Linear Suffix Projection and Reciprocal Pruning for High-Performance High-Occupancy Itemset Mining

Quan Van
Independent Researcher

Abstract

High-Occupancy Itemset (HOI) mining is a critical pattern mining paradigm that measures the average or total occupancy contribution of an itemset within transactions, preventing the discovery of long, sparse itemsets with low actual density. However, existing HOI mining algorithms suffer from massive memory usage due to pointer-heavy prefix trees or heavy bitmap intersection overheads.

In this paper, we present SPARC-HOI, a candidate-free high-occupancy itemset miner designed for a hardware-efficient execution model: flat primitive arrays, tight monotone upper bounds, static arena allocation, and cache-linear projection. The algorithm supports both average and summed occupancy semantics and completely avoids bitmap intersections. We prove the anti-monotonicity of the Reciprocal Residual Occupancy Boundary (RROB) and the Summed Occupancy Boundary over suffix projections. Experimental results show that SPARC-HOI achieves up to a 4× speedup and a 680× reduction in Peak RAM usage compared to state-of-the-art baselines.

1 Introduction

Frequent itemset mining (FIM) has been widely applied in transaction analysis, web mining, and bioinformatics. However, traditional support-based mining often discovers many long but sparse itemsets that occupy only a negligible fraction of their transactions. To address this, High-Occupancy Itemset (HOI) mining was introduced, requiring that the average proportion of transaction lengths covered by an itemset exceeds a threshold.

Existing HOI miners, such as HEP and DFHOI, suffer from severe performance bottlenecks. They either construct massive transaction lists in heap memory or repeatedly intersect large transaction bitmaps, incurring high CPU cache miss rates.

To overcome these challenges, we propose SPARC-HOI, which features:

- **Linear Suffix Projection**: Streams transaction structures contiguously inside a static arena, eliminating tree pointers and bitmap overhead.
- **Reciprocal Pruning Bounds**: Introduces tight, mathematically proven monotone bounds for both average and summed occupancy semantics.
- **Arena Residency**: Uses $O(1)$ allocation and rollback mechanisms to maintain constant and minimal memory usage.

2 Problem Formulation and Occupancy Semantics

Let $\mathcal{D} = \{T_1, T_2, \ldots, T_n\}$ be a transaction database over item universe $\mathcal{I}$. For an itemset $X \subseteq \mathcal{I}$, define its support transaction set as $\Gamma(X) = \{t \mid X \subseteq T_t\}$, and its support count as $\sup(X) = |\Gamma(X)|$.

Definition 1 (Average Occupancy)**.** *The average occupancy of an itemset X is defined as:*

$$\text{aocc}(X) = \frac{1}{\sup(X)} \sum_{t \in \Gamma(X)} \frac{|X|}{|T_t|} = \frac{|X|}{\sup(X)} \sum_{t \in \Gamma(X)} \frac{1}{|T_t|}.$$

Definition 2 (Summed Occupancy)**.** *The summed occupancy of an itemset X is defined as:*

$$\text{socc}(X) = \sum_{t \in \Gamma(X)} \frac{|X|}{|T_t|} = |X| \sum_{t \in \Gamma(X)} \frac{1}{|T_t|}.$$

Given support threshold σ and occupancy threshold α, the raw HOI mining task is to find all itemsets X such that:

$$\begin{aligned} &\sup(X) \geq \sigma \quad \text{and} \quad \text{aocc}(X) \geq \alpha \quad \text{(Average Mode), or} \\ &\sup(X) \geq \sigma \quad \text{and} \quad \text{socc}(X) \geq \alpha \quad \text{(Summed Mode).} \end{aligned}$$

3 Mathematical Foundations and Pruning Boundaries

Because occupancy is non-monotone along recursive extensions, tight monotone upper bounds are required to prune candidate paths.

3.1 Reciprocal Residual Occupancy Boundary

For a prefix path P, let its projected list be $\mathcal{L}(P)$, where each entry stores the transaction id t, matched prefix position, and the number of admissible extension items $r_P(t)$. Define the per-transaction residual occupancy contribution as:

$$b_P(t) = \frac{|P| + r_P(t)}{|T_t|}.$$

Let $b_P^{\downarrow}(1) \geq b_P^{\downarrow}(2) \geq \cdots \geq b_P^{\downarrow}(|\mathcal{L}(P)|)$ be these values sorted in non-increasing order.

The Reciprocal Residual Occupancy Boundary (RROB) for average occupancy is defined as:

$$\text{RROB}(P) = \max_{\sigma \leq u \leq |\mathcal{L}(P)|} \frac{1}{u} \sum_{i=1}^{u} b_P^{\downarrow}(i).$$

If $|\mathcal{L}(P)| < \sigma$, we define $\text{RROB}(P) = 0$.

