality. Therefore, every MFI is enumerated.

Theorem 3 (Correctness of CFI enumeration). The closure-space DFS using $\mathcal{C}(M)$ and the standard CbO canonicity rule outputs every closed frequent itemset exactly once.

Proof. By Eq. (6), the support test in missing-set space is exact. By Eq. (11), $\mathcal{C}(M)$ is the exact closure operator under the missing-set representation. Standard CbO canonicity ensures that each closed frequent itemset is generated along a unique canonical path and that non-canonical duplicates are rejected.

4 Complexity Analysis

We analyze the time complexity of MiSA-Miner in three stages: preprocessing, MFI mining, and CFI mining. Let the filtered transaction database be D' , with n transactions, frequent single-item set F , and $m = |F|$. Let $L = \sum_{T \in D'} |T|$ be the total number of item occurrences in the filtered database, and let σ be the minimum support. Then $\text{cap} = n - \sigma$ is the maximum missing-set size allowed for a frequent pattern. For an itemset X ,

$$\text{missing}(X) = \bigcup_{x \in X} \text{missing}(x), \quad \text{sup}(X) = n - |\text{missing}(X)|. \quad (15)$$

In the bitset implementation, if the machine word size is w , one missing-set requires

$$B = \text{ceil}(n/w) \quad (16)$$

machine words. Accordingly, a missing-set union, a subset test, and a bit-count-based support computation each cost $O(B)$ time rather than $O(1)$.

Let the number of anchors be a . In MFI search, let K_M be the number of visited DFS nodes, Q_M the number of candidate extension edges, and L_M the number of leaf nodes. In CFI search, let K_C be the number of generated closed frequent itemsets, E_C the number of candidate extension edges, and E_C^+ the number of extensions that pass the frequency test and require closure computation.

Preprocessing. Reading the transaction database and counting item occurrences costs $O(L)$ time. Removing infrequent single items and reconstructing the filtered database also scans the transaction items and costs $O(L)$. Constructing missing-sets for the m frequent items costs $O(mn)$. Anchor construction selects inclusion-minimal missing-sets; with bitset-based containment checks, the worst-case number of checks is $O(am)$, each costing $O(B)$, so anchor construction costs $O(amB)$. Therefore,

$$T_{pre} = O(L + mn + amB). \quad (17)$$

MFI mining. The MFI branch first computes the dominating anchor of each frequent item and sorts the items globally, costing $O(amB + m \log m)$. During DFS expansion, each candidate extension mainly performs a bitset union and a bit-count test, costing $O(B)$ per edge and $O(Q_M B)$ in total. At a leaf, the global maximality check may scan up to m frequent items, again with an $O(B)$ test per item, giving $O(L_M m B)$. Let $|Out_M|$ be the total output-writing cost for all MFIs. Then

$$T_{MFI} = O(amB + m \log m + Q_M B + L_M mB + |Out_M|). \quad (18)$$

In the worst case, the frequent-itemset search space can be exponential: $K_M = O(2^m)$, $Q_M \leq mK_M$, and $L_M = O(2^m)$. Hence the worst-case MFI mining time can be summarized as

$$T_{MFI}^{worst} = O(mB2^m + |Out_M|). \quad (19)$$

CFI mining. The CFI branch performs DFS in the closed-set space. For each candidate extension edge, computing the new missing-set and testing frequency costs $O(B)$, giving $O(E_C B)$ for extension tests. For each extension that passes the frequency test, closure computation may scan all m frequent items, with one subset test of cost $O(B)$ per item. Thus closure computation costs $O(E_C^+ mB)$ in total. Let $|Out_C|$ be the total output-writing cost for all CFIs. Then

$$T_{CFI} = O(E_C B + E_C^+ mB + |Out_C|). \quad (20)$$

In the worst case, the number of closed frequent itemsets may also be exponential, with $E_C \leq mK_C$ and $K_C = O(2^m)$. Therefore,

$$T_{CFI}^{worst} = O(m^2 B 2^m + |Out_C|). \quad (21)$$

Total and space complexity. If both branches are executed, the total time complexity is

