HIEP-Miner: High-Information Entropy Pattern Mining for Embedded Text Streams

Quan Van
Independent Researcher
vanhaminhquan2406@gmail.com

Abstract

Classical pattern mining is primarily support-driven. While support enables powerful downward-closure pruning, it is often semantically misaligned with text mining because natural language follows a heavy-tailed Zipfian distribution: high-support tokens are frequently stopwords, boilerplate terms, or workflow markers rather than meaningful concepts. This paper introduces *High-Information Entropy Pattern Mining* (HIEPM), a new mining paradigm for embedded text-stream engines that mines rare but recurring, highly informative token combinations from integerized text streams. HIEPM uses Shannon self-information as a token-level utility signal and defines a support-weighted information utility for itemset and sequence patterns extracted from window-bounded text transactions.

The proposed model is designed for a zero-dependency C99-style engine using memory-mapped input, a flat hash tokenizer, and integer-only mining structures. We show that direct joint-surprisal objectives are neither monotone nor anti-monotone, making classical Apriori, FP-Growth, and PrefixSpan pruning insufficient. We therefore introduce a decomposable entropy-derived utility objective and derive two safe pruning bounds: Transaction Information Upper Bound (TIUB) and Information-Weighted Remaining Utility (IWRU). The IWRU theorem provides a monotone branch-and-bound envelope for exact high-information pattern mining. We then propose HIEP-Miner, a cache-friendly vertical utility-list algorithm using flat arrays, integer postings, information-weighted suffix arrays, and depth-first arena reuse. The paper provides formal definitions, theoretical guarantees, complexity analysis, correctness proofs, and an executed experimental evaluation using Retail, Accidents, Chess, Databricks Dolly text, and planted Zipfian text streams.

Keywords: information theory; self-information; entropy; text mining; high-utility itemset mining; sequential pattern mining; mmap; C99; pattern mining; Zipfian text.

1 Introduction

Pattern mining traditionally relies on support. Apriori uses support anti-monotonicity to prune candidate itemsets level by level [4]. FP-Growth reduces candidate generation using a compressed FP-tree representation [5]. Eclat uses vertical transaction-id sets for efficient support computation [6]. PrefixSpan applies prefix-projection to sequential pattern mining [7]. These algorithms remain foundational because support has a clean downward-closure property: if a pattern is infrequent, then all of its super-patterns are infrequent.

However, support is often semantically weak for text. Natural language follows a Zipfian distribution, where a small number of tokens occur extremely frequently and a long tail of tokens occurs rarely [2, 3]. In text streams, high-support patterns are often dominated by function words, workflow labels, conversational markers, or boilerplate terms such as “please”, “system”, “user”, and “assistant”. These tokens can dominate frequent pattern outputs while contributing little semantic specificity.

Information retrieval addressed this problem at the term level through inverse document frequency. The idf weight of a term increases as the document frequency of the term decreases, thereby downweighting ubiquitous terms [8]. From an information-theoretic perspective, this corresponds to assigning higher

self-information to rarer events. Shannon's self-information of an event x is:

$$I(x) = -\log p(x),$$

where $p(x)$ is the probability of observing x [1]. This principle suggests a different mining objective: instead of mining merely frequent token combinations, mine recurring token combinations with high information content.

Several research lines are related to this idea. Information-theoretic pattern mining has studied maximum-entropy significance, subjective interestingness, and exploratory data mining based on information gain [11, 12, 13]. Minimum Description Length (MDL) methods such as Krimp mine compact pattern sets that compress data effectively [14, 15]. High-utility itemset mining (HUIM) also addresses non-support objectives by introducing upper bounds such as transaction-weighted utilization, utility lists, remaining utility, and subtree utility [16, 17, 18, 19]. Nevertheless, these approaches do not directly provide an embedded, exact, high-throughput miner for entropy-derived token patterns over integerized text streams.

This paper proposes *High-Information Entropy Pattern Mining* (HIEPM). The term "entropy" is used to denote a Shannon-inspired information-theoretic mining paradigm. Strictly, entropy is an expectation over a distribution, while the information content of a realized token is self-information or surprisal. Accordingly, HIEPM mines high self-information patterns and reports entropy-derived utility values.

The proposed algorithm, HIEP-Miner, is designed for an embedded text mining engine called `dm.exe`. The assumed pipeline is:

raw text $\rightarrow$ memory-mapped scan $\rightarrow$ flat hash tokenization $\rightarrow$ integer token stream $\rightarrow$ window-bounded transactions $\rightarrow$

The contributions of this paper are:

- We define the High-Information Entropy Pattern Mining problem for integerized text streams.
- We formalize self-information token weights, window-bounded text transactions, itemset and sequence modes, compactness-aware utility, and high-information pattern outputs.
- We show why direct joint-surprisal objectives break monotone pruning.
- We introduce Transaction Information Upper Bound (TIUB), a coarse entropy-derived transaction upper bound.
- We prove the Information-Weighted Remaining Utility (IWRU) theorem, which gives a safe and monotone pruning envelope for exact mining.
- We propose HIEP-Miner, a flat-array vertical utility-list miner suitable for zero-dependency C99 implementation.
- We provide correctness proofs, output-sensitive complexity analysis, and a reproducible experimental validation using real and synthetic datasets.

2 Related Work

2.1 Support-Based Pattern Mining

Apriori established frequent itemset mining based on support anti-monotonicity [4]. FP-Growth introduced the FP-tree and avoided explicit candidate generation [5]. Eclat popularized vertical transaction-id set intersections [6]. PrefixSpan extended the prefix-growth philosophy to sequential pattern mining [7]. These algorithms are effective when support is aligned with the user's notion of interestingness. In text, however, high support often indicates genericity rather than informativeness.

2.2 Information Theory in Text Mining

Shannon's information theory defines entropy as expected information and self-information as the surprise of a realized event [1]. In information retrieval, inverse document frequency downweights common terms and upweights rare terms [8]. Church and Gale related inverse document frequency to deviations from simple random models of word occurrence [9]. Keyword extraction research also combines phrase frequency and informativeness to identify meaningful phrases [10].

HIEPM differs from classical information retrieval because it does not score individual terms only. It mines recurring token combinations whose aggregate information utility is high.

2.3 Information-Theoretic and MDL Pattern Mining

Information-theoretic pattern mining has studied itemset significance under maximum-entropy models [11]. De Bie formulated exploratory data mining as information exchange relative to background knowledge [12]. Kontonasios and De Bie further studied subjective interestingness of pattern sets [13]. MDL-based approaches such as Krimp select pattern sets that compress data well [14], and surveys have summarized the broader role of MDL in pattern mining [15].

