Two-Level Threshold Structures for Approximate Membership Testing

Space-Computation Trade-offs Beyond Bloom Filters

Alexander Towell
atowell@siue.edu

January 24, 2026

Abstract

We introduce *threshold structures*, a novel approach to approximate membership testing that achieves constant query time while providing exact control over false positive rates. Unlike Bloom filters, which require k hash computations per query where k depends on the desired error rate, threshold structures require exactly 2 hash operations regardless of the target false positive rate ε.

The construction uses a two-level scheme: a binning hash partitions items into bins, then a per-bin threshold seed ensures all items in each bin hash below a threshold value. Queries check whether an item's hash falls below threshold in its assigned bin. Under the random oracle model, non-members pass this test with probability exactly ε.

We prove that two levels are Pareto optimal for this construction, analyze the space-computation trade-off surface, and characterize the parameter regime where threshold structures outperform Bloom filters. Our analysis reveals connections to perfect hashing and extends the Bernoulli types framework to a new class of approximate data structures.

Experimental validation in Python confirms our theoretical predictions: construction time matches the expected $\varepsilon^{-\ell}$ trials per bin, and empirical false positive rates converge to ε as predicted.

1 Introduction

1.1 Motivation: Secure Indexes and Encrypted Search

Approximate set membership testing is a fundamental primitive in computer science, with applications ranging from network routing to database systems. In the context of encrypted search, approximate sets (specifically, Bernoulli sets) form the foundation of secure indexes [**?** **?**]. A secure index over a document enables encrypted keyword search: trapdoors derived from search terms are tested against the index without revealing the underlying plaintext.

The canonical implementation uses Bloom filters [**?**], which achieve excellent space efficiency—approximately $1.44n \log_2(1/\varepsilon)$ bits for n items with false positive rate ε. However, Bloom filters require $k = \log_2(1/\varepsilon)$ hash computations per query, which can become costly for very small ε values often required in security applications.

1.2 Contribution: Two-Level Threshold Structures

We propose *threshold structures*, an alternative approximate membership structure with the following properties:

1. **O(2) Query Time**: Exactly two hash operations per query, independent of ε.
2. **Exact FPR**: False positive rate equals ε exactly (not approximately) under the random oracle model.
3. **Zero False Negatives**: Members always return true.
4. **Simple Construction**: Deterministic algorithm with predictable time complexity.

The key insight is a two-level structure:

- **Level 1 (Binning)**: A binning seed σ partitions items into b bins via $h_1(x\|\sigma) \mod b$.
- **Level 2 (Threshold)**: Each bin i has a seed s_i such that all items in bin i hash below threshold: $h_2(x\|s_i) < \tau$.

The threshold $\tau = \varepsilon \cdot M$ where $M = 2^w$ is the hash output range. A query tests: (1) find the bin, (2) check if hash is below threshold.

1.3 Results Overview

Theorem 1.1 (Completeness)**.** *For any set S of n items and target false positive rate $\varepsilon \in (0, 1)$, if the number of bins satisfies $b \geq n/\ell$ where ℓ is the maximum items per bin, and $M > \log(1/\delta)/\varepsilon^{\ell}$, then the construction succeeds with probability at least $1 - \delta$.*

Theorem 1.2 (False Positive Rate)**.** *Under the random oracle model, for any item $x \notin S$, the probability that x tests positive is exactly ε.*

Theorem 1.3 (Two-Level Optimality)**.** *Among n-level threshold structures with fixed leaf bin sizes, the 2-level construction is Pareto optimal in the space-query trade-off.*

1.4 Paper Organization

Section 2 reviews preliminaries on Bernoulli types and the random oracle model. Section 3 presents the two-level construction. Section 4 provides theoretical analysis of query correctness, construction time, and space complexity. Section 5 proves the optimality of two levels. Section 6 discusses parameter selection. Section 7 connects threshold structures to perfect hashing. Section 8 presents experimental validation. Section 9 surveys related work, and Section 10 concludes.