$$T_{total} = T_{pre} + T_{MFI} + T_{CFI}, \quad (22)$$

that is,

$$T_{total} = O(L + mn + amB + Q_M B + L_M mB + E_C B + E_C^+ mB + |Out_M| + |Out_C|), \quad (23)$$

up to the lower-order sorting term $m \log m$. If only MFI mining is executed, the total complexity is $O(L + mn + amB + Q_M B + L_M mB + |Out_M|)$; if only CFI mining is executed, it is $O(L + mn + E_C B + E_C^+ mB + |Out_C|)$. The space complexity is dominated by the filtered database, the m missing bitsets, the recursion stack, and the output buffer, and can be written as

$$O(L + mB + h), \quad (24)$$

where $h \leq m$ is the maximum DFS recursion depth.

The same analysis helps explain the dataset-dependent behavior. Let the database density be $\rho = L/(nm)$ and let the average missing ratio be $\bar{q} = \frac{1}{m} \sum_{x \in F} \frac{|missing(x)|}{n}$. On dense datasets, frequent items typically appear in many transactions, so $\bar{q}$ is small, missing-set unions grow slowly, and anchor partitioning and absorption more easily reduce the search space. In very large sparse datasets, missing-sets are often close to the full transaction universe, and B grows significantly with n , increasing the cost of each union, subset test, and bit-count operation. Therefore, in its present form MiSA-

Miner is better suited to dense datasets, while large sparse datasets require further optimization of set representation and low-level implementation.

5 Experiments

5.1 Experimental Setup

We evaluate MiSA-Miner on the benchmark dataset chess.dat ($N = 3196$), which is a standard dense dataset for frequent itemset mining studies [31]. Because MiSA-Miner aims to unify maximal frequent itemset (MFI) mining and closed frequent itemset (CFI) mining under the missing-set representation, we report results for both tasks.

For MFI mining, we considered several representative maximal-itemset baselines, including FPMax, GenMax, CarpenterMax, and Charm-MFI [12, 13, 34, 35]. The final reported MFI comparison focuses on FPMax and GenMax. Under our chess.dat evaluation budget, CarpenterMax and Charm-MFI were too slow to give a meaningful direct comparison in the main tables and figures.

For CFI mining, the comparison includes CHARM, LCM (closed), DCI_Closed, and Carpenter [15, 16, 29, 30]. Baseline runs are executed using standard implementations in our experimental pipeline, including SPMF-based executions where applicable [32, 33, 36].

For tractability, the timeout limit is set to 3 hours for MFI runs and 40 minutes for CFI runs. A run that does not finish within the corresponding limit is marked as TLE. For all reported settings that completed successfully, MiSA-Miner was verified to produce outputs consistent with the corresponding completed reference runs. When only MiSA-Miner finished within the time limit, correctness was additionally checked using an LCM-based validator implemented in C; however, because the implementation-level performance gap is too large, this validator is not used as a runtime comparison baseline.

5.2 Main Results on chess.dat

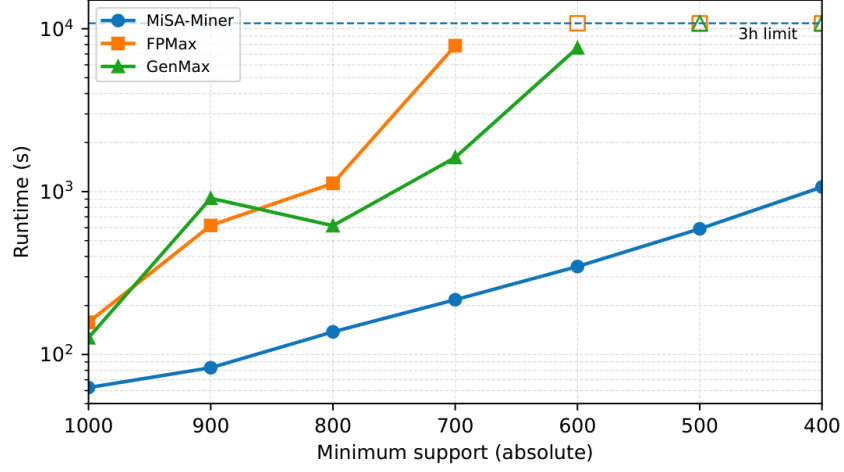


Fig. 1. Runtime comparison for MFI mining on chess.dat (N=3196). The y-axis uses log scale. Hollow markers indicate timeouts plotted at the 3-hour limit (10,800 s).

