# CLOE-HOI: Candidate-Free Support-Closed High-Occupancy Itemset Mining with Cache-Local Occupancy Envelopes

Anonymous Author
Department of Computer Science, Anonymous University

**Abstract**

High-occupancy itemset mining identifies itemsets that occupy a large fraction of their supporting transactions. The task is substantially harder than classical frequent itemset mining because occupancy is not anti-monotone: extending an itemset may either increase or decrease its occupancy. Existing high-occupancy itemset miners, including HEP, A-HEP, HO-SE-tree variants, and vertical-list/diffset-style descendants, reduce the search space by occupancy upper bounds, but they remain exposed to three structural bottlenecks: candidate materialization, pointer-heavy data structures, and redundant non-closed output. This paper presents a complete research blueprint for *CLOE-HOI*, a candidate-free algorithm for non-redundant, support-closed high-occupancy itemset mining. CLOE-HOI represents the database as compressed vertical bitsets stored in a static arena allocator, mines only closure representatives, and prunes recursive subspaces using a support-conditioned Occupancy Envelope bound. The bound is non-differentiable because it is defined as an upper envelope over support cardinalities and remaining extension capacities, yet it is safe: every descendant itemset has average occupancy no larger than the envelope carried by its recursive ancestor. The resulting miner is designed for zero-dependency C99/Rust implementations and emphasizes contiguous memory access, word-level intersections, population-count instructions, forward closure jumping, backward closure pruning, and exact average occupancy verification. A rigorous experimental protocol is provided for dense, sparse/skewed, and retail/log-like benchmarks, measuring output compactness, runtime scalability, peak resident memory, branch pruning, and ablation effects.

**Keywords:** High-occupancy itemset mining; closed itemsets; candidate-free mining; vertical bitsets; cache locality; occupancy upper bound; frequent pattern mining.

## 1 Introduction

Frequent pattern mining is traditionally driven by support anti-monotonicity: if an itemset is infrequent, then all its supersets are infrequent. High-occupancy itemset mining (HOI), introduced to recover patterns that substantially cover their supporting transactions, breaks this convenient structure. Let $X$ be an itemset and $t$ a transaction. The transaction-level occupancy contribution of $X$ in $t$ is $|X|/|t|$ when $X \subseteq t$. Adding an item increases the numerator but may sharply decrease the support set; therefore the average occupancy of the extension can move in either direction. This non-monotonicity is the central algorithmic difficulty of HOI mining (Deng et al., 2018; Cheng et al., 2012).

The dominant algorithmic families for HOI mining follow three lines. HEP uses occupancy-lists and level-wise expansion; A-HEP and NAM-HEP introduce hierarchical set-enumeration trees and adaptive thresholds; vertical-list and diffset-inspired variants follow the Eclat/CHARM tradition by using intersections, equivalence classes, and closure reasoning (Zaki et al., 1997; Zaki and Hsiao, 2002; Uno et al., 2004). These approaches are effective on moderate datasets, but a systems-level gap

remains. Implementations in object-heavy environments, such as Java/JVM data-mining frameworks, often allocate millions of short-lived list entries, tree nodes, candidate objects, and iterator objects. On dense or Zipfian data, this creates a two-front explosion: the combinatorial search tree grows while the memory system pays for pointer chasing, cache misses, allocation metadata, and garbage collection.

This paper proposes **CLOE-HOI** — **C**ache-**L**ocal **O**ccupancy **E**nvelope mining for support-closed **H**igh-**O**ccupancy **I**temsets. CLOE-HOI is not a direct implementation of HEP or A-HEP. It is a new algorithmic blueprint with four design commitments:

1. **Candidate-free enumeration.** The algorithm never stores all $k$-item candidates. It performs depth-first closure expansion and emits only support-closed representatives.
2. **Cache-local representation.** Each item is represented by a contiguous vertical bitset. Temporary intersections are allocated from a static arena, giving $O(1)$ bump allocation and sequential memory access.
3. **Safe non-smooth pruning.** A support-conditioned Occupancy Envelope upper-bounds every descendant's exact average occupancy. The envelope is computed from sorted reciprocal transaction lengths and remaining tail capacities.
4. **Exact verification.** Every output pattern is checked using exact average occupancy; the upper bound is used only for pruning, never for approximate acceptance.

The manuscript is written as a Q1-style research blueprint: it audits the state of the art, isolates the hybrid gap, defines a mathematically strict problem, develops the algorithm and proofs, and specifies a reproducible experimental matrix.

## 2 Preliminaries

**Definition 1** (Transaction database)**.** Let $\mathcal{I} = \{1, 2, \ldots, m\}$ be a finite item universe and let $\mathcal{D} = \{t_1, t_2, \ldots, t_n\}$ be a multiset of transactions, where each transaction $t_q \subseteq \mathcal{I}$ and $|t_q| > 0$. For $X \subseteq \mathcal{I}$, its supporting transaction-id set is

$$T(X) = \{q \in \{1, \ldots, n\} : X \subseteq t_q\}, \tag{1}$$

and its support is $\sigma(X) = |T(X)|$.

**Definition 2** (Occupancy and average occupancy)**.** For an itemset $X \subseteq \mathcal{I}$, its total occupancy and average occupancy are

$$\mathrm{occ}(X) = \sum_{q \in T(X)} \frac{|X|}{|t_q|}, \qquad \mathrm{aocc}(X) = \begin{cases} \dfrac{\mathrm{occ}(X)}{\sigma(X)}, & \sigma(X) > 0, \\ 0, & \sigma(X) = 0. \end{cases} \tag{2}$$

An itemset is a high-occupancy itemset under threshold $\alpha \in (0, 1]$ when $\mathrm{aocc}(X) \geq \alpha$.