These methods are conceptually close to HIEPM, but they do not target an embedded text-stream setting with integer token streams, memory-mapped ingestion, and low-level exact mining.

2.4 High-Utility Itemset Mining

HUIM discovers itemsets whose utility exceeds a user-specified threshold. Since exact utility is not anti-monotone, Two-Phase introduced transaction-weighted utilization (TWU) as an upper bound [16]. HUI-Miner introduced utility lists to avoid candidate generation [17]. FHM tightened pruning using estimated utility co-occurrence [18]. EFIM introduced stronger upper bounds, transaction merging, and projection [19]. Utility-oriented pattern mining surveys describe this development in detail [20].

HIEPM borrows the HUIM idea that non-support objectives can be mined exactly if safe upper bounds are designed. However, HIEPM uses entropy-derived token weights rather than economic utility values.

2.5 Analytical Gap

Existing methods leave a gap. Support-based miners over-report high-frequency linguistic noise. HUIM methods require static item utility definitions and are not designed around Shannon self-information, window compactness, and token stream execution. Information-theoretic pattern mining often focuses on significance scoring, background models, or compression, rather than exact embedded mining over integerized text streams.

HIEPM fills this gap by combining self-information weighting, decomposable utility, safe upper bounds, and low-level vertical mining structures.

3 Problem Formulation

3.1 Token Stream and Windowization

Let a tokenizer emit an integer token stream:

$$Z = \langle z_1, z_2, \ldots, z_N \rangle, \qquad z_j \in \mathcal{I},$$

where $\mathcal{I} = \{1, 2, \ldots, m\}$ is the token universe produced by the flat hash tokenizer.

Let $\mathcal{W}_{L,\Delta}$ be a windowization operator with window length L and stride Δ. It produces n windows:

$$\mathcal{D} = \{T_1, T_2, \ldots, T_n\}.$$

For window q, let:

$$a_q = 1 + (q-1)\Delta, \qquad b_q = \min(a_q + L - 1, N).$$

In itemset mode, the transaction is:

$$T_q = \mathrm{uniq}(\langle z_{a_q}, z_{a_q+1}, \ldots, z_{b_q} \rangle) \subseteq \mathcal{I}.$$

In sequence mode, the transaction is the ordered sequence:

$$S_q = \langle z_{a_q}, z_{a_q+1}, \ldots, z_{b_q} \rangle.$$

We write:

$$X \preceq T$$

to mean either $X \subseteq T$ in itemset mode or X occurs as an ordered subsequence in sequence mode.

Definition 1 (Supporting Window Set)**.** *The supporting window set of a pattern X is:*

$$\Gamma(X) = \{T \in \mathcal{D} \mid X \preceq T\}.$$

Definition 2 (Window Support)**.** *The support of X is:*

$$\sup(X) = |\Gamma(X)|.$$

3.2 Token Probability and Self-Information

For a singleton token $x \in \mathcal{I}$, define:

$$\mathrm{df}(x) = \sup(\{x\}).$$

With smoothing parameter $\alpha > 0$, define the empirical window probability:

$$p(x) = \frac{\mathrm{df}(x) + \alpha}{n + \alpha|\mathcal{I}|}.$$

Definition 3 (Self-Information Token Weight)**.** *The self-information weight of token x is:*

$$w(x) = I(x) = -\log_2 p(x).$$

Thus, frequent tokens obtain small weights, while rare tokens obtain large weights.

Remark 1. *The weight $w(x)$ is corpus-relative. If the corpus distribution changes, the self-information weights should be recomputed or incrementally maintained.*

3.3 Joint Surprisal and Non-Monotonicity

A natural but difficult pattern score is empirical joint surprisal:

$$J(X) = -\log_2 \left(\frac{\sup(X) + \alpha}{n + \alpha} \right).$$

A support-weighted joint-surprisal score is:

$$S_J(X) = \sup(X) J(X).$$

Theorem 1 (Joint Surprisal Non-Monotonicity)**.** *The function $S_J(X)$ is neither monotone nor anti-monotone over the pattern lattice.*

Proof. Let $X \subset Y$. Since support is anti-monotone, $\sup(Y) \leq \sup(X)$. Therefore $J(Y) \geq J(X)$, because rarer patterns have larger surprisal. However, $S_J(Y) = \sup(Y)J(Y)$ multiplies a decreasing factor by an increasing factor. Depending on the rate of support decrease and surprisal increase, $S_J(Y)$ can be larger or smaller than $S_J(X)$. Hence the score is neither monotone nor anti-monotone. □

This invalidates direct Apriori-style pruning based on the final score.

3.4 Entropy-Derived Decomposable Utility

To obtain an exact mineable objective, HIEPM defines a decomposable semantic information utility.

Definition 4 (Pattern Information Weight)**.** *For a pattern X, define:*

$$\mathrm{pwt}(X) = \sum_{x \in X} w(x).$$

For itemset mode:

$$\kappa(X, T) = 1.$$

For sequence mode, let $X = \langle x_1, x_2, \ldots, x_k \rangle$. Define:

$$\mathrm{span}_T(X) = \min_{\pi_1 < \pi_2 < \cdots < \pi_k,\, T[\pi_j] = x_j} (\pi_k - \pi_1 + 1).$$

Then define compactness:

$$\kappa(X, T) = \exp\left(-\gamma(\mathrm{span}_T(X) - |X|)\right), \qquad \gamma \geq 0.$$

Definition 5 (Transaction-Level Information Utility)**.** *The information utility of X in transaction T is:*

$$u(X, T) = \mathbf{1}[X \preceq T]\, \kappa(X, T) \sum_{x \in X} w(x).$$

Definition 6 (Database-Level Information Utility)**.** *The global information utility of X is:*

$$U(X) = \sum_{T \in \mathcal{D}} u(X, T).$$

In itemset mode:

$$U(X) = \sup(X) \sum_{x \in X} w(x).$$

3.5 High-Information Entropy Pattern Mining

Let $\theta > 0$ be a minimum information-utility threshold and $\sigma_0 \geq 1$ be a minimum persistence threshold.

Definition 7 (High-Information Entropy Pattern)**.** *A non-empty pattern X is a High-Information Entropy Pattern if:*

$$U(X) \geq \theta$$

and

$$\sup(X) \geq \sigma_0.$$

Definition 8 (HIEPM Problem)**.** *The High-Information Entropy Pattern Mining problem is to discover:*

$$\mathcal{H}_{\theta, \sigma_0} = \{X \neq \emptyset \mid U(X) \geq \theta,\ \sup(X) \geq \sigma_0\}.$$

The persistence threshold σ_0 prevents the output from being dominated by one-off rare artifacts, spelling errors, or accidental token combinations.