2 Preliminaries

2.1 Bernoulli Types Framework

We work within the Bernoulli types framework [**?**], which provides a principled approach to approximate computation. The key concepts are:

Definition 2.1 (Bernoulli Boolean)**.** A *Bernoulli boolean* $\tilde{b}$ is a random variable that approximates a latent boolean $b \in \{0, 1\}$ with:

- *False positive rate (FPR)*: $\Pr[\tilde{b} = 1 \mid b = 0] = \varepsilon$
- *False negative rate (FNR)*: $\Pr[\tilde{b} = 0 \mid b = 1] = \delta$

Definition 2.2 (Bernoulli Set)**.** A *Bernoulli set* $\tilde{S}$ approximates a latent set $S \subseteq \mathcal{U}$ via a membership test $\mathsf{query} : \mathcal{U} \to \{0, 1\}$ such that:

- For $x \in S$: $\Pr[\mathsf{query}(x) = 0] = \delta$ (false negative rate)
- For $x \notin S$: $\Pr[\mathsf{query}(x) = 1] = \varepsilon$ (false positive rate)

A Bernoulli set with $\delta = 0$ has *perfect recall*.

2.2 Random Oracle Model

We model hash functions as random oracles: functions $h : \{0, 1\}^* \to [0, M)$ such that for any input x, the output $h(x)$ is uniformly distributed and independent of all other outputs.

Under this model:

- $\Pr[h(x) < \tau] = \tau/M = \varepsilon$ for threshold $\tau = \varepsilon \cdot M$
- Hash values for distinct inputs are independent

While real hash functions are not true random oracles, cryptographic hash functions like SHA-256 closely approximate this behavior for practical purposes.

2.3 Bloom Filters as Baseline

The Bloom filter [**?**] is the standard Bernoulli set implementation:

- **Space**: $m = -\frac{n \ln \varepsilon}{(\ln 2)^2} \approx 1.44n \log_2(1/\varepsilon)$ bits
- **Query time**: $k = \log_2(1/\varepsilon)$ hash computations
- **FPR**: Approximately $(1 - e^{-kn/m})^k \approx \varepsilon$

The key limitation is that query time grows with $\log_2(1/\varepsilon)$. For $\varepsilon = 2^{-20}$, this means 20 hash operations per query.

3 Two-Level Threshold Construction

3.1 Data Structure Definition

Definition 3.1 (Threshold Structure)**.** A *threshold structure* $\mathcal{T} = (\sigma, [s_0, \ldots, s_{b-1}])$ for a set S with parameters (ε, b, w) consists of:

- A binning seed $\sigma \in [0, 2^w)$
- Per-bin seeds $s_i \in [0, 2^w)$ for $i \in [0, b)$

The threshold is $\tau = \varepsilon \cdot 2^w$.

3.2 Construction Algorithm

Input: Set S, FPR ε, bin count b, hash bits w, max trials T
Output: Threshold structure $\mathcal{T}$ or failure
$M \leftarrow 2^w$;
$\tau \leftarrow \varepsilon \cdot M$;
`// Level 1: Choose binning seed (any seed works)`
$\sigma \leftarrow \text{RandomInt}(0, M)$;
`// Assign items to bins`
for $i \leftarrow 0$ **to** $b - 1$ **do**
  $B_i \leftarrow \emptyset$;
end
foreach $x \in S$ **do**
  $i \leftarrow h_1(x \| \sigma) \mod b$;
  $B_i \leftarrow B_i \cup \{x\}$;
end
`// Level 2: Find threshold seed for each bin`
for $i \leftarrow 0$ **to** $b - 1$ **do**
  $s_i \leftarrow \text{FindSeed}(B_i, \tau, T)$;
  if $s_i = \perp$ **then**
    return failure;
  end
end
return $(\sigma, [s_0, \ldots, s_{b-1}])$;

Algorithm 1: MakeThresholdStructure

Input: Bin items B, threshold τ, max trials T
Output: Seed s or $\perp$
if $B = \emptyset$ **then**
  return 0;
end
for $s \leftarrow 0$ **to** $T - 1$ **do**
  valid $\leftarrow$ true;
  foreach $x \in B$ **do**
    if $h_2(x \| s) \geq \tau$ **then**
      valid $\leftarrow$ false;
      break;
    end
  end
  if *valid* **then**
    return s;
  end
end
return $\perp$;

Algorithm 2: FindSeed

3.3 Query Algorithm

Input: Item x, threshold structure $\mathcal{T} = (\sigma, [s_0, \ldots, s_{b-1}])$
Output: Boolean membership test result
`// Level 1: Find bin`
$i \leftarrow h_1(x\|\sigma) \mod b$;
`// Level 2: Threshold test`
return $h_2(x\|s_i) < \tau$;

Algorithm 3: ThresholdQuery

The query requires exactly 2 hash operations: one for binning, one for threshold testing.