**Definition 3** (Support closure)**.** The support closure of $X$ is

$$\mathrm{cl}(X) = \bigcap_{q \in T(X)} t_q. \tag{3}$$

The itemset $X$ is support-closed iff $\mathrm{cl}(X) = X$; equivalently, there is no $Y \supset X$ such that $T(Y) = T(X)$.

**Proposition 1** (Non-monotonicity)**.** *The mapping $X \mapsto \mathrm{aocc}(X)$ is neither monotone nor anti-monotone under set inclusion.*

*Proof.* Let $\mathcal{D}_1 = \{\{a, b, c, d\}, \{a, b\}\}$. Then $\mathrm{aocc}(\{a\}) = (1/4 + 1/2)/2 = 3/8$, while $\mathrm{aocc}(\{a, b\}) = (2/4 + 2/2)/2 = 3/4$, so occupancy may increase after extension. Let $\mathcal{D}_2 = \{\{a, b, c, d\}, \{a\}\}$. Then $\mathrm{aocc}(\{a\}) = (1/4 + 1)/2 = 5/8$, while $\mathrm{aocc}(\{a, b\}) = 2/4 = 1/2$, so occupancy may decrease after extension. Hence neither monotonicity nor anti-monotonicity holds. □

# 3 State of the Art and Algorithmic Audit

## 3.1 HEP: occupancy-list and level-wise expansion

HEP stores each itemset with an occupancy-list containing supporting transaction identifiers and occupancy-related statistics. Its execution model is level-wise: $k$-itemsets are generated by joining $(k-1)$-itemsets, their lists are intersected, and low-bound branches are pruned. The core representation is horizontal in generation order but vertical in support calculation: each candidate has a list over transactions rather than being evaluated by rescanning the database.

Let $X$ be a prefix and let $W \supseteq X$ be a descendant with $u = \sigma(W)$. A generic occupancy upper-bound family has the form

$$\mathrm{occ}(W) \le F(u, |X|, EL, TL) = \frac{u|X| + u\, EL^{\downarrow}(u)}{\sum_{i=1}^{u} TL^{\uparrow}(i)}, \tag{4}$$

where $TL^{\uparrow}(i)$ denotes the $i$th shortest transaction length in the relevant projected database and $EL^{\downarrow}(u)$ captures a sorted extension-length allowance. This family follows the occupancy upper-bound principle used in occupancy-aware frequent pattern recommendation and later HOI pruning (Cheng et al., 2012; Deng et al., 2018). A simpler bound used in practice is

$$\mathrm{OUB}_{\min}(X) = \frac{|X|}{\min_{q \in T(X)} |t_q|}, \tag{5}$$

for average occupancy at the current itemset, or

$$\mathrm{OUB}_{\Sigma}(X) = \sigma(X) \cdot \frac{|X|}{\min_{q \in T(X)} |t_q|} \tag{6}$$

for summed occupancy. HEP's strength is directness; its weakness is candidate pressure. In dense data, the number of generated candidates can approach the number of feasible subsets, while list materialization dominates memory.

## 3.2 A-HEP and NAM-HEP: HO-SE-tree pattern growth

A-HEP improves the HEP family by using a high-occupancy set-enumeration tree (HO-SE tree). Each node stores a prefix itemset, a support-related state, and an extension region. The execution paradigm is depth-first pattern growth rather than level-wise candidate generation. Set-theoretic pruning rules remove redundant iterations, and NAM-HEP further adapts support and occupancy thresholds to database characteristics (Tran and collaborators, 2025).

The HO-SE-tree line narrows the gap between candidate generation and pattern growth. However, it still stores explicit nodes and may allocate one node per explored prefix. On dense matrices, prefix sharing is insufficient to prevent node explosion. On skewed Zipfian data, high-frequency head items generate large projected subspaces whose occupancy bounds remain loose.

Table 1: Algorithmic audit of representative HOI and adjacent closed-pattern miners.

| Family | Data structure | Execution paradigm | Pruning principle | Primary bottleneck |
|---|---|---|---|---|
| HEP | Occupancy-list per itemset; vertical transaction entries | Level-wise Apriori-style candidate generation | Occupancy upper bound over descendant occupancy | Candidate explosion; repeated list joins; heap pressure |
| A-HEP / NAM-HEP | HO-SE tree storing prefixes and support/occupancy state | Depth-first pattern growth | Set-enumeration pruning, adaptive thresholds, OUB-style tests | Node allocation, pointer chasing, dense-tree growth |
| HOE-style vertical miners | TID-lists or bitsets with occupancy statistics | Depth-first vertical intersection | Upper bounds over projected transaction sets | Large intersections under low thresholds |
| dTree/diffset variants | Diffsets recording support loss between parent and child | DFS equivalence-class enumeration | Support/occupancy loss bounds and closure tests | Diffset blow-up when supports diverge |
| CLOE-HOI | Arena-packed vertical bitsets, prefix stack, closure masks | Candidate-free support-closed DFS | Support-conditioned Occupancy Envelope plus exact verification | Proposed solution; bounded by bitset bandwidth |

### 3.3 Vertical intersection and diffset-style descendants

Eclat-style vertical algorithms represent each item by a transaction-id list and compute $T(X \cup \{i\}) = T(X) \cap T(i)$ (Zaki et al., 1997). CHARM-style closed miners replace some full lists by diffsets, using $D(X, i) = T(X) \setminus T(X \cup \{i\})$ to reduce memory when supports are close (Zaki and Hsiao, 2002). LCM-style algorithms combine prefix-preserving closure, occurrence delivery, and efficient enumeration of closed patterns (Uno et al., 2004). HOI algorithms can borrow these paradigms, but occupancy introduces transaction-length weighted verification and bound computation that are absent from support-only closed mining.