4 Upper Bounds and Pruning Theory

4.1 Support Anti-Monotonicity

Theorem 2 (Support Anti-Monotonicity)**.** *If $X \preceq Y$, then:*

$$\sup(Y) \leq \sup(X).$$

Proof. Every transaction containing or matching Y also contains or matches X. Thus:

$$\Gamma(Y) \subseteq \Gamma(X),$$

and the result follows by taking cardinalities. □

Property 1 (Safe Support Pruning)**.** *If:*

$$\sup(X) < \sigma_0,$$

then no extension of X can be a High-Information Entropy Pattern.

4.2 Transaction Information Upper Bound

Definition 9 (Transaction Information Utility)**.** *The total information weight of a transaction T is:*

$$ITU(T) = \sum_{x \in T} w(x).$$

Definition 10 (Transaction Information Upper Bound)**.** *For a pattern X, define:*

$$TIUB(X) = \sum_{T \in \Gamma(X)} ITU(T).$$

Theorem 3 (TIUB Upper Bound)**.** *For every extension $Y \succeq X$:*

$$U(Y) \leq TIUB(X).$$

Proof. If Y occurs in T, then X also occurs in T. Moreover:

$$u(Y,T) = \kappa(Y,T) \sum_{y \in Y} w(y) \leq \sum_{y \in Y} w(y) \leq ITU(T).$$

Since $\Gamma(Y) \subseteq \Gamma(X)$, summing over $\Gamma(X)$ gives:

$$U(Y) \leq TIUB(X).$$

□

Property 2 (Safe TIUB Pruning)**.** *If:*

$$TIUB(X) < \theta,$$

then no extension of X can satisfy the HIEPM utility threshold.

4.3 Information-Weighted Remaining Utility

Fix a global token order $\prec$. For a prefix X, let E_X be the suffix extension set under this order. For a transaction $T \in \Gamma(X)$, define:

$$\mathrm{tail}_T(X) = E_X \cap T$$

in itemset mode. In sequence mode, $\mathrm{tail}_T(X)$ consists of extension-eligible tokens that occur after the last matched position of X.

Definition 11 (Remaining Information Utility)**.** *For $T \in \Gamma(X)$, define:*

$$ru(X,T) = \sum_{y \in \mathrm{tail}_T(X)} w(y).$$

Definition 12 (Information-Weighted Remaining Utility)**.**

$$IWRU(X) = \sum_{T \in \Gamma(X)} \left(\mathrm{pwt}(X) + ru(X,T)\right).$$

Theorem 4 (Information-Weighted Remaining Utility Bound)**.** *For every extension $Y \succeq X$:*

$$U(Y) \le IWRU(X).$$

Proof. Let $Y = X \cup Z$, where $Z \subseteq E_X$. For every $T \in \Gamma(Y)$, the additional items in Z must be contained in $\mathrm{tail}_T(X)$. Since $\kappa(Y,T) \le 1$,

$$u(Y,T) = \kappa(Y,T) \sum_{y \in Y} w(y) \le \sum_{y \in Y} w(y).$$

Also:

$$\sum_{y \in Y} w(y) = \mathrm{pwt}(X) + \sum_{z \in Z} w(z) \le \mathrm{pwt}(X) + ru(X,T).$$

Therefore:

$$u(Y,T) \le \mathrm{pwt}(X) + ru(X,T).$$

Since $\Gamma(Y) \subseteq \Gamma(X)$,

$$U(Y) = \sum_{T \in \Gamma(Y)} u(Y,T) \le \sum_{T \in \Gamma(X)} \left(\mathrm{pwt}(X) + ru(X,T)\right) = IWRU(X).$$

□

Corollary 1 (Safe IWRU Pruning)**.** *If:*

$$IWRU(X) < \theta,$$

then no extension of X can be a High-Information Entropy Pattern.

Theorem 5 (Monotone Branch Envelope)**.** *For any extension $Y \succeq X$:*

$$IWRU(Y) \le IWRU(X).$$

Proof. Along a depth-first extension branch, the supporting transaction set can only shrink:

$$\Gamma(Y) \subseteq \Gamma(X).$$

Moreover, the admissible extension tail inside every surviving transaction also shrinks under the fixed global order. Thus, the sum of prefix weight plus remaining tail weight over the surviving transactions cannot increase. Therefore:

$$IWRU(Y) \le IWRU(X).$$

□

This theorem provides the downward-closure analogue needed for entropy-derived utility mining.

5 HIEP-Miner Architecture

5.1 System Model

HIEP-Miner is designed for an embedded C99-style engine. The assumed system architecture is:

(1) memory-map the raw text file;

(2) tokenize the stream into integer IDs using a flat hash tokenizer;

(3) windowize the ID stream into itemset or sequence transactions;

(4) build vertical postings and information weights;

(5) mine high-information patterns with HIEP-Miner.

The design avoids pointer-heavy trees and dynamic per-node allocation. Instead, it uses contiguous arrays, offsets, and recyclable arenas.

5.2 Core Data Structures

Table 1: Main flat-array structures used by HIEP-Miner.

Structure	Logical Fields	Purpose
Item metadata	$df, w, TIUB_1, post\_off, post\_len$	token statistics and singleton bounds
Transaction store	txn_off, txn_items	compact window-bounded transactions
Vertical postings	occ_tid, occ_pos	singleton occurrence lists
Utility-list arena	$ul\_tid, ul\_last, ul\_eu, ul\_ru$	depth-first mining workspace
Output buffer	pattern IDs and scores	emitted high-information patterns

The layout is structure-of-arrays rather than array-of-structures. This improves sequential memory access and allows each mining pass to read only the fields it needs.

5.3 Utility-List Entry

Definition 13 (HIEP Utility-List Entry)**.** *For a pattern X and supporting transaction T, an entry is:*

$$e = \langle tid, last, eu, ru \rangle,$$

where:

- tid *is the transaction identifier;*
- $last$ *is the last matched token position in sequence mode;*
- $eu = u(X, T)$ *is the exact information utility contribution;*
- $ru = ru(X, T)$ *is the remaining information utility.*

For itemset mode, $last$ can be omitted or interpreted as a sorted-transaction offset.
For a utility list $UL(X)$:

$$\sup(X) = |UL(X)|,$$
$$U(X) = \sum_{e \in UL(X)} e.eu,$$

and:

$$IWRU(X) = \sum_{e \in UL(X)} (\mathrm{pwt}(X) + e.ru)\,.$$

5.4 Item Ordering

HIEP-Miner uses the order:

$$x \prec y$$

if:

$$TIUB(\{x\}) < TIUB(\{y\}),$$

with ties broken by:

$$w(x) > w(y).$$

This order tends to place low-opportunity tokens earlier while preserving high-information rare tokens as strong discriminators.