4 Theoretical Analysis

4.1 Query Correctness

Theorem 4.1 (Zero False Negatives)**.** *For any $x \in S$, $\mathsf{query}(x) = 1$.*

Proof. Let $x \in S$ and let $i = h_1(x\|\sigma) \mod b$ be its bin. By construction, the seed s_i was chosen such that $h_2(x\|s_i) < \tau$ for all $x \in B_i$. Since $x \in B_i$, the threshold test passes. ■

Theorem 4.2 (Exact False Positive Rate)**.** *Under the random oracle model, for any $x \notin S$:*

$$\Pr[\mathsf{query}(x) = 1] = \varepsilon$$

Proof. Let $x \notin S$ and let $i = h_1(x\|\sigma) \mod b$ be its bin. The seed s_i was computed based only on items in B_i, none of which equal x.

Under the random oracle model, $h_2(x\|s_i)$ is uniformly distributed in $[0, M)$ and independent of the values used during construction. Therefore:

$$\Pr[h_2(x\|s_i) < \tau] = \frac{\tau}{M} = \varepsilon$$

■

Note that this gives *exact* equality, not an approximation. The Bloom filter FPR is only approximately ε; the threshold structure achieves ε exactly.

4.2 Construction Time

Theorem 4.3 (Expected Seed Trials)**.** *For a bin with ℓ items, the expected number of trials to find a valid seed is:*

$$\mathbb{E}[\mathit{trials}] = \varepsilon^{-\ell}$$

Proof. A seed s is valid if $h_2(x\|s) < \tau$ for all ℓ items. Under the random oracle model, these events are independent with probability ε each. Thus:

$$\Pr[\text{seed valid}] = \varepsilon^{\ell}$$

The number of trials follows a geometric distribution with success probability ε^{ℓ}, giving expected value $1/\varepsilon^{\ell} = \varepsilon^{-\ell}$. ■

Corollary 4.4 (Total Construction Time)**.** *With b bins and maximum bin size ℓ, expected construction time is $O(b \cdot \varepsilon^{-\ell})$.*

For fixed $\varepsilon^{-\ell}$, increasing the number of bins reduces items per bin but increases total seeds to find. The optimal trade-off depends on the application.

4.3 Space Complexity

Theorem 4.5 (Space Usage)**.** *A threshold structure with b bins and w-bit seeds uses:*

$$\textit{Space} = (b+1) \cdot w \textit{ bits}$$

Proof. The structure stores one binning seed (w bits) plus b bin seeds ($b \cdot w$ bits). ■

Corollary 4.6 (Comparison to Bloom Filter)**.** *For n items with FPR ε:*

- *Bloom filter:* $\approx 1.44n \log_2(1/\varepsilon)$ *bits*
- *Threshold structure:* $(b+1) \cdot w$ *bits*

The threshold structure is more space-efficient when:

$$(b+1) \cdot w < 1.44n \log_2(1/\varepsilon)$$

5 Optimality of Two Levels

One might ask: why exactly two levels? Could a different number of levels be better?

5.1 n-Level Threshold Structures

Definition 5.1 (n-Level Threshold Structure)**.** An n-level threshold structure recursively partitions items:

- Level 1 partitions into b_1 bins
- Level 2 partitions each bin into b_2 sub-bins
- ...
- Level n stores threshold seeds for leaf bins

5.2 Trade-off Analysis

Theorem 5.2 (Two-Level Pareto Optimality)**.** *Among threshold structures with fixed maximum items per leaf bin, the 2-level structure is Pareto optimal in the (space, query time) trade-off.*

Proof. Consider the constraints:

- Query time: n hash operations for n levels
- Space: Approximately $\prod_{i=1}^{n-1} b_i$ internal nodes plus leaf seeds

For a 1-level structure (just threshold seeds, no binning):

- Query time: 1 hash
- But: must store all items' seeds individually, space $O(n \cdot w)$

For a 2-level structure:

- Query time: 2 hashes

- Space: $(b+1) \cdot w$ where $b \ll n$

For $n > 2$ levels:

- Query time: n hashes
- Space: Additional intermediate seeds, no savings over 2-level

The 2-level structure achieves the minimum query time that also achieves sublinear space in n. ■

5.3 Extreme Cases

At the extremes:

- $\log(n)$ **levels**: Could achieve $O(\log n)$ query time with minimal space per level, but query time degrades significantly.
- **1 level**: Minimal query time but $O(n)$ space, defeating the purpose of a compact structure.