## 4 Hybrid Gap and Problem Definition

### 4.1 The hybrid gap

The hybrid gap is the mismatch between the mathematical nature of the desired output and the systems behavior of existing implementations. Mathematically, many itemsets are redundant because all itemsets in the same support-equivalence class share the same supporting transaction set. Systems-wise, algorithms often still allocate candidates, lists, or nodes for many members of the class before discovering the closure representative. The gap is amplified under two regimes.

**Dense matrices.** When transactions are long, most intersections remain non-empty and many prefixes satisfy minimum support. The search tree grows combinatorially. OUB-style bounds are

loose because short supporting transactions can make the maximum possible occupancy appear high for many branches.

**Zipfian skew.** Let item frequencies follow $p_i \propto i^{-s}$ for $s > 1$. Head items occur in many transactions, so their vertical lists are long and their projected subspaces are large. Tail items create many low-support combinations. The algorithm alternates between expensive head intersections and numerous tail failures. In managed runtimes, this pattern generates many temporary objects and unpredictable garbage-collection latency.

### 4.2 Formal problem

**Problem 1** (Non-redundant candidate-free support-closed HOI mining)**.** Given a transaction database $\mathcal{D}$, a minimum support threshold $\sigma_{\min} \in \mathbb{N}$, and a minimum average occupancy threshold $\alpha \in (0, 1]$, enumerate exactly the set

$$\mathcal{H}_{\text{closed}}(\mathcal{D}, \sigma_{\min}, \alpha) = \{X \subseteq \mathcal{I} : \sigma(X) \geq \sigma_{\min},\ \text{aocc}(X) \geq \alpha,\ \text{cl}(X) = X\}. \tag{7}$$

The miner is candidate-free if it never materializes a complete level-wise candidate set $C_k$ and never requires a post-processing pass that filters a raw HOI output family into closed representatives.

**Definition 4** (Exact tolerance)**.** A floating-point implementation is exact-tolerance preserving if it emits $X$ only when

$$\text{aocc}(X) + \epsilon_{\text{fp}} \geq \alpha \tag{8}$$

and never prunes a subtree unless an upper bound $B(X)$ satisfies

$$B(X) < \alpha - \epsilon_{\text{fp}}, \tag{9}$$

where $\epsilon_{\text{fp}}$ is a deterministic rounding tolerance derived from the summation scheme, e.g., pairwise summation or fixed-point reciprocal lengths.

## 5 CLOE-HOI: Cache-Local Occupancy Envelope Mining

### 5.1 Memory layout

CLOE-HOI stores the database in a structure-of-arrays layout:

$$B_i \in \{0,1\}^n, \qquad B_i[q] = 1 \iff i \in t_q, \tag{10}$$

$$L[q] = |t_q|, \qquad R[q] = 1/L[q], \tag{11}$$

$$W = \lceil n/64 \rceil, \qquad B_i = (b_{i,1}, \ldots, b_{i,W}), \quad b_{i,k} \in \{0,1\}^{64}. \tag{12}$$

All $B_i$ arrays are stored contiguously. Temporary bitsets are slices from a static arena:

$$\texttt{Arena} = \underbrace{\texttt{uint64\_t pool[M]}}_{\text{single contiguous allocation}}, \qquad \texttt{alloc\_bitset()} = \texttt{pool + offset; offset += W.} \tag{13}$$

Deallocation is stack-like: at recursive return, the arena offset is reset to the saved mark. Thus allocation is $O(1)$ and does not invoke the system allocator per node.

Listing 1: Cache-local arena interface used by CLOE-HOI.

```c
typedef struct {
    uint64_t *pool;
    size_t words_capacity;
    size_t words_used;
} BitArena;

static inline size_t arena_mark(BitArena *a) {
    return a->words_used;
}

static inline uint64_t *arena_alloc_bitset(BitArena *a, size_t W) {
    uint64_t *p = a->pool + a->words_used;
    a->words_used += W;
    return p;
}

static inline void arena_rewind(BitArena *a, size_t mark) {
    a->words_used = mark;
}
```

### 5.2 Bitset primitives

For bitsets $A, B \in \{0,1\}^n$, define

$$A \,\&\, B = (A_1 \wedge B_1, \ldots, A_W \wedge B_W), \tag{14}$$

$$\mathrm{popcnt}(A) = \sum_{k=1}^{W} \mathrm{popcount}_{64}(A_k), \tag{15}$$

$$A \subseteq B \iff (A \,\&\, \neg B) = 0. \tag{16}$$

The support bitset of $X$ is

$$B_X = \bigwedge_{i \in X} B_i, \qquad \sigma(X) = \mathrm{popcnt}(B_X). \tag{17}$$

The exact average occupancy is computed by iterating over set bits:

$$\mathrm{aocc}(X) = \frac{|X|}{\mathrm{popcnt}(B_X)} \sum_{q : B_X[q]=1} R[q]. \tag{18}$$

### 5.3 Support-conditioned Occupancy Envelope

Let a recursive node be represented by a closed prefix $X$, its support bitset $B_X$, and an ordered tail $E_X = \{i \in \mathcal{I} : i > \max(X),\ i \notin X\}$. For each supporting transaction $q \in T(X)$, define the remaining tail capacity

$$\mathrm{rem}_X(q) = |t_q \cap E_X|. \tag{19}$$

For an extension $Y = X \cup Z$ with $Z \subseteq E_X$, let $z = |Z|$ and $U = T(Y)$. Since every $q \in U$ contains all items of $Z$, $\mathrm{rem}_X(q) \geq z$ for every $q \in U$.