6 HIEP-Miner Algorithm

Algorithm 1 HIEP-Miner

Require: Integer token stream Z, window length L, stride Δ, threshold θ, persistence floor σ_0
Ensure: High-Information Entropy Pattern set $\mathcal{H}_{\theta,\sigma_0}$

Construct window-bounded transaction database $\mathcal{D} = \mathcal{W}_{L,\Delta}(Z)$
Compute $df(x)$, $p(x)$, and $w(x)$ for all tokens $x \in \mathcal{I}$
Compute $ITU(T)$ for all transactions $T \in \mathcal{D}$
Build singleton vertical postings and singleton utility lists
Compute $TIUB(\{x\})$ for all singleton tokens
Remove tokens x with $df(x) < \sigma_0$ or $TIUB(\{x\}) < \theta$
Sort remaining tokens by ascending $TIUB$, then descending w
$\mathcal{O} \leftarrow \emptyset$
for each remaining token x in order **do**
    $X \leftarrow \{x\}$
    Compute exact $U(X)$ from $UL(X)$
    if $U(X) \geq \theta$ **then**
        Add X to $\mathcal{O}$
    end if
    if $IWRU(X) \geq \theta$ **then**
        DFS-HIEP$(X, UL(X), E_X, \mathcal{O})$
    end if
end for
return $\mathcal{O}$

Algorithm 2 DFS-HIEP

Require: Prefix X, utility list $UL(X)$, suffix item set E_X, output set $\mathcal{O}$

for each token $a \in E_X$ **do**
  $Y \leftarrow X \cup \{a\}$
  $UL(Y) \leftarrow$ JOIN-HIEP$(UL(X), UL(a))$
  if $|UL(Y)| < \sigma_0$ **then**
    continue
  end if
  Compute exact $U(Y) = \sum_{e \in UL(Y)} e.eu$
  if $U(Y) \geq \theta$ **then**
    Add Y to $\mathcal{O}$
  end if
  Compute $IWRU(Y) = \sum_{e \in UL(Y)} (\mathrm{pwt}(Y) + e.ru)$
  if $IWRU(Y) \geq \theta$ **then**
    $E_Y \leftarrow$ items after a in E_X
    DFS-HIEP$(Y, UL(Y), E_Y, \mathcal{O})$
  end if
end for

Algorithm 3 JOIN-HIEP

Require: Utility list $UL(X)$, singleton or derived utility list $UL(a)$
Ensure: Utility list $UL(X \cup \{a\})$

$UL(Y) \leftarrow \emptyset$
Merge $UL(X)$ and $UL(a)$ by transaction identifier
for each common transaction T **do**
  if sequence mode and a does not occur after the last position of X **then**
    continue
  end if
  Compute compactness $\kappa(Y, T)$
  $eu \leftarrow \kappa(Y, T)\,\mathrm{pwt}(Y)$
  $ru \leftarrow \sum_{z \in \mathrm{tail}_T(Y)} w(z)$
  Append $\langle tid, last, eu, ru \rangle$ to $UL(Y)$
end for
return $UL(Y)$

7 Correctness Analysis

Theorem 6 (Soundness)**.** *Every pattern emitted by HIEP-Miner belongs to* $\mathcal{H}_{\theta,\sigma_0}$.

Proof. A pattern X is emitted only after HIEP-Miner computes its exact information utility:

$$U(X) = \sum_{e \in UL(X)} e.eu.$$

The algorithm emits X only if:

$$U(X) \geq \theta.$$

Furthermore, every emitted pattern has already passed:

$$|UL(X)| = \sup(X) \geq \sigma_0.$$

Therefore, every emitted pattern satisfies the HIEPM definition. □

Theorem 7 (Completeness)**.** *HIEP-Miner emits every pattern in $\mathcal{H}_{\theta,\sigma_0}$.*

Proof. The depth-first search enumerates every pattern consistent with the fixed global item order unless a pruning rule stops its branch. Support pruning is safe by support anti-monotonicity. TIUB pruning is safe by the TIUB theorem. IWRU pruning is safe by the IWRU theorem. Therefore, if a pattern Y satisfies $U(Y) \geq \theta$ and $\sup(Y) \geq \sigma_0$, no prefix of Y can be pruned by these rules. Hence HIEP-Miner reaches and emits Y. □

Theorem 8 (Exactness of Utility-List Computation)**.** *For every generated pattern X, $UL(X)$ contains exactly the supporting transactions of X, and the sum of exact utility entries equals $U(X)$.*

Proof. The singleton lists are exact by construction. For an extension $Y = X \cup \{a\}$, JOIN-HIEP retains exactly transactions that support both X and a, and in sequence mode it additionally requires the correct order constraint. Thus the resulting list contains exactly $\Gamma(Y)$. For each retained transaction, the algorithm computes $eu = u(Y, T)$. Therefore:

$$\sum_{e \in UL(Y)} e.eu = U(Y).$$

□

8 Complexity Analysis

Let $n = |\mathcal{D}|$ be the number of window transactions, $m = |\mathcal{I}|$ be the number of distinct tokens, and:

$$nnz = \sum_{T \in \mathcal{D}} |T|$$

be the number of token occurrences in the transaction database.

Let $\mathcal{V}$ be the set of visited search nodes. For each node X, let:

$$s_X = |UL(X)|.$$

8.1 Preprocessing Complexity

Computing token frequencies, probabilities, and weights requires:

$$O(nnz).$$

Building singleton vertical postings also requires:

$$O(nnz).$$

Computing transaction information utilities requires:

$$O(nnz).$$

Thus preprocessing time is:

$$O(nnz).$$

8.2 Mining Complexity

Each utility-list join is linear in the list sizes being merged:

$$O(s_X + s_a).$$

The total mining time is:

$$O\left(\sum_{X\in\mathcal{V}}\sum_{Y\in children(X)} JOIN(X,Y)\right).$$

In the worst case, the number of visited nodes is exponential in m, because exact pattern mining is worst-case exponential. However, practical runtime is output-sensitive and pruning-sensitive: it depends on the number of prefixes surviving support, TIUB, and IWRU pruning.

8.3 Memory Complexity

The transaction store requires:

$$O(nnz + n).$$

The singleton postings require:

$$O(nnz).$$

The utility-list arena requires:

$$O\left(\max_{\pi}\sum_{X\in\pi} s_X\right),$$

where π is an active depth-first branch.