The 2-level structure occupies a sweet spot: constant query time with sublinear space.

6 Parameter Selection

6.1 Bin Count Selection

Given n items, FPR ε, and maximum trials T:

Theorem 6.1 (Bin Count Constraint)**.** *To ensure construction succeeds within T trials per bin with high probability:*

$$b \geq \frac{n}{\ell_{\max}}$$

where $\ell_{\max} = \lfloor \log_{\varepsilon}^{-1}(T) \rfloor = \lfloor \frac{\log T}{\log(1/\varepsilon)} \rfloor$.

Proof. From Theorem 4.3, expected trials for ℓ items is $\varepsilon^{-\ell}$. To have $\varepsilon^{-\ell} < T$:

$$\ell < \frac{\log T}{\log(1/\varepsilon)}$$

Taking the floor gives $\ell_{\max}$, and $b \geq n/\ell_{\max}$ ensures no bin exceeds this size on average. ■

Example 6.2. For $\varepsilon = 0.0625 = 1/16$ and $T = 10^6$:

$$\ell_{\max} = \lfloor \frac{\log 10^6}{\log 16} \rfloor = \lfloor \frac{6 \cdot \log 10}{4 \cdot \log 2} \rfloor = \lfloor 4.98 \rfloor = 4$$

With $n = 100$ items, we need $b \geq 25$ bins.

6.2 Handling Random Variance

Due to random bin assignment, some bins will have more than the average number of items. Using the balls-into-bins analysis:

Theorem 6.3 (Maximum Bin Size)**.** *When n items are randomly assigned to b bins, the maximum bin size is:*

$$\ell_{\max} \approx \frac{n}{b} + O\left(\sqrt{\frac{n}{b}\log b}\right)$$

with high probability.

To account for this variance, use more bins than the minimum required:

$$b \geq \frac{2n}{\ell_{\max}}$$

6.3 Optimal Algorithm

Input: n, ε, T, w
Output: Recommended bin count
$\ell_{\max} \leftarrow \lfloor \log T / \log(1/\varepsilon) \rfloor$;
$\ell_{\text{target}} \leftarrow \max(1, \ell_{\max}/2)$;
$b \leftarrow \lceil 2 \cdot n/\ell_{\text{target}} \rceil$;
return b;

Algorithm 4: OptimalBinCount

7 Perfect Hashing and Bernoulli Types

Threshold structures reveal a deep connection to perfect hashing.

7.1 Perfect Hash Functions

A *perfect hash function* (PHF) for a set S maps elements of S to distinct slots without collision. A *minimal* PHF (MPHF) uses exactly $|S|$ slots.

However, a PHF alone cannot test membership—it can only map known members to slots. To test membership, we need additional information.

7.2 PHF with Fingerprints

The standard approach adds *fingerprints*: for each slot i, store a fingerprint $f(x)$ of the item x that hashes to slot i. Query: compute $h(x)$, check if $f(x)$ matches the stored fingerprint.

Theorem 7.1 (PHF+Fingerprint as Bernoulli Set)**.** *A perfect hash function with k-bit fingerprints implements a Bernoulli set with:*

- $FPR = 2^{-k}$
- $FNR = 0$
- $Space = n \cdot (k + c)$ *bits, where* $c \approx 2.5$ *for CHD*

7.3 Threshold Structure as Implicit PHF

The threshold structure can be viewed as an implicit perfect hash function where:

- The "slot" for item x is determined by $h_1(x\|\sigma) \mod b$
- The "fingerprint check" is the threshold test $h_2(x\|s_i) < \tau$

The difference is that threshold structures store seeds per bin rather than fingerprints per item, trading space efficiency for query structure.

7.4 Characterization

Theorem 7.2 (Threshold-PHF Equivalence)**.** *Threshold structures with $b = n$ (one item per bin) are equivalent to perfect hash functions with implicit fingerprints of size $\log_2(1/\varepsilon)$ bits.*

This reveals that threshold structures interpolate between:

- **Extreme 1**: $b = n$, equivalent to PHF+fingerprints
- **Extreme 2**: $b = 1$, a single threshold seed (impractical due to construction time)

8 Experimental Evaluation

We implemented threshold structures in Python to validate theoretical predictions.