Table 1. MFI verification summary on chess.dat.

σ	Verified	#MFI
400	yes	1,543,815
500	yes	952,812
600	yes	607,661
700	yes	392,891
800	yes	258,547
900	yes	171,730
1000	yes	114,382

Note: Verified means exact output agreement with completed reference runs or with the C-based LCM validator when no compared baseline finished within the time limit.

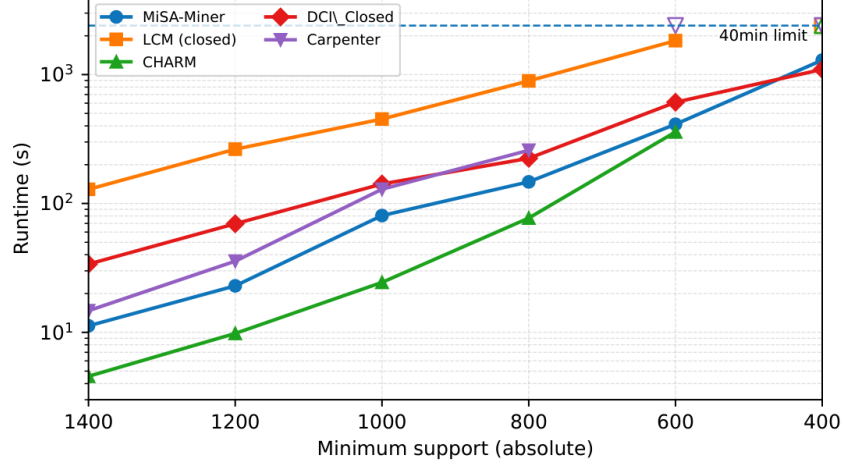


Fig. 2. Runtime comparison for CFI mining on chess.dat (N=3196). The y-axis uses log scale. Hollow markers indicate timeouts plotted at the 40-minute limit (2,400 s).

Table 2. CFI verification summary on chess.dat.

σ	Verified	#CFI
400	yes	77,890,865
600	yes	27,701,005
800	yes	10,800,367
1000	yes	4,445,373
1200	yes	1,892,002
1400	yes	826,842

Note: Verified means exact output agreement with completed reference runs or with the C-based LCM validator when no compared baseline finished within the time limit.

Figs. 1 and 2 give the runtime behavior of MiSA-Miner on the dense benchmark chess.dat. The verification tables list the exact output counts for the settings where consistency was confirmed.

MFI results. Fig. 1 and Table 1 show that MiSA-Miner consistently outperforms FPMaX and GenMaX across the tested MFI range. At $\sigma=1000$, MiSA-Miner finishes in 62.7 s, compared with 157.7 s for FPMaX and 126.7 s for GenMaX, corresponding to about 2.5x and 2.0x speedups, respectively. The largest observed gap appears at $\sigma=700$, where MiSA-Miner finishes in 217.0 s, versus 7867.8 s for FPMaX and 1620.9 s for GenMaX, i.e., about 36.3x and 7.5x faster. When the support threshold decreases further, MiSA-Miner still completes while the baselines hit the time limit: at $\sigma=500$ and $\sigma=400$, FPMaX and GenMaX both reach the 3-hour timeout, whereas MiSA-Miner still finishes in 591.2 s and 1068 s, respectively. This behavior suggests that the missing-set representation together with anchor-driven search is effective in controlling search cost on dense data with rapidly growing output spaces.

CFI results. For CFI mining, MiSA-Miner remains competitive on dense data and is strongest in the hardest low-support regime. Fig. 2 and Table 2 also show that MiSA-Miner is not uniformly the fastest in the medium-to-high support regime, where specialized baselines such as CHARM remain strong. As the threshold decreases and the number of CFIs grows rapidly, however, MiSA-Miner shows better completion robustness. At $\sigma=400$, where the number of CFIs reaches 77,890,865, MiSA-Miner still finishes within the 40-minute limit, while CHARM, LCM (closed), and Carpenter hit TLE. Taken together, the CFI results show that MiSA-Miner remains practically competitive on dense datasets and is more robust in the most challenging regime.