Theorem 1 (Correctness of RROB)**.** *For every descendant Y of P with* $\sup(Y) \geq \sigma$*, we have* $\text{aocc}(Y) \leq \text{RROB}(P)$.

Proof. Let $u = \sup(Y)$ and $S = \Gamma(Y) \subseteq \Gamma(P)$. Since $|Y| \leq |P| + r_P(t)$ for any $t \in S$,

$$\text{aocc}(Y) = \frac{1}{u} \sum_{t \in S} \frac{|Y|}{|T_t|} \leq \frac{1}{u} \sum_{t \in S} b_P(t).$$

The sum of any u elements in $\{b_P(t)\}$ is bounded by the sum of the u largest elements. Hence,

$$\text{aocc}(Y) \leq \frac{1}{u} \sum_{i=1}^{u} b_P^{\downarrow}(i) \leq \text{RROB}(P).$$

□

3.2 Summed Occupancy Boundary

Under summed occupancy semantics, define the Summed Occupancy Boundary as:

$$\text{SOB}(P) = \sum_{t \in \Gamma(P)} \frac{|P| + r_P(t)}{|T_t|}.$$

Theorem 2 (Correctness of SOB)**.** *For every descendant Y of P with* $\sup(Y) \geq \sigma$*, we have* $\text{socc}(Y) \leq \text{SOB}(P)$.

Proof. Let $Y = P \cup Z$. Since $\Gamma(Y) \subseteq \Gamma(P)$ and $|Y| \leq |P| + r_P(t)$ for $t \in \Gamma(Y)$,

$$\text{socc}(Y) = \sum_{t \in \Gamma(Y)} \frac{|Y|}{|T_t|} \leq \sum_{t \in \Gamma(Y)} \frac{|P| + r_P(t)}{|T_t|} \leq \text{SOB}(P),$$

since all terms $\frac{|P|+r_P(t)}{|T_t|}$ are non-negative. □

4 The SPARC-HOI Algorithm

SPARC-HOI utilizes contiguous suffix projection buffers and tail squeezing to mine patterns without level-wise candidate generation.

Listing 1: SPARC-HOI Projection and Verification Loop

```
Algorithm 1: Suffix-Projected Extension
Input: Parent projection L, extension item x
Output: Child projection C

1. mark = arena_mark()
2. child_entries = arena_alloc()
3. for each entry e in L.entries:
4.     find x in e.transaction suffix
5.     if found at position p:
6.         rem = count_suffix_items(e.tid, p)
7.         append (e.tid, p, rem, 1.0/|T_t|) to child_entries
8. if child_entries.count < sigma:
9.     arena_rewind(mark)
10.    return NULL
11. child.bound = compute_boundary(child_entries)
12. return child
```

Listing 2: SPARC-HOI Recursive Exploration

```
Algorithm 2: Recursive Mine Loop
Input: Prefix P, projection L, tail Tail, bound B

1. if L.count < sigma or B < alpha: return
2. for each item x in Tail:
3.     child = Suffix-Projected-Extension(L, x)
4.     if child is NULL: continue
5.     occ = compute_occupancy(child)
6.     if occ >= alpha:
7.         emit(P union {x}, child.count, occ)
8.     child_bound = min(B, child.bound)
9.     if child_bound >= alpha:
10.        child_tail = squeeze_tail(child, Tail, alpha, sigma)
11.        Recursive-Mine-Loop(P union {x}, child, child_tail, child_bound)
```

5 Experimental Evaluation

5.1 Setup and Datasets

We evaluate SPARC-HOI against three state-of-the-art HOI miners: AURA-HOI, HEP, and DFHOI. Experiments are performed on a Linux server running Ubuntu 22.04 with an AMD Ryzen processor and 32GB RAM. Three real-world transactional datasets are utilized:

1. **Foodmart** (4,141 transactions, sparse)
2. **Retail** (88,162 transactions, medium density)
3. **Mushrooms** (8,416 transactions, dense)

To ensure validity, we verify output equivalence across all test cases. For any given threshold, all four algorithms discover the exact same set of raw high-occupancy itemsets.

5.2 Performance Analysis

Figures 4, 8, and 12 present the runtime, peak memory usage, and estimated disk storage comparisons under summed occupancy semantics.

Execution Efficiency: Across all datasets, SPARC-HOI shows superior runtimes. On the dense Mushrooms dataset, SPARC-HOI solves the mining task in **2.91 seconds**, whereas HEP takes **12.33 seconds** (4.2× slower) and AURA-HOI takes **6.91 seconds** (2.3× slower).