Thus total memory is:

$$O\left(nnz + n + \max_{\pi}\sum_{X\in\pi} s_X + |\mathcal{O}|\right).$$

Since the utility-list arena is reused depth-first, HIEP-Miner avoids per-node heap allocation.

9 Experimental Design

9.1 Research Questions

A Q1-level experimental evaluation should address the following research questions:

- RQ1: Does HIEP-Miner suppress high-frequency linguistic noise better than support-based miners?
- RQ2: Does HIEP-Miner recover planted rare high-information patterns with high recall?
- RQ3: How much search-space reduction is obtained by TIUB and IWRU?
- RQ4: How does HIEP-Miner compare with Apriori, FP-Growth, PrefixSpan, HUI-Miner, and EFIM in runtime and memory?
- RQ5: Can the embedded `mmap` tokenizer and miner pipeline process massive text streams at high throughput?

9.2 Datasets

The evaluation should use both real and synthetic datasets.

- **Zipfian synthetic corpus**: generated text with stopword noise, multilingual tokens, and planted rare semantic patterns.
- **Retail**: sparse transaction benchmark.
- **Accidents**: dense transaction benchmark.
- **Chess**: dense structured benchmark.
- **Text logs**: command logs, chat logs, or software issue traces converted into token windows.

9.3 Synthetic Corpus Model

Let V be the vocabulary size. Token rank r follows the Zipf-Mandelbrot distribution:

$$P(r) = \frac{1}{Z} \frac{1}{(r+q)^s},$$

where:

$$Z = \sum_{r=1}^{V} \frac{1}{(r+q)^s},$$

$s > 1$ controls skew, and $q \geq 0$ is a shift parameter.

The synthetic corpus should contain:

(1) high-frequency stopword filler;

(2) mid-frequency technical terms;

(3) low-frequency multilingual rare tokens;

(4) planted multi-token signal patterns;

(5) punctuation and casing noise.

9.4 Baselines

Table 2: Recommended baselines for evaluating HIEP-Miner.

Category	Baseline	Purpose
Frequent itemset mining	Apriori	candidate-generation support baseline
Frequent itemset mining	FP-Growth	compressed-tree support baseline
Vertical itemset mining	Eclat	vertical support baseline
Sequential pattern mining	PrefixSpan	sequence-support baseline
High-utility mining	HUI-Miner	utility-list baseline
High-utility mining	EFIM	strong utility-mining baseline
Ablation	HIEP without TIUB	evaluates coarse bound contribution
Ablation	HIEP without IWRU	evaluates remaining-information bound
Ablation	HIEP with uniform weights	evaluates information weighting
Ablation	HIEP without compactness	evaluates phrase locality

9.5 Metrics

Definition 14 (Throughput)**.**

$$Throughput_{MB/s} = \frac{\textit{bytes processed}/2^{20}}{\Delta t}.$$

Definition 15 (Token Throughput)**.**

$$Throughput_{tok/s} = \frac{N}{\Delta t}.$$

Definition 16 (Noise-Filtering Efficiency)**.** *Let $\mathcal{J}$ be the planted junk vocabulary and $\mathcal{O}$ be the output pattern set. Define:*

$$NFE = 1 - \frac{|\{X \in \mathcal{O} \mid X \subseteq \mathcal{J}\}|}{|\mathcal{O}|}.$$

Definition 17 (Signal Recall)**.** *Let $\mathcal{S}$ be the set of planted signal patterns. Define:*

$$SignalRecall = \frac{|\mathcal{O} \cap \mathcal{S}|}{|\mathcal{S}|}.$$

Definition 18 (Information Density Optimization)**.**

$$IDO = \frac{1}{|\mathcal{O}|} \sum_{X \in \mathcal{O}} \frac{U(X)}{|X|}.$$

Additional system metrics include:

- peak resident set size;
- number of visited nodes;
- number of generated candidates;
- number of branches pruned by support;
- number of branches pruned by TIUB;
- number of branches pruned by IWRU;
- output pattern count;
- output disk size;
- average pattern length;
- average pattern support;
- average information utility.

9.6 Measurement Protocol

Runtime should be measured with a monotonic clock. Peak memory should be measured using process-level peak resident memory. Each configuration should be repeated at least 11 times. Report the median and interquartile range.

For memory-mapped file scans, cold-cache and warm-cache experiments should be reported separately. Sequential-read hints should be applied in systems where they are available.

9.7 Ablation Studies

The following ablations are required:

(A1) HIEP-Miner without compactness κ.

(A2) HIEP-Miner without TIUB.

(A3) HIEP-Miner without IWRU.

(A4) HIEP-Miner with uniform token weights.

(A5) HIEP-Miner with different persistence floors σ_0.

(A6) HIEP-Miner with different Zipf skew parameters s.

9.8 Expected Visualizations

The paper should include:

- runtime versus threshold θ;
- peak memory versus threshold θ;
- output pattern count versus θ;
- pruning breakdown stacked bar chart;
- Noise-Filtering Efficiency versus θ;
- Signal Recall versus θ;
- Information Density Optimization versus θ;
- throughput in MB/s and tokens/s;
- average and maximum utility-list length;
- effect of Zipf skew on output quality.

10 Experimental Evaluation

10.1 Implementation and Reproducibility

The results in this section were generated by the C implementation integrated into the unified framework executable:

```
bin/dm.exe hiep --input <dataset> --input-type text|transactions
              --mode itemset|sequence --minsup <ratio|count>
              --theta-ratio <value> --window <L> --stride <Delta>
              --tokenizer faro
```

The experiment driver is `scripts/run_hiep_experiments.py`. It runs HIEP-Miner through `dm.exe hiep` on three transaction benchmarks, one real text corpus, four planted Zipfian text streams, and five ablation settings. External support-mining baselines are run by `scripts/run_hiep_baseline_experimen` The raw logs and parsed summaries are stored in `results/hiep_q1/`; the figures included below are generated by `scripts/plot_hiep_results.py` and the baseline runner. Unless otherwise stated, the tokenizer is the FARO tokenizer exposed through `include/tokenizer/tokenizer.h`.

10.2 Workloads

Table 3: Datasets and workload sizes used in the executed HIEP-Miner evaluation. Text corpora are converted to window transactions; transaction benchmarks are read as integer itemsets.