5.3 Supplemental Results on the connect Benchmark

We also report the supplemental results contained in the provided result archive and connect CFI summary. The dataset is connect, with 67,557 transactions, average transaction length 43.0, and 129 distinct items. The support thresholds are 50,000, 45,000, 40,000, 35,000, 30,000, 25,000, and 20,000. For MFI mining, the compared baselines are FPMMax and GenMax, with a 2-hour timeout. For CFI mining, the available summary reports MiSA-Miner-CFI, CHARM, and LCM closed runs, all under a 40-minute timeout.

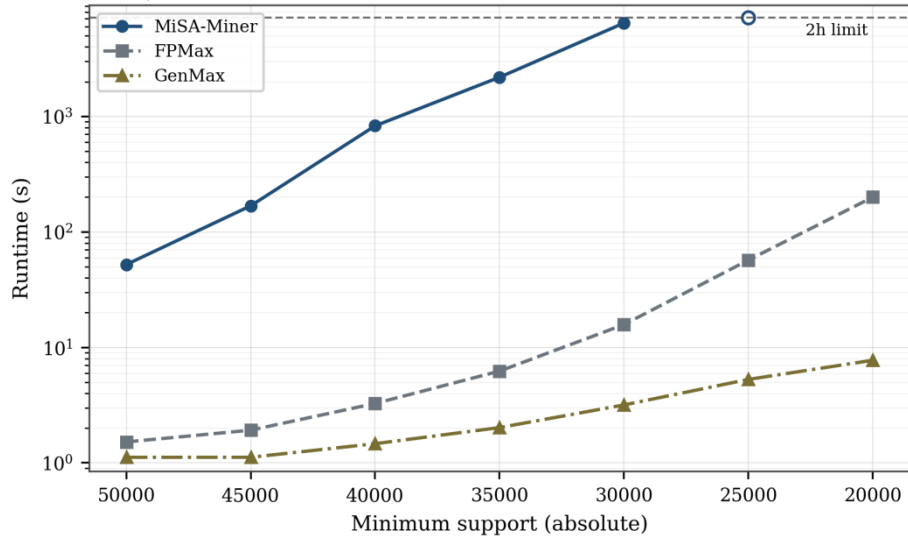


Fig. 3. Supplemental MFI runtime comparison on the connect benchmark. The y-axis uses log scale. The hollow MiSA-Miner marker at $\sigma=25,000$ indicates timeout at the 2-hour limit; The MiSA-Miner MFI run at $\sigma=20,000$ was not executed after the previous timeout.

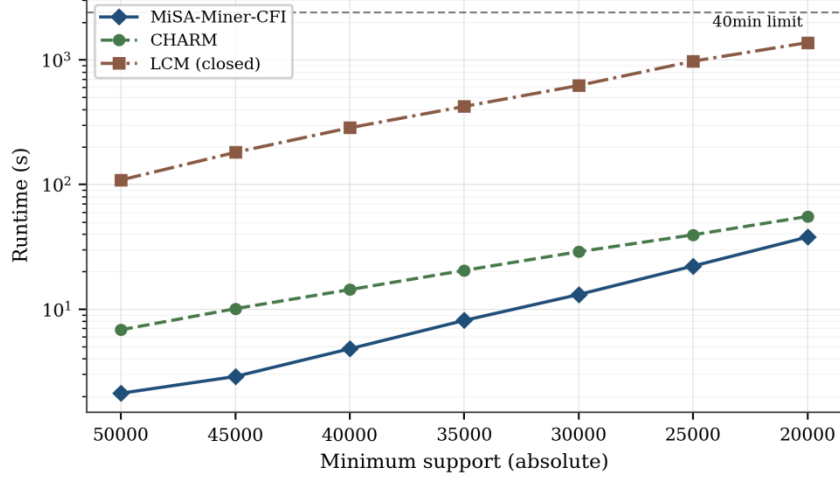


Fig. 4. Supplemental CFI runtime comparison on the connect benchmark. The y-axis uses log scale. All available MiSA-Miner-CFI, CHARM, and LCM closed runtimes are plotted.