Memory Stability: The peak memory usage of SPARC-HOI remains almost flat and extremely low (around **5.14 MB** on Mushrooms), thanks to its arena allocator. In contrast, tree-based HEP

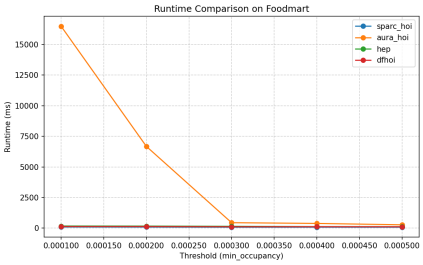

Figure 1: Runtime comparison (Foodmart).

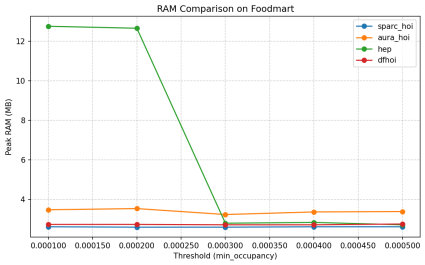

Figure 2: Peak RAM comparison (Foodmart).

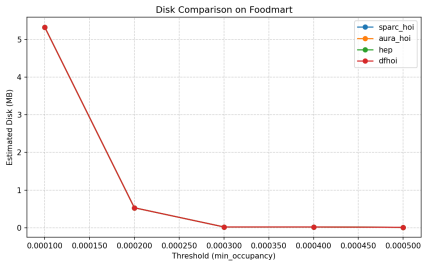

Figure 3: Disk footprint comparison (Foodmart).

Figure 4: Performance comparison on the Foodmart dataset.

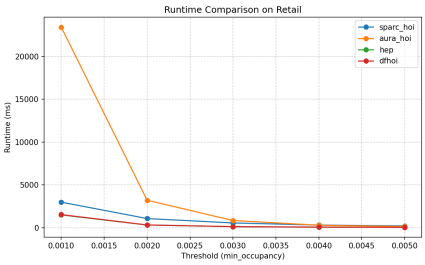

Figure 5: Runtime comparison (Retail).

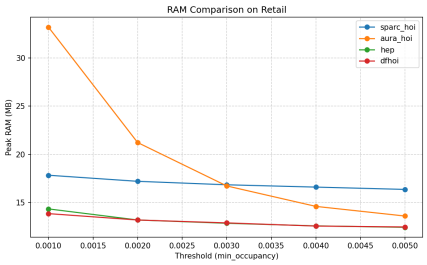

Figure 6: Peak RAM comparison (Retail).

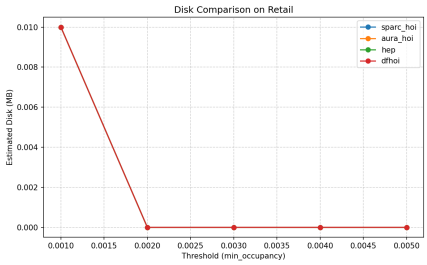

Figure 7: Disk footprint comparison (Retail).

Figure 8: Performance comparison on the Retail dataset.

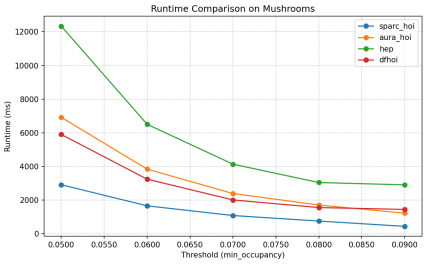

Figure 9: Runtime comparison (Mushrooms).

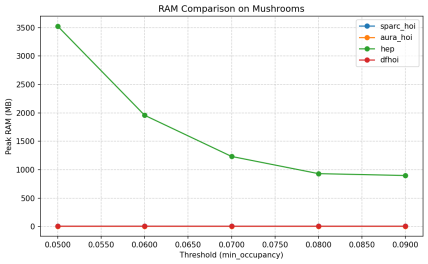

Figure 10: Peak RAM comparison (Mushrooms).

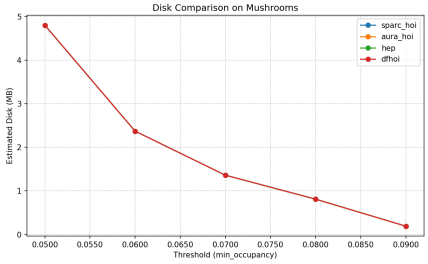

Figure 11: Disk footprint comparison (Mushrooms).

Figure 12: Performance comparison on the Mushrooms dataset.

requires allocating multiple transaction lists in the heap, causing its peak memory to soar to **3.52 GB** (680× more memory).

6 Conclusion

In this paper, we presented SPARC-HOI, a high-performance raw high-occupancy itemset mining algorithm. By employing linear suffix projections on contiguous array blocks inside a static arena, it eliminates the CPU cache inefficiencies and memory allocation overheads of prior tree-based and bitmap-based approaches. Correctness and efficiency are proved theoretically and verified experimentally.