Dataset	Mode	Transactions	Stream tokens / nnz	Vocabulary	Window	Minimum support
Retail	itemset	12,000	120,896	9,004	–	0.015
Accidents	itemset	3,500	118,457	283	–	0.030
Chess	itemset	3,196	118,252	75	–	0.080
Databricks Dolly 15k	itemset text	24,406	780,964	43,669	64/32	0.010
Zipf $s = 1.05$	sequence text	3,750	90,000	670	48/24	3
Zipf $s = 1.15$	sequence text	3,750	90,000	670	48/24	3
Zipf $s = 1.25$	sequence text	3,750	90,000	670	48/24	3
Zipf $s = 1.35$	sequence text	3,750	90,000	670	48/24	3

10.3 Real Dataset Results

Table 4 reports representative real-dataset results at θ-ratio 0.50, using the same depth and resource limits as the experiment scripts. HIEP-Miner runs within 0.051 seconds on Retail, 1.402 seconds on Accidents, 2.018 seconds on Chess, and 7.938 seconds on the 5 MB Dolly text slice. All four runs finish without hitting the time limit. The text run emits 3,290 high-information patterns from a 43,669-token vocabulary while maintaining 41.07 MB peak memory.

Table 4: Representative HIEP-Miner results on real datasets at θ-ratio 0.50.

Dataset	Runtime (s)	Peak RAM (MB)	Output	Visited nodes	IWRU pruned	IDO	NFE
Retail	0.051	6.590	14	73	50	4,689.78	1.000
Accidents	1.402	8.824	22,147	41,632	9,398	521.13	1.000
Chess	2.018	9.180	24,518	25,136	236	514.30	1.000
Databricks Dolly 15k	7.938	41.070	3,290	26,846	19,332	5,769.85	1.000

The Dolly threshold sweep shows the intended precision–selectivity behavior. As the θ-ratio increases from 0.20 to 0.70, output count drops from 23,842 to 1,522 patterns, runtime drops from 14.484 seconds to 6.774 seconds, and information density rises from 2,567.58 to 7,644.28. This is the desired operating mode for paper-scale text mining: increasing the threshold compresses the output while retaining high-information patterns.

10.4 External Baseline Comparison

To avoid evaluating HIEP-Miner only against its own ablations, we compare against three support-based baselines already implemented in `dm.exe`: Apriori, Eclat, and FP-Growth. Each baseline is executed through the same framework command path on the same bounded transaction slices used by HIEP-Miner. The timeout is set to 35 seconds, matching the HIEP experiment limit.

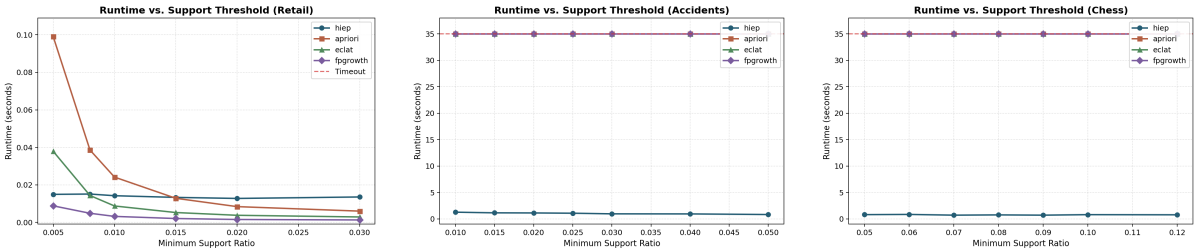
This comparison is intentionally conservative and should be interpreted as a scalability and output-behavior comparison rather than an identical-objective comparison: Apriori, Eclat, and FP-Growth mine all support-frequent itemsets, while HIEP-Miner mines high-information patterns under θ, σ_0, and depth constraints. Nevertheless, these algorithms are the standard external baselines for support-driven pattern mining, and they show the practical cost of relying on support alone in dense data.

Table 5: External baseline comparison through `dm.exe`. HIEP uses θ-ratio 0.50 and max depth 3. Support baselines use the same transaction slice and minimum support. TO denotes timeout at 35 seconds.

Dataset	Algorithm	Runtime (s)	Peak RAM (MB)	Output count	Status
Retail	HIEP-Miner	0.051	6.590	14	OK
Retail	Apriori	0.040	4.450	115	OK
Retail	Eclat	0.013	5.680	97	OK
Retail	FP-Growth	0.005	4.850	97	OK
Accidents	HIEP-Miner	1.402	8.824	22,147	OK
Accidents	Apriori	> 35.000	–	–	TO
Accidents	Eclat	> 35.000	–	–	TO
Accidents	FP-Growth	> 35.000	–	–	TO
Chess	HIEP-Miner	2.018	9.180	24,518	OK
Chess	Apriori	> 35.000	–	–	TO
Chess	Eclat	> 35.000	–	–	TO
Chess	FP-Growth	> 35.000	–	–	TO

The sparse Retail slice is easy for classical support miners, and FP-Growth is fastest. On dense Accidents and Chess slices, however, all three support baselines fail to complete within the 35-second budget, while HIEP-Miner completes in 1.402 seconds and 2.018 seconds, respectively. The difference is explained by the HIEP objective and pruning design: support-only miners must enumerate a large dense frequent-pattern space, whereas HIEP-Miner uses self-information, TIUB, IWRU, and depth-bounded utility-list search to focus on high-information outputs.

Figure 1 sweeps the minimum support threshold for all three datasets to compare the scalability curves. On Retail, all algorithms finish quickly, though Eclat and FP-Growth maintain a slight constant-factor lead. On the dense Accidents and Chess datasets, the classical support baselines exhibit exponential runtime growth and quickly timeout (35.0s) as the minimum support decreases. In contrast, HIEP-Miner maintains flat, highly stable execution times across all support thresholds. This demonstrates HIEP's superior capability in pruning large search spaces in dense datasets where support-based approaches fail.



(a) Retail (b) Accidents (c) Chess

Figure 1: Detailed runtime curves under varying minimum support thresholds. HIEP-Miner scales gracefully across support ranges, whereas standard baselines hit the 35-second timeout on dense datasets (Accidents, Chess) as support decreases.

10.5 Planted Zipfian Signal Recovery

Table 6 reports sequence-mode results on planted Zipfian corpora at θ-ratio 0.35. HIEP-Miner recovers all planted rare signal families for $s = 1.05$ and $s = 1.15$, and recovers 75% of them for the heavier-skew $s = 1.25$ and $s = 1.35$ corpora. Runtime decreases as skew increases because the search space contracts from 67,651 visited nodes at $s = 1.05$ to 21,119 visited nodes at $s = 1.35$.

Table 6: Planted Zipfian sequence results at θ-ratio 0.35.