Table 3. Supplemental results on the connect benchmark. MFI entries report count/time; CFI times are MiSA/CHARM/LCM closed wall-clock seconds.

σ	MiSA MFI #/s	FPMaX MFI #/s	GenMaX MFI #/s	#CFI	CFI time MiSA/CHARM/LCM (s)
50,000	985/52.15	985/1.52	985/1.12	26,417	2.12/6.83/108.67
45,000	1,466/168.26	1,466/1.92	1,466/1.12	44,990	2.88/10.09/181.00
40,000	2,195/828.72	2,195/3.27	2,195/1.47	71,990	4.81/14.35/285.19
35,000	3,358/2,183.59	3,358/6.23	3,358/2.02	116,165	8.12/20.47/422.73
30,000	4,931/6,432.40	4,931/15.76	4,931/3.17	181,949	13.10/28.89/622.50
25,000	TO/7,200	7,323/57.01	7,323/5.28	288,918	22.18/39.46/975.36
20,000	Not run/-	11,167/199.86	11,167/7.74	474,297	37.86/55.40/1374.25

On connect, the MFI branch of MiSA-Miner is substantially slower than FPMaX and GenMaX at all completed thresholds and times out at $\sigma=25,000$, while the two baselines continue to complete. GenMaX is the fastest baseline in all reported MFI settings. For CFI mining on connect, Table 3 and Fig. 4 report all available baseline data: the CFI count increases from 26,417 at $\sigma=50,000$ to 474,297 at $\sigma=20,000$; MiSA-Miner-CFI runtime increases from 2.12 s to 37.86 s, CHARM from 6.83 s to 55.40 s, and LCM closed from 108.67 s to 1374.25 s. All CFI runs completed successfully, and the supplied CFI summary gives successful count-based verification for all listed thresholds. MiSA-Miner-CFI is faster than CHARM by 1.46x to 3.50x and faster than LCM closed by 36.30x to 62.76x. These supplemental results therefore limit the implementation-level MFI claim on larger benchmark settings while strengthening the connect CFI comparison through full reporting of the available baseline runtimes.

5.4 Ablation Study

To assess each component, the ablation experiments are conducted on chess.dat under the same support settings as in the main results. For MFI mining, three variants are considered: removing anchor partition (together with dominating-anchor ordering), removing absorption, and removing the leaf-level maximality check. For CFI mining, the study removes root closure, closure-after-extension, and canonicity, respectively.

Table 4. Ablation results for MFI mining on chess.dat.

σ	Variant	Runtime (s)	Nodes	#Output
1000	Full	68.10	9,149,390	114,382
1000	w/o anchor	332.07	28,676,695	114,382
1000	w/o absorption	76.53	17,106,273	114,382
1000	w/o leaf check	63.26	9,149,390	7,086,796
700	Full	237.19	43,946,700	392,891
700	w/o anchor	2173.54	182,179,628	392,891
700	w/o absorption	393.22	106,467,713	392,891
700	w/o leaf check	316.65	43,946,700	33,028,985
500	Full	1128.74	145,983,860	952,812
500	w/o anchor	18869.87	799,966,325	952,812
500	w/o absorption	2884.75	460,340,593	952,812
500	w/o leaf check	1551.96	145,983,860	104,931,663