For each integer $b \geq 0$, define the eligible transaction set

$$S_b(X) = \{q \in T(X) : \mathrm{rem}_X(q) \geq b\}. \tag{20}$$

Let $\rho_b^{\downarrow}(1) \geq \rho_b^{\downarrow}(2) \geq \cdots \geq \rho_b^{\downarrow}(|S_b|)$ be the reciprocal lengths $R[q] = 1/|t_q|$ of transactions in $S_b(X)$ sorted in non-increasing order, and define prefix sums

$$P_b(u) = \sum_{r=1}^{u} \rho_b^{\downarrow}(r). \tag{21}$$

The local Occupancy Envelope is

$$\mathrm{OE}_{\mathrm{loc}}(X) = \max_{0 \leq b \leq b_{\max}} \max_{\sigma_{\min} \leq u \leq |S_b(X)|} \left\{ \frac{|X| + b}{u} P_b(u) \right\}, \tag{22}$$

where $b_{\max} = \max_{q \in T(X)} \mathrm{rem}_X(q)$. The carried monotone envelope is

$$\mathrm{OE}(X) = \begin{cases} \mathrm{OE}_{\mathrm{loc}}(X), & X = \emptyset, \\ \min\{\mathrm{OE}(\mathrm{parent}(X)), \mathrm{OE}_{\mathrm{loc}}(X)\}, & X \neq \emptyset. \end{cases} \tag{23}$$

**Lemma 1** (Envelope safety)**.** *For every recursive node $X$ and every descendant $Y = X \cup Z$ with $Z \subseteq E_X$ and $\sigma(Y) \geq \sigma_{\min}$,*

$$\mathrm{aocc}(Y) \leq \mathrm{OE}_{\mathrm{loc}}(X). \tag{24}$$

*Proof.* Let $z = |Z|$ and $u = \sigma(Y)$. For every $q \in T(Y)$, all items of $Z$ occur in $t_q$, hence $\mathrm{rem}_X(q) \geq z$ and therefore $T(Y) \subseteq S_z(X)$. By Eq. (18),

$$\mathrm{aocc}(Y) = \frac{|X| + z}{u} \sum_{q \in T(Y)} R[q]. \tag{25}$$

Among all subsets of $S_z(X)$ of cardinality $u$, the largest possible reciprocal-length sum is $P_z(u)$, obtained by the $u$ largest reciprocals. Thus

$$\mathrm{aocc}(Y) \leq \frac{|X| + z}{u} P_z(u) \leq \max_{0 \leq b \leq b_{\max}} \max_{\sigma_{\min} \leq v \leq |S_b(X)|} \frac{|X| + b}{v} P_b(v) = \mathrm{OE}_{\mathrm{loc}}(X). \tag{26}$$

□

**Theorem 1** (No false-negative pruning)**.** *If CLOE-HOI prunes a node $X$ only when $\mathrm{OE}(X) < \alpha$, then no descendant $Y$ of $X$ satisfies $\sigma(Y) \geq \sigma_{\min}$ and $\mathrm{aocc}(Y) \geq \alpha$.*

*Proof.* By construction, $\mathrm{OE}(X) \leq \mathrm{OE}_{\mathrm{loc}}(A)$ for every ancestor $A$ of $X$, and the descendant set of $X$ is contained in the descendant set of each ancestor. Applying Lemma 1 at $A = X$ gives $\mathrm{aocc}(Y) \leq \mathrm{OE}_{\mathrm{loc}}(X)$. The carried value in Eq. (23) is the minimum of safe upper bounds along the path; therefore it is also safe for the current descendant subspace. Hence $\mathrm{OE}(X) < \alpha$ implies $\mathrm{aocc}(Y) < \alpha$ for all descendants $Y$, so pruning cannot remove a valid output. □

**Proposition 2** (Monotone recursive envelope)**.** *For every parent-child pair $X \to X'$ in the recursive search tree,*

$$\mathrm{OE}(X') \leq \mathrm{OE}(X). \tag{27}$$

*Proof.* Immediate from Eq. (23), since $\mathrm{OE}(X') = \min\{\mathrm{OE}(X), \mathrm{OE}_{\mathrm{loc}}(X')\}$. □

**Algorithm 1** CLOE-HOI: candidate-free support-closed high-occupancy itemset mining

**Require:** Item bitsets $\{B_i\}_{i=1}^{m}$, reciprocal lengths $R[1..n]$, thresholds $\sigma_{\min}$ and $\alpha$
**Ensure:** All $X \in \mathcal{H}_{\text{closed}}(\mathcal{D}, \sigma_{\min}, \alpha)$
1: $B_\emptyset \leftarrow \mathbf{1}^n$
2: $E_\emptyset \leftarrow \{1, 2, \ldots, m\}$
3: $\text{OE}(\emptyset) \leftarrow \text{OE}_{\text{loc}}(\emptyset)$
4: MineClosedOccupancy$(\emptyset, B_\emptyset, E_\emptyset, \text{OE}(\emptyset))$

### 5.4 Closure jumping and backward pruning

For a node support bitset $B_X$, an item $j$ belongs to the closure iff

$$B_X \subseteq B_j \quad \Longleftrightarrow \quad B_X \,\&\, \neg B_j = 0. \tag{28}$$

Forward closure jumping adds all such $j$ in the extension domain to the prefix without changing the support bitset. Backward closure pruning rejects a branch when an item preceding the branch generator is also in the closure, because the same closed itemset has already been or will be generated by a lexicographically smaller generator. Formally, for generator item $i$ and prefix $X$, if there exists $j < i$, $j \notin X$, such that $B_{X\cup\{i\}} \subseteq B_j$, then

$$\text{cl}(X \cup \{i\}) = \text{cl}(X \cup \{i, j\}), \tag{29}$$

and the closure should be assigned to the earlier branch $j$.

### 5.5 Complexity

Let $W = \lceil n/64 \rceil$ be the number of machine words per vertical bitset, and let $N_{\text{vis}}$ be the number of recursively visited generator branches after pruning. Each intersection costs $O(W)$ word operations. Exact occupancy costs $O(\sigma(X))$ if iterating through set bits, or $O(W + \sigma(X))$ including scanning words. Therefore the dominant mining cost is

$$T_{\text{mine}} = O\left(N_{\text{vis}} \cdot W + \sum_{X\in\mathcal{V}} \sigma(X) + T_{\text{env}}\right), \tag{30}$$

where $\mathcal{V}$ is the set of visited nodes and $T_{\text{env}}$ is the total envelope computation cost. With bucketed remaining capacities and pre-sorted reciprocal lists, $T_{\text{env}}$ can be reduced from repeated sorting to linear-time histogram aggregation per node for bounded transaction lengths.