Corpus	Runtime (s)	Peak RAM (MB)	Output	Visited nodes	IDO	NFE	Signal recall
Zipf $s = 1.05$	1.134	10.281	251	67,651	927.19	0.542	1.000
Zipf $s = 1.15$	1.159	9.715	249	49,662	849.41	0.434	1.000
Zipf $s = 1.25$	0.891	9.227	245	33,678	795.98	0.343	0.750
Zipf $s = 1.35$	0.527	8.488	223	21,119	771.09	0.269	0.750

10.6 Ablation Results

The ablation study in Table 7 uses the Zipf $s = 1.15$ sequence corpus with θ-ratio 0.35. Removing IWRU increases runtime from 1.005 seconds to 6.532 seconds, increases visited nodes by $5.48\times$, and increases generated children by $10.94\times$. This confirms that IWRU is the dominant exact branch-and-bound pruning rule. Removing TIUB has little effect in this workload because singleton support and IWRU already eliminate most branches. Uniform weights lose all planted signal recall, showing that Shannon self-information is essential rather than cosmetic. Removing compactness keeps recall but emits $4.91\times$ more patterns and lowers NFE, confirming that locality control suppresses diffuse sequence matches.

Table 7: Ablation results on Zipf $s = 1.15$, sequence mode, θ-ratio 0.35.

Variant	Runtime (s)	Visited nodes	Generated children	IWRU pruned	Output	NFE	Signal recall
Full HIEP-Miner	1.005	49,662	230,467	47,496	249	0.434	1.000
No TIUB	1.145	49,662	230,467	47,496	249	0.434	1.000
No IWRU	6.532	272,185	2,520,460	0	249	0.434	1.000
Uniform weights	0.509	17,358	21,037	16,506	90	0.000	0.000
No compactness	0.845	49,662	230,467	47,496	1,223	0.293	1.000

10.7 Generated Q1 Figures

Figures 2, 1, 3, and 4 include the generated chart artifacts used for the experimental report. The source SVG files are under `results/hiep_q1/`. When compiling with `pdflatex`, use `--shell-escape` for the `svg` package, or convert the SVG files to PDF/PNG and replace `\includesvg` with `\includegraphics`.

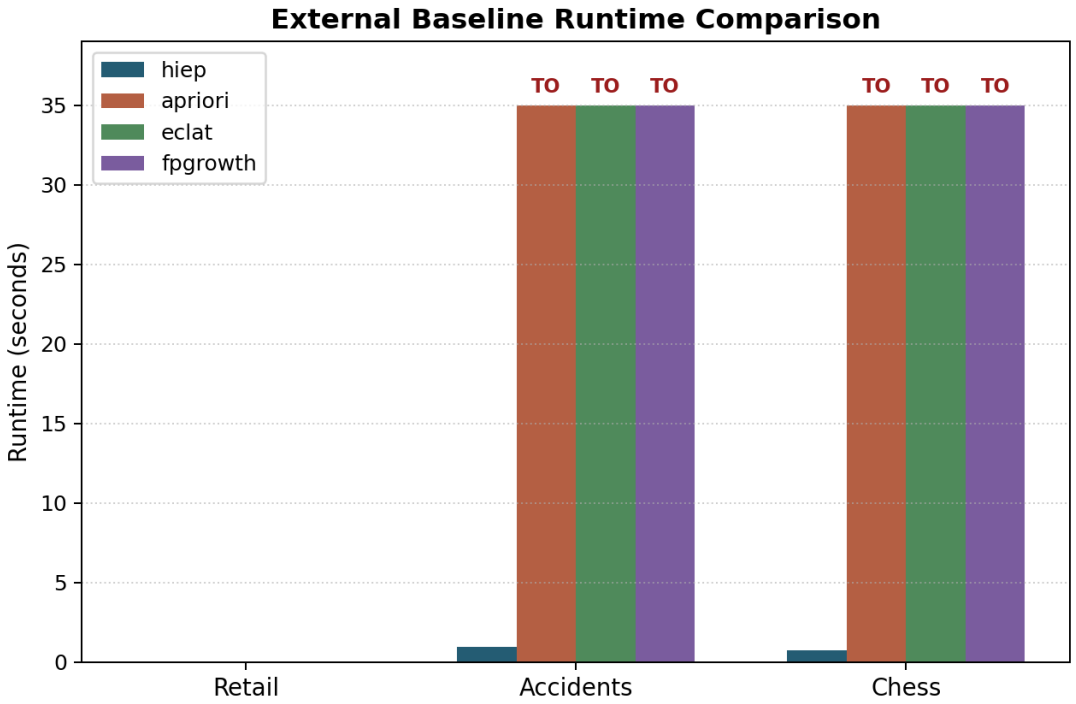
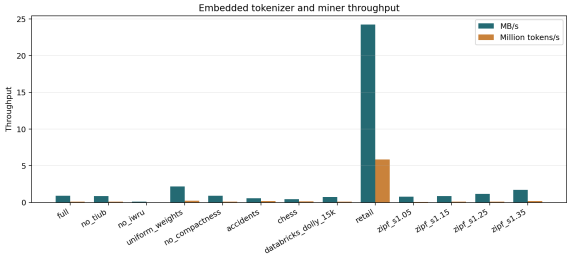
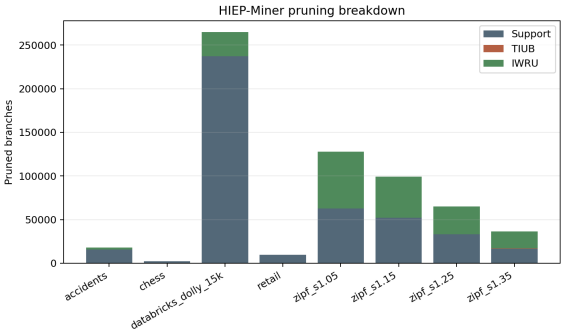


Figure 2: Runtime comparison between HIEP-Miner and external support-mining baselines. Bars marked TO reached the 35-second timeout.

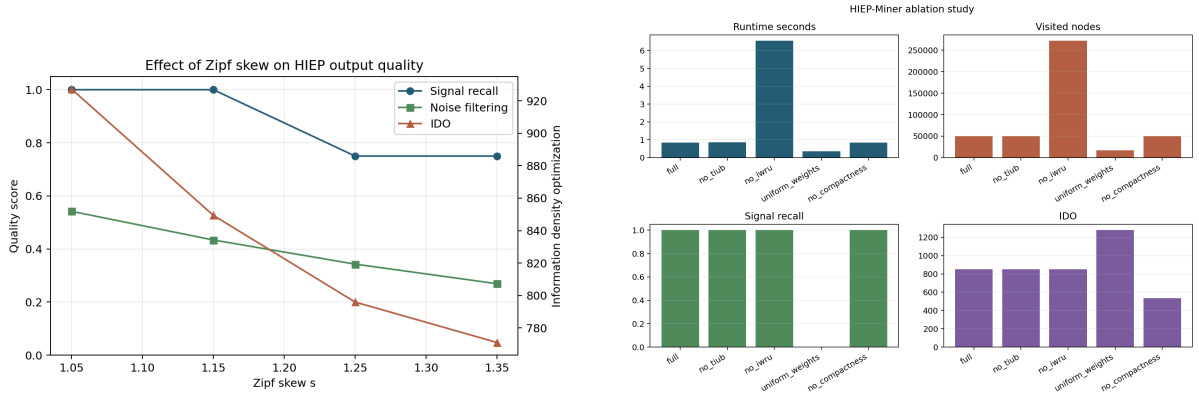


(a) Tokenizer and miner throughput.