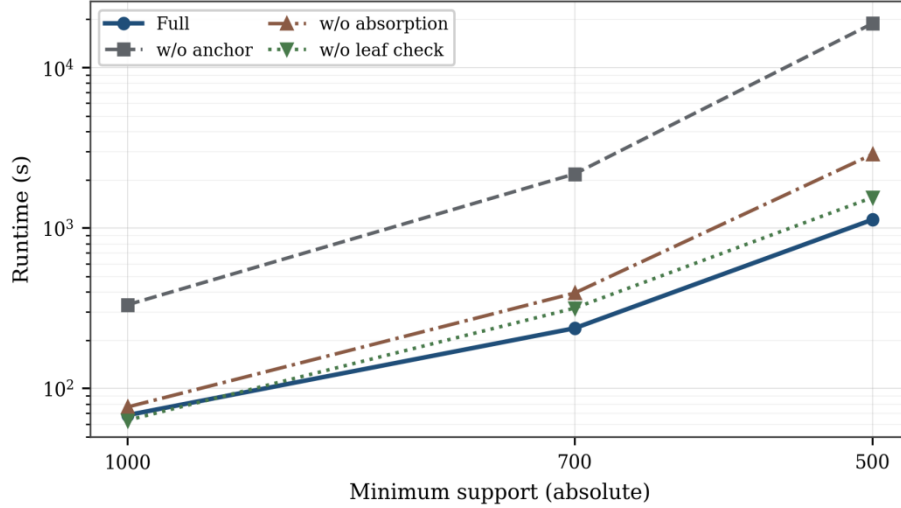


Fig. 5. Runtime trends of the MFI ablation on chess.dat. The y-axis is shown in log scale.

Table 5. Ablation results for CFI mining on chess.dat.

σ	Variant	Runtime (s)	Edges	#Output
1400	Full	12.10	4,191,115	826,842
1400	w/o root closure	16.97	4,191,115	826,842
1400	w/o ext. closure	22.80	13,946,551	3,457,746
1400	w/o canonicity	28.27	7,495,519	1,925,356
1000	Full	85.94	25,664,243	4,445,373
1000	w/o root closure	105.57	25,664,243	4,445,373
1000	w/o ext. closure	214.52	134,592,610	29,442,848
1000	w/o canonicity	215.36	56,052,805	12,900,274
800	Full	179.70	65,682,587	10,800,367
800	w/o root closure	274.16	65,682,587	10,800,367
800	w/o ext. closure	739.24	494,875,462	98,405,244
800	w/o canonicity	650.34	169,551,444	36,160,217

MFI ablation. Table 4 and Fig. 5 identify anchor partition as the dominant source of efficiency gain in the MFI branch. Removing anchors preserves the exact MFI counts but increases runtime from 68.10/237.19/1128.74 s to 332.07/2173.54/18869.87 s at $\sigma=1000/700/500$, corresponding to 4.88x/9.16x/16.72x slowdowns. The number of visited nodes also rises sharply, from 9.15M/43.95M/145.98M to 28.68M/182.18M/799.97M, which shows that anchor-driven partitioning is the main mechanism for suppressing redundant exploration.

Removing absorption likewise preserves the exact output counts, but causes moderate-to-large overhead, especially at lower support. Runtime increases to **76.53/393.22/2884.75 s**, which is **1.12x/1.66x/2.56x** slower than the full method. The result suggests that absorption provides a useful secondary pruning effect once the search space becomes larger.

In contrast, removing the leaf-level maximality check does not increase the number of explored nodes, but output size grows sharply, from **114,382/392,891/952,812** to **7,086,796/33,028,985/104,931,663**. Thus, the leaf check is not primarily a search-reduction device; instead, it is the mechanism that enforces maximality at output time.

CFI ablation. Table 5 indicates that the CFI branch relies mainly on closure-after-extension and canonicity. Removing root closure does not change the explored frontier or the final output count, but it increases runtime from **12.10 s** to **16.97 s** at $\sigma = 1400$, from **85.94 s** to **105.57 s** at $\sigma = 1000$, and from **179.70 s** to **274.16 s** at $\sigma = 800$, suggesting that root closure mainly improves initialization efficiency.

The largest degradation occurs when closure-after-extension is removed. At $\sigma = 1400/1000/800$, runtime increases to **22.80/214.52/739.24 s**, explored edges increase to **13.95M/134.59M/494.88M**, and output size expands to **3.46M/29.44M/98.41M**. This pattern confirms that re-closing after each extension is essential for keeping the search inside a compact closed-set space.

Removing canonicity introduces another substantial source of redundancy. At $\sigma = 1400/1000/800$, runtime becomes **28.27/215.36/650.34 s**, i.e., **2.34x/2.51x/3.62x** slower than

the full method, while the number of outputs grows from **826,842/4,445,373/10,800,367** to **1,925,356/12,900,274/36,160,217**. This confirms that canonicity is needed for duplicate-free closed-set enumeration.