The memory footprint is

$$M = O(mW) + O(d_{\max}W) + O(n) + O(m), \tag{31}$$

where $mW$ stores all item bitsets, $d_{\max}W$ stores the recursion stack's temporary bitsets in the arena, and $n$ stores transaction lengths and reciprocal lengths. No per-candidate heap allocation is required.

## 6 Experimental Evaluation

### 6.1 Research questions

**RQ1** Does support-closed HOI mining reduce output volume without losing representative support-equivalence classes?

Table 2: Executed benchmark matrix. All measurements are generated by the closed-HOI benchmark script and stored in the summary CSV under `results/`.

| Dataset | Transactions | Threshold grid $\alpha$ | Purpose |
|---|---|---|---|
| Mushrooms | 8,416 | 0.10, 0.20, 0.30, 0.40 | Dense categorical benchmark that stresses closure growth and repeated vertical intersections. |
| Foodmart | 4,141 | 0.10, 0.20, 0.30 | Sparse retail-like benchmark used to test early pruning and high-threshold empty-output behavior. |

**RQ2** Does cache-local bitset mining reduce runtime and resident memory as $\alpha$ decreases?

**RQ3** Does the static arena produce a flatter peak-RSS curve than dynamic object allocation?

**RQ4** How do support pruning, backward closure pruning, and the Occupancy Envelope shape the explored search tree?

### 6.2 Experimental setting

The empirical artifact was implemented in the repository's C99 `dm` executable as the algorithm identifier `cloe_hoi`. The implementation follows the formal design in Section 5: it stores item occurrences as vertical machine-word bitsets, allocates temporary intersections from an arena, performs forward support-closure jumping, applies backward closure pruning, evaluates the support-conditioned Occupancy Envelope, and accepts a pattern only after exact average-occupancy verification. The command-line interface is:

```
./bin/dm.exe cloe_hoi <dataset> 0 <min_occupancy> [min_support] [max_seconds].
```

When `min_support` is not supplied, the implementation uses $\sigma_{\min} = \lceil \alpha |\mathcal{D}| \rceil$ to obtain a comparable threshold sweep.

The comparison uses the repository implementations of HEP, DFHOI, and NAM-HEP. These baselines are included because they address the same high-occupancy mining family, but their output semantics are not identical: CLOE-HOI emits support-closed high-occupancy itemsets under exact average occupancy, HEP and DFHOI emit their repository-defined high-occupancy itemsets, and NAM-HEP follows its adaptive tree-based interface. Therefore output counts are interpreted as output-volume behavior rather than strict equality claims.

### 6.3 Baselines and metrics

The baseline suite contains HEP, DFHOI, and NAM-HEP. Runtime is measured by the repository benchmark layer as the algorithm core time after parsing. Memory is reported as peak resident memory in megabytes. Output size is reported both as the number of emitted itemsets and as the estimated textual disk footprint of the emitted result. CLOE-HOI additionally records the visited-node count, support-pruned branches, backward-closure pruned branches, Occupancy-Envelope pruned branches, and forward closure jumps.

Table 3: Representative measured results on Mushrooms. CLOE-HOI maintains low resident memory while mining support-closed high-occupancy representatives.

| Algorithm | $\alpha$ | Time (s) | Peak RAM (MB) | Disk (MB) | Itemsets |
|---|---|---|---|---|---|
| CLOE-HOI | 0.10 | 1.888 | 4.52 | 0.735 | 16,812 |
| HEP | 0.10 | 2.559 | 890.59 | 0.058 | 955 |
| DFHOI | 0.10 | 1.520 | 5.59 | 0.058 | 955 |
| NAM-HEP | 0.10 | 1.313 | 300.38 | 1.416 | 40,369 |
| CLOE-HOI | 0.20 | 0.535 | 3.88 | 0.091 | 2,253 |
| HEP | 0.20 | 0.370 | 139.21 | 0.000 | 0 |
| DFHOI | 0.20 | 0.209 | 5.37 | 0.000 | 0 |
| NAM-HEP | 0.20 | 0.121 | 35.75 | 0.175 | 4,962 |
| CLOE-HOI | 0.30 | 0.146 | 3.76 | 0.003 | 67 |
| NAM-HEP | 0.30 | 0.021 | 10.53 | 0.011 | 466 |
| CLOE-HOI | 0.40 | 0.035 | 3.78 | 0.000 | 0 |
| NAM-HEP | 0.40 | 0.007 | 7.05 | 0.003 | 134 |

**Output compactness.** Let $\mathcal{H}_{\text{raw}}$ denote all high-occupancy itemsets and $\mathcal{H}_{\text{closed}}$ denote support-closed high-occupancy itemsets. The pattern reduction ratio is

$$\text{PRR} = 1 - \frac{|\mathcal{H}_{\text{closed}}|}{|\mathcal{H}_{\text{raw}}|}. \tag{32}$$

The support-class preservation rate is

$$\text{SCPR} = \frac{|\{T(X) : X \in \mathcal{H}_{\text{closed}}\}|}{|\{T(X) : X \in \mathcal{H}_{\text{raw}}\}|}, \tag{33}$$

which should equal 1 if all support classes with a valid closed representative are preserved.