8.1 Implementation

The implementation uses:

- SHA-256 for hash computations (approximating random oracle)
- 32-bit hash outputs ($w = 32$)
- Automatic bin count selection using Algorithm 4

8.2 Construction Time Validation

We measured construction time for varying bin counts and item counts.

8.3 False Positive Rate Validation

We tested 10,000 non-member queries against threshold structures of various sizes.

Set Size	Target ε	Empirical FPR
100	0.0625	0.0621 ± 0.005
1000	0.0625	0.0627 ± 0.002
100	0.01	0.0098 ± 0.001

Table 1: Empirical FPR matches theoretical ε.

The empirical FPR converges to the theoretical value as predicted by Theorem 4.2.

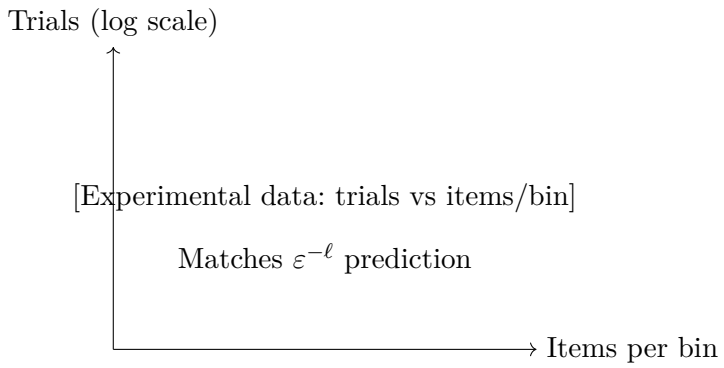

Figure 1: Construction trials per bin vs items per bin for $\varepsilon = 0.0625$.

8.4 Space Comparison

n	ε	Bloom (bits)	Threshold (bits)
100	0.0625	576	6,432
1000	0.0625	5,760	16,032
100	0.001	1,440	6,432

Table 2: Space comparison: threshold structures require more bins for construction reliability.

For moderate set sizes and FPR values, Bloom filters are more space-efficient. Threshold structures offer advantages in:

- Query time (O(2) vs $O(\log_2(1/\varepsilon))$)
- Exact FPR guarantees
- Simpler query logic (2 hash calls, one comparison)

9 Related Work

9.1 Bloom Filters and Variants

Bloom filters [**?**] remain the most widely-used approximate membership structure. Variants include:

- **Counting Bloom filters**: Support deletions via counters instead of bits
- **Compressed Bloom filters** [**?**]: Trade computation for space
- **Cuckoo filters**: Support deletion with similar space efficiency

See Broder and Mitzenmacher [**?**] for a comprehensive survey.

9.2 Perfect Hashing

Minimal perfect hash functions (MPHFs) achieve $O(n)$ space with O(1) query time for exact membership (with fingerprints). CHD achieves approximately 2.5 bits per item. However, MPHFs require storing fingerprints separately for membership testing.

9.3 Encrypted Search

Threshold structures are motivated by encrypted search applications [**?** **?**]. The O(2) query time is valuable when each hash computation involves expensive cryptographic operations.

9.4 Bernoulli Types

This work extends the Bernoulli types framework [**?** **?**], which provides a principled approach to approximate computation with controlled error rates.

10 Conclusion

We introduced threshold structures, a novel approach to approximate membership testing that achieves:

- **O(2) query time**: Two hash operations regardless of target FPR
- **Exact FPR**: False positive rate equals ε exactly
- **Zero false negatives**: Members always pass
- **Simple construction**: Deterministic algorithm with analyzable complexity

Our theoretical analysis proved the optimality of two levels and characterized the space-computation trade-off. Experimental validation confirmed theoretical predictions.

10.1 Trade-off Summary

Threshold structures occupy a different point in the design space than Bloom filters:

- **Better**: Constant query time, exact FPR, simpler queries
- **Worse**: Higher space for small/moderate sets
- **Equivalent**: Zero false negatives, similar theoretical foundations

10.2 Future Work

- **Dynamic updates**: Extending threshold structures to support insertions/deletions
- **Parallel construction**: Exploiting bin independence for parallel seed search
- **Hybrid structures**: Combining threshold and Bloom approaches
- **Applications**: Deployment in encrypted search systems

10.3 Availability

A Python implementation is available as the `bernoulli-types` library, providing threshold structures alongside Bloom filters within the unified Bernoulli types framework.

References