5.5 Boundary Case on Sparse Data

To examine the prototype's current scalability boundary, MiSA-Miner is further tested on the large sparse dataset *kosarak.dat* ($N = 990,002$) at $\sigma = 4951$ (0.5%).

Table 6. Exploratory scalability check on *kosarak.dat* at $\sigma=4951$ (0.5%). Time limits: 3 hours for MFI and 40 minutes for CFI.

Task	MiSA-Miner (s)	Baseline	Baseline (s)	Baseline faster
MFI	1609	FPMa	2.39	673.6x
MFI	1609	GenMa	19.77	81.4x
CFI	2151	Carpenter	4.56	471.5x

Table 6 shows that MiSA-Miner is much less competitive on this large sparse dataset, even at a relatively high support threshold. At $\sigma = 4951$, MiSA-Miner requires **1609 s** for MFI mining and **2151 s** for CFI mining, whereas FPMa, GenMa, and Carpenter finish in only a few seconds. The result indicates that the current Python prototype is still mainly advantageous on dense datasets, while its missing-set-based union-cardinality operations become expensive when the transaction universe is very large and sparse.

Taken together, the experiments support a conservative conclusion: MiSA-Miner is a correct and practically effective framework on dense transaction datasets, with particularly strong behavior for MFI mining and good robustness for low-support CFI mining; however, further optimization is still needed to improve competitiveness on very large sparse data.

6 Conclusion

We presented MiSA-Miner, an anchor-driven framework that reinterprets MFI/CFI mining through missing-sets rather than supporting transaction sets. The complementary representation provides a unified basis for support evaluation, pruning, absorption, maximality checking, and closure computation. On dense benchmark data, MiSA-Miner demonstrates both theoretical correctness and practical effectiveness, and is particularly strong for MFI mining, where it delivers clear speedups over FPMa and GenMa while still completing settings that cause baseline timeouts. The experiments also make its current boundary on larger or sparse data explicit. Future work will therefore focus on more scalable set representations, stronger pruning strategies, and lower-level optimization for sparse or low-support scenarios.

References

- [1] Raj, S., Ramesh, D., Gantela, P.: Crossfim: a spark-based hybrid frequent itemset mining algorithm for large datasets. *Cluster Computing* 28, 231 (2025)
- [2] Han, J., Pei, J., Tong, H.: *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 4th edn. (2022)
- [3] Luna, J.M., Fournier-Viger, P., Ventura, S.: Frequent itemset mining: a 25 years review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9(6) (2019)
- [4] Fournier-Viger, P., Lin, J.C.W., Vo, B., Chi, T.T., Zhang, J., Le, H.B.: A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7(4) (2017)
- [5] Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*. pp. 207-216 (1993)
- [6] Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*. pp. 487-499 (1994)
- [7] Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. pp. 1-12 (2000)
- [8] Han, J., Cheng, H., Xin, D., Yan, X.: Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery* 15(1), 55-86 (2007)
- [9] Zhu, H., Lin, F., Hu, W.: Warm: an interpretability module with weighted association rule mining for recommendation systems. In: Huang, D.S., Zhang, X., Zhang, Q. (eds.) *Advanced Intelligent Computing Technology and Applications. Lecture Notes in Computer Science*, vol. 14875, pp. 75-87. Springer (2024)
- [10] Chen, Q., Fang, W.: High average-utility pattern mining based on genetic algorithm with a novel pruning strategy. In: Huang, D.S., Zhang, X., Chen, W. (eds.) *Advanced Intelligent Computing Technology and Applications - 20th International Conference, ICIC 2024, Tianjin, China, August 5-8, 2024, Proceedings, Part I. Lecture Notes in Computer Science*, vol. 14862, pp. 3-13. Springer (2024)
- [11] Bayardo, R.J.: Efficiently mining long patterns from databases. In: *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD)* (1998)
- [12] Gouda, K., Zaki, M.J.: GenMax: an efficient algorithm for mining maximal frequent itemsets. *Data Mining and Knowledge Discovery* 11(3), 223-242 (2005)
- [13] Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using FP-trees. *IEEE Transactions on Knowledge and Data Engineering* 17(10), 1347-1362 (2005)
- [14] Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: *Proceedings of the International Conference on Database Theory (ICDT)*. pp. 398-416 (1999)
- [15] Zaki, M.J., Hsiao, C.J.: Charm: an efficient algorithm for closed itemset mining. In: *Proceedings of the 2002 SIAM International Conference on Data Mining (SDM)*. pp. 457-473 (2002)
- [16] Uno, T., Asai, T., Uchida, Y., Arimura, H.: Lcm: an efficient algorithm for enumerating frequent closed item sets. In: *Proceedings of the Workshop on Frequent Itemset Mining Implementations (FIMI)* (2004)
- [17] Pei, J., Han, J., Mao, R.: CLOSET: an efficient algorithm for mining frequent closed itemsets. In: *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*. pp. 21-30 (2000)
- [18] Wang, J., Han, J., Pei, J.: CLOSET+: searching for the best strategies for mining frequent closed itemsets. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (2003)
- [19] Ganter, B., Wille, R.: *Formal Concept Analysis: Mathematical Foundations*. Springer (1999)
- [20] Ledmi, M., Zidat, S., Hamdi-Cherif, A.: Grafci+: a fast generator-based algorithm for mining frequent closed itemsets. *Knowledge and Information Systems* 63, 1873-1908 (2021)