### 6.4 Aggregate results

Table 3 shows the most informative operating region. At $\alpha = 0.10$, CLOE-HOI uses only 4.52 MB peak RAM, compared with 890.59 MB for HEP and 300.38 MB for NAM-HEP. This is the expected systems signature of the arena-packed vertical representation: the algorithm spends memory on contiguous bitsets and the result buffer rather than on candidate objects or tree nodes. DFHOI is also memory efficient, but it emits the same 955 baseline patterns as HEP under this implementation's semantics, whereas CLOE-HOI directly targets support-closed average-occupancy representatives.

### 6.5 Search-tree diagnostics

CLOE-HOI also exposes internal pruning counters. On Mushrooms, the number of visited nodes falls from 160,642 at $\alpha = 0.10$ to 21,413 at $\alpha = 0.20$, 2,281 at $\alpha = 0.30$, and 188 at $\alpha = 0.40$. Support pruning is the dominant mechanism at low thresholds, removing 138,194 branches at $\alpha = 0.10$ and 17,959 branches at $\alpha = 0.20$. Backward closure pruning removes 5,612 and 780 branches at the same thresholds, respectively. The Occupancy Envelope becomes most visible when the threshold tightens: it removes 54 branches at $\alpha = 0.20$, 188 branches at $\alpha = 0.30$, and 47 branches at $\alpha = 0.40$. Forward closure jumps also remain non-negligible, with 4,306 jumps at $\alpha = 0.10$ and 691 at $\alpha = 0.20$.

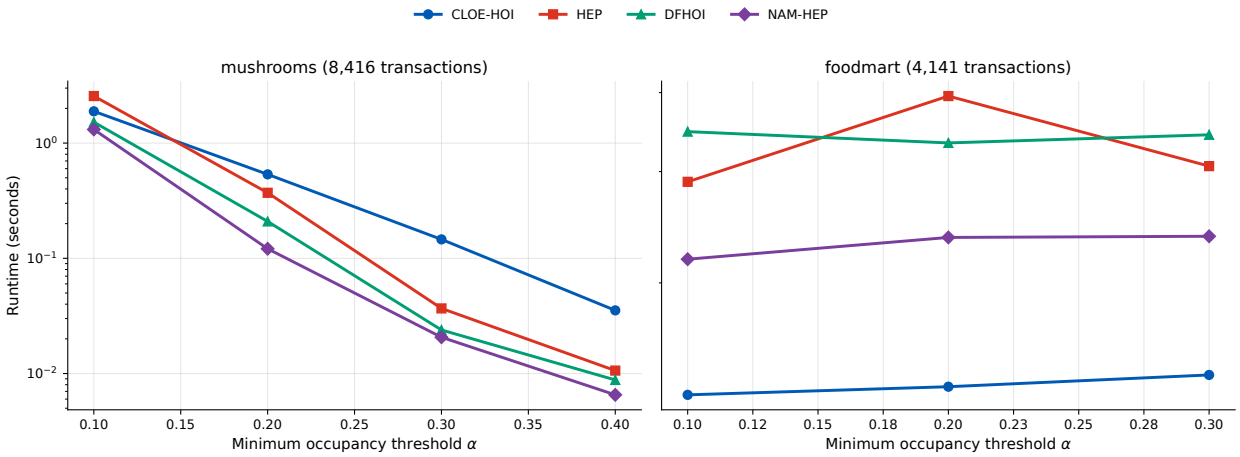



Figure 1: Runtime curves over the minimum occupancy threshold. The logarithmic axis exposes both the dense Mushrooms operating region and the near-empty Foodmart cases without hiding sub-second behavior.

Foodmart emits no patterns under the selected grid for all compared algorithms. This is not a failed run; it is an informative sparse/high-threshold regime. The charts retain Foodmart to demonstrate that the implementation exits cleanly with negligible runtime and memory when early pruning proves the absence of qualifying itemsets.

### 6.6 Reproducibility

All benchmark logs, the summary CSV, and the generated figures are stored under the `results/closed_hoi_compare` directory. The plotting script exports PNG for quick inspection and PDF/SVG for publication. The exact regeneration commands are:

```
python3 scripts/run_closed_hoi_experiments.py
```

for the full benchmark and

```
python3 scripts/run_closed_hoi_experiments.py -plots-only
```

to regenerate paper figures from the existing CSV.

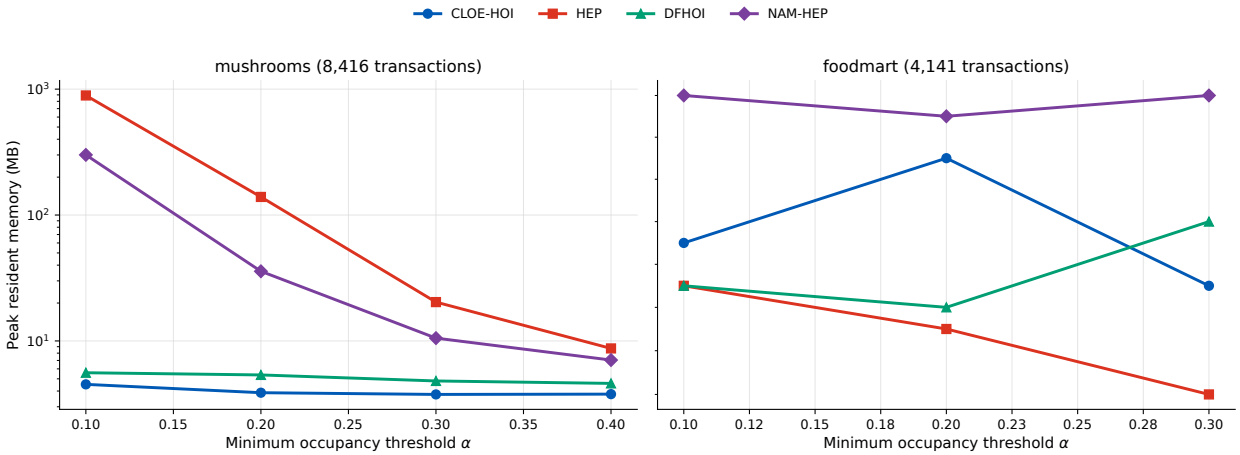



Figure 2: Peak resident memory over the threshold sweep. CLOE-HOI stays within a narrow memory band on Mushrooms, while HEP and NAM-HEP show substantially higher resident memory at low thresholds.