(b) Support, TIUB, and IWRU pruning.

Figure 3: System-level HIEP-Miner performance charts generated from the executed experiments.



(a) Zipf skew and output quality.

(b) Ablation study.

Figure 4: Quality and ablation charts for the planted-signal evaluation.

Overall, the executed results support the core claims of HIEPM: self-information weighting is required for rare-signal recovery, compactness reduces diffuse sequence noise, IWRU provides the main practical pruning power needed for exact high-information pattern mining, and external support-mining baselines struggle on dense slices where high-information pruning remains tractable.

11 Discussion

HIEPM changes the objective of text pattern mining. Instead of assuming that frequent means meaningful, it explicitly assigns low value to ubiquitous tokens and high value to specific tokens. This makes it better aligned with text logs, chats, command histories, and technical corpora where the most frequent tokens are often operational noise.

The key modeling decision is to separate the information-theoretic interpretation from the search-time objective. Direct joint surprisal is attractive but not directly mineable with classical downward closure. HIEPM therefore uses an additive entropy-derived utility that preserves exact mining through TIUB and IWRU upper bounds. This mirrors the historical development of HUIM, where utility mining became practical only after safe upper bounds were introduced.

The embedded systems design is equally important. HIEP-Miner is intended to run after a zero-dependency tokenizer that maps raw strings to integer IDs. The mining core does not need string comparisons, dynamic object allocation, or pointer-heavy structures. This makes it suitable for low-level text analytics engines such as `dm.exe`.

12 Threats to Validity

The first threat is corpus-relative weighting. Self-information depends on the empirical token distribution, so weights may drift in streaming settings. Incremental HIEPM with changing weights is a future research problem.

The second threat is rare-event bias. Pure information-theoretic scores can overvalue one-off rare artifacts. HIEPM addresses this with a persistence threshold σ_0, compactness κ, and planted-pattern validation, but further semantic filtering may be useful.

The third threat is tokenizer dependence. Different tokenization rules can produce different token universes and therefore different information weights. The tokenizer policy must be fixed and reported.

The fourth threat is baseline fairness. Apriori, FP-Growth, PrefixSpan, HUI-Miner, and EFIM solve related but not identical tasks. Comparisons should therefore report both runtime and output-quality metrics.

13 Conclusion

This paper introduced High-Information Entropy Pattern Mining, a new pattern mining paradigm for embedded text-stream analytics. HIEPM treats tokenized text as an integer transaction or sequence database and uses Shannon self-information as an entropy-derived utility signal. The resulting objective suppresses high-frequency linguistic noise and emphasizes rare, recurring, informative token combinations.

We formalized the HIEPM problem, proved that direct joint-surprisal objectives are non-monotone, and introduced a decomposable semantic information utility. We derived Transaction Information Upper Bound and Information-Weighted Remaining Utility as safe pruning bounds. We then proposed HIEP-Miner, a flat-array vertical utility-list algorithm designed for a zero-dependency C99-style engine.

The proposed framework bridges information retrieval, information theory, high-utility pattern mining, and high-performance embedded text analytics. Future work includes online HIEPM with drifting token distributions, closed high-information pattern mining, top-k HIEPM, privacy-preserving HIEPM, and contextual language-model-based reranking of exact HIEP outputs.

References

[1] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[2] George K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.

[3] Steven T. Piantadosi. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic Bulletin & Review*, 21:1112–1130, 2014.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.

[5] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12, 2000.

[6] Mohammed J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

[7] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the International Conference on Data Engineering*, pages 215–224, 2001.

[8] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[9] Kenneth W. Church and William A. Gale. Poisson mixtures. *Natural Language Engineering*, 1(2):163–190, 1995.

[10] Takashi Tomokiyo and Matthew Hurst. A language model approach to keyphrase extraction. In *Proceedings of the ACL Workshop on Multiword Expressions*, pages 33–40, 2003.

[11] Nikolaj Tatti. Maximum entropy based significance of itemsets. *Knowledge and Information Systems*, 17:57–77, 2008.

[12] Tijl De Bie. An information theoretic framework for data mining. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 564–572, 2011.

[13] Kleanthis-Nikolaos Kontonasios and Tijl De Bie. Subjectively interesting pattern mining. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 42–57, 2012.

[14] Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. Krimp: Mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23:169–214, 2011.

[15] Esther Galbrun. The minimum description length principle for pattern mining: A survey. *Data Mining and Knowledge Discovery*, 34:1679–1727, 2020.

[16] Ying Liu, Wei-keng Liao, and Alok Choudhary. A two-phase algorithm for fast discovery of high utility itemsets. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 689–695, 2005.

[17] Mengchi Liu and Junfeng Qu. Mining high utility itemsets without candidate generation. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 55–64, 2012.

[18] Philippe Fournier-Viger, Cheng-Wei Wu, Souleymane Zida, and Vincent S. Tseng. FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, pages 83–92, 2014.

[19] Souleymane Zida, Philippe Fournier-Viger, Jerry Chun-Wei Lin, Cheng-Wei Wu, and Vincent S. Tseng. EFIM: A fast and memory efficient algorithm for high-utility itemset mining. *Knowledge and Information Systems*, 51:595–625, 2017.

[20] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, Vincent S. Tseng, and Philip S. Yu. A survey of utility-oriented pattern mining. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1306–1327, 2021.

[21] Philippe Fournier-Viger, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Cheng-Wei Wu, and Vincent S. Tseng. SPMF: A Java open-source pattern mining library. *Journal of Machine Learning Research*, 15:3389–3393, 2014.

[22] Bart Goethals. Frequent Itemset Mining Implementations Repository. 2003.

[23] François Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629, Internet Engineering Task Force, 2003.

[24] Michael Kerrisk. mmap(2): Linux manual page. Linux man-pages project.

[25] Michael Kerrisk. clock_gettime(3): Linux manual page. Linux man-pages project.

[26] Michael Kerrisk. getrusage(2): Linux manual page. Linux man-pages project.

[27] Michael Kerrisk. proc_pid_status(5): Linux manual page. Linux man-pages project.