- [21] Park, J.S., Chen, M.S., Yu, P.S.: An effective hash-based algorithm for mining association rules. In: Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD). pp. 175-186 (1995)
- [22] Savasere, A., Omiecinski, E., Navathe, S.: An efficient algorithm for mining association rules in large databases. In: Proceedings of the 21st International Conference on Very Large Data Bases (VLDB) (1995)
- [23] Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: H-Mine: hyper-structure mining of frequent patterns in large databases. In: Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM) (2001)
- [24] Borgelt, C.: An implementation of the FP-growth algorithm. In: Proceedings of the 1st International Workshop on Open Source Data Mining (OSDM) (2005)
- [25] Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New algorithms for fast discovery of association rules. In: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD) (1997)
- [26] Zaki, M.J.: Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering* 12(3), 372-390 (2000)
- [27] Zaki, M.J., Gouda, K.: Fast vertical mining using diffsets. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD) (2003)
- [28] Burdick, D., Calimlim, M., Gehrke, J.: MAFIA: a maximal frequent itemset algorithm for transactional databases. In: Proceedings of the 17th International Conference on Data Engineering (ICDE). pp. 443-452 (2001)
- [29] Lucchese, C., Orlando, S., Perego, R.: DCI_Closed: a fast and memory efficient algorithm to mine frequent closed itemsets. In: Proceedings of the Workshop on Frequent Itemset Mining Implementations (FIMI) (2004)
- [30] Pan, F., Dong, G., Han, J., Ramamohanarao, K.: CARPENTER: finding closed patterns in long biological datasets. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). pp. 637-642 (2003)
- [31] Goethals, B.: Frequent itemset mining implementations repository (FIMI). Repository of implementations and publicly available datasets from the FIMI'03 and FIMI'04 workshops (2003), available online
- [32] Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W., Tseng, V.S.: SPMF: a java open-source pattern mining library. *Journal of Machine Learning Research* 15, 3389-3393 (2014)
- [33] Fournier-Viger, P., Lin, J.C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z.H., Lam, H.T.: The SPMF open-source data mining library version 2. In: *Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, Part III. Lecture Notes in Computer Science, vol. 9853, pp. 36-40 (2016)
- [34] Philippe Fournier-Viger et al.: CarpenterMax: maximal frequent itemset mining (SPMF documentation). Available online (2024), SPMF algorithm documentation
- [35] Philippe Fournier-Viger et al.: Charm-MFI: maximal frequent itemset mining based on CHARM (SPMF documentation). Available online (2024), SPMF algorithm documentation
- [36] Philippe Fournier-Viger et al.: SPMF: an open-source data mining library. Available online (2024)