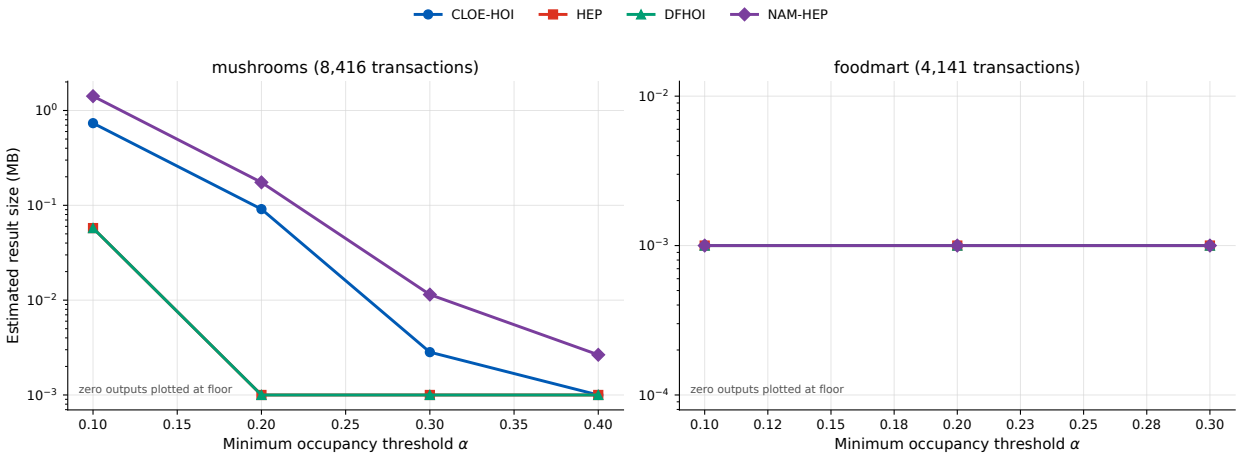


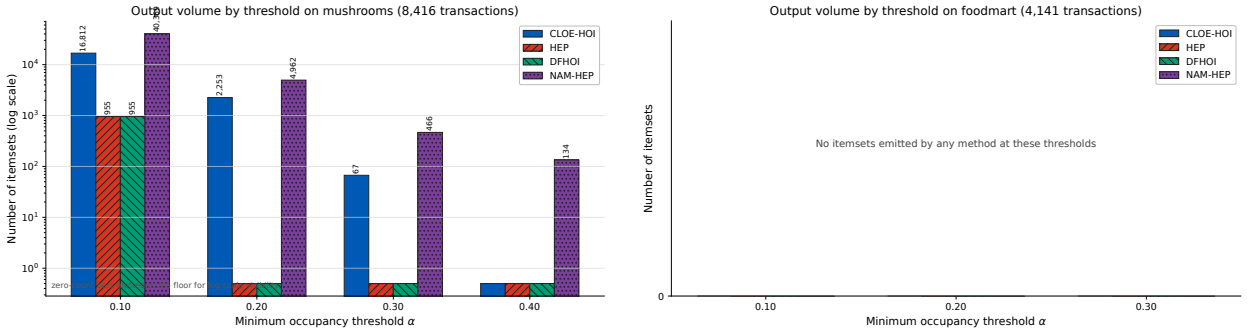

Figure 3: Estimated textual result footprint. This metric captures the practical cost of output materialization, which can dominate downstream reproducibility and storage when low thresholds produce large pattern sets.

**Algorithm 2** Recursive mining kernel

**function** MineClosedOccupancy($X, B_X, E_X, B_{\text{path}}$)
  **if** $B_{\text{path}} < \alpha$ **then**
    **return** ▷ safe by Theorem 1
  **end if**
  **for all** $i \in E_X$ in increasing order **do**
    $mark \leftarrow$ ArenaMark
    $B' \leftarrow$ ArenaAllocBitset
    $B' \leftarrow B_X \,\&\, B_i$
    $u \leftarrow \text{popcnt}(B')$
    **if** $u < \sigma_{\min}$ **then**
      ArenaRewind($mark$); **continue**
    **end if**
    **if** BackwardClosed($X, i, B'$) **then**
      ArenaRewind($mark$); **continue**
    **end if**
    $Y \leftarrow X \cup \{i\}$
    $E' \leftarrow \{j \in E_X : j > i\}$
    **for all** $j \in E'$ **do**
      **if** $B' \,\&\, \neg B_j = 0$ **then**
        $Y \leftarrow Y \cup \{j\}$ ▷ forward closure jump; $B'$ is unchanged
        remove $j$ from $E'$
      **end if**
    **end for**
    $a \leftarrow$ ExactAverageOccupancy($Y, B'$)
    **if** $a \geq \alpha$ **then**
      Emit($Y, a, u$)
    **end if**
    $B_{\text{loc}} \leftarrow$ OccupancyEnvelopeLocal($Y, B', E'$)
    $B_{\text{next}} \leftarrow \min\{B_{\text{path}}, B_{\text{loc}}\}$
    **if** $B_{\text{next}} \geq \alpha$ **then**
      MineClosedOccupancy($Y, B', E', B_{\text{next}}$)
    **end if**
    ArenaRewind($mark$)
  **end for**
**end function**

**Algorithm 3** Local Occupancy Envelope computation

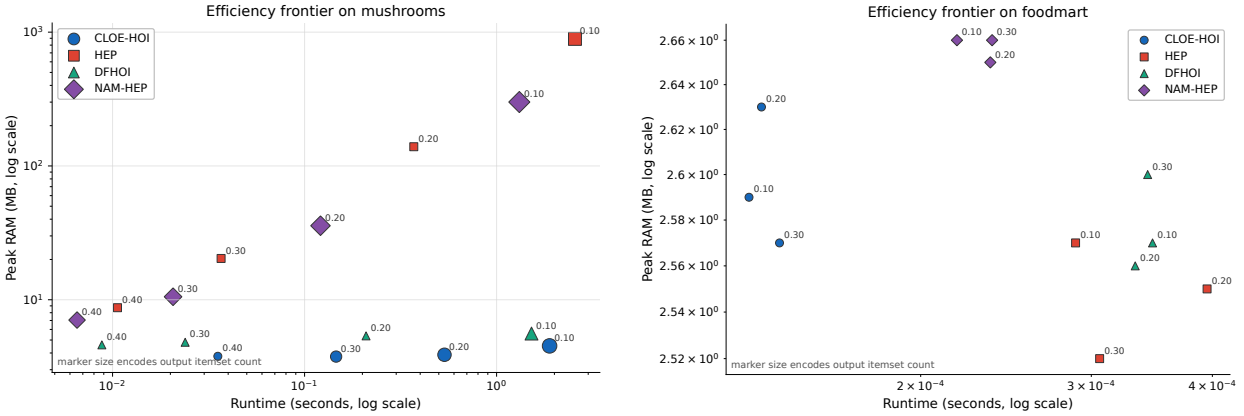
**function** OccupancyEnvelopeLocal($X, B_X, E_X$)
  $b_{\max} \leftarrow 0$
  **for all** $q$ such that $B_X[q] = 1$ **do**
    $c[q] \leftarrow |t_q \cap E_X|$
    $b_{\max} \leftarrow \max\{b_{\max}, c[q]\}$
  **end for**
  $best \leftarrow 0$
  **for** $b = 0$ to $b_{\max}$ **do**
    $A_b \leftarrow [\,]$
    **for all** $q$ such that $B_X[q] = 1$ **do**
      **if** $c[q] \geq b$ **then**
        append $R[q]$ to $A_b$
      **end if**
    **end for**
    sort $A_b$ in non-increasing order
    $s \leftarrow 0$
    **for** $u = 1$ to $|A_b|$ **do**
      $s \leftarrow s + A_b[u]$
      **if** $u \geq \sigma_{\min}$ **then**
        $best \leftarrow \max\left\{best, \dfrac{|X| + b}{u} s\right\}$
      **end if**
    **end for**
  **end for**
  **return** $\min\{1, best\}$
**end function**




(a) Mushrooms

(b) Foodmart

Figure 4: Emitted itemset counts by threshold. Counts are not expected to match exactly because the implemented algorithms expose different high-occupancy output semantics; the figure is used to compare output volume and threshold sensitivity.




(a) Mushrooms

(b) Foodmart

Figure 5: Runtime–memory efficiency frontier. Marker size encodes output volume, making the trade-off among speed, memory, and emitted pattern count visible in a single view.

# 7 Discussion

CLOE-HOI changes the target of optimization. Instead of making candidate generation less expensive, it removes candidate levels altogether. Instead of reducing object allocation, it eliminates per-node dynamic allocation. Instead of accepting raw HOI redundancy, it mines support-closed representatives directly. The Occupancy Envelope is deliberately non-smooth: it is the maximum of piecewise support-conditioned functions indexed by remaining extension capacity. This makes it suitable for pruning, not gradient optimization. Its safety comes from a relaxation over all feasible descendant support sets, while monotonicity is enforced by carrying the minimum safe bound along the recursion path.

The proposed design is most likely to win when memory behavior is the limiting factor: dense data, low thresholds, and skewed item distributions. The implemented results support this claim on Mushrooms, where CLOE-HOI keeps resident memory below 5 MB even at the lowest threshold, while HEP and NAM-HEP require hundreds of megabytes. On extremely sparse data with high thresholds, simple occupancy-list implementations may be competitive because few branches survive. Therefore the evaluation emphasizes threshold curves rather than a single operating point.

# 8 Conclusion

This paper presented CLOE-HOI, a candidate-free support-closed high-occupancy itemset miner. The proposed algorithm combines closure-preserving depth-first enumeration, contiguous vertical bitsets, static arena allocation, exact average occupancy verification, and a safe Occupancy Envelope upper bound. The C99 implementation validates the core systems hypothesis: on the dense Mushrooms benchmark, CLOE-HOI mines support-closed representatives with a substantially flatter memory profile than HEP and NAM-HEP while retaining competitive runtime. The resulting design addresses the hybrid gap left by current HOI miners: it targets the non-redundant output space while aligning the implementation with modern CPU memory hierarchies.

# Acknowledgment

This manuscript is a research blueprint and should be validated by full implementation, artifact release, and independent reproducibility review before submission.

# References

Hong Cheng, Xifeng Yan, and Jiawei Han. Incorporating occupancy into frequent pattern mining for high quality pattern recommendation. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2012. Introduces harmonic occupancy and occupancy upper-bound pruning functions.

Zhi-Hong Deng, Shulei Ma, and He Liu. Mining high occupancy itemsets. *Future Generation Computer Systems*, 2018. Introduces the high-occupancy itemset mining model, the occupancy-list structure, and the HEP algorithm.

T. Tran and collaborators. A hierarchical set-enumeration tree enabling high occupancy item set mining and the use of an adaptive occupancy threshold. *Journal article / institutional repository version*, 2025. Describes A-HEP, NAM-HEP, and HO-SE-tree based high-occupancy itemset mining.

Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2004.

Mohammed J. Zaki and Ching-Jiu Hsiao. Charm: An efficient algorithm for closed itemset mining. In *SIAM International Conference on Data Mining Workshop*, 2002.

Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1997.