The *Perfect Hash Filter*

A fast and efficient implementation of the *random approximate set* and *random approximate map* abstract data types.

Alexander Towell
atowell@siue.edu

Abstract

We define the semantics of the random approximate set and derive an immutable data structure that implements the s*static* random approximate set, denoted the Perfect Hash Filter. We analyze the theoretical properties of the Perfect Hash Filter, demonstrating that it is a *succinct* data structure that asymptotically obtains the information-theoretic lower bound on the average space complexity. Finally, we provide an implementation and compare it to the theoretical analysis.

Contents

1 Introduction

The Perfect Hash Filter is a data structure that is suitable to implement two types of sets, *positive approximate sets*[**?**] and *oblivious sets*[**?**]. Informally, positive approximate sets generate false positives at specifiable rate and oblivious sets are a type of approximate set with additional *confidentiality* guarantees.

In **??**, we precisely define concept of a set. Then, we define the approximate and oblivious positive sets. In **??**, we derive the Perfect Hash Filter, a data structure that implements the approximate positive set and prove various properties, such as its absolute space efficiency. Following from that, in **??** we show how to tweak the implementation of the Perfect Hash Filter[1] to implement the *oblivious set*.

[1] For instance, by varying the load factor.

2 Approximate and oblivious sets

A set is given by the following definition.

Definition 2.1. *A set is an unordered collection of distinct elements from a universe of elements.*

A countable set is a *finite set* or a *countably infinite set*. A *finite set* has a finite number of elements. For example,

$$\mathcal{S} = \{1, 3, 5\}$$

is a finite set with three elements. A *countably infinite set* can be put in one-to-one correspondence with the set of natural numbers. The cardinality of a set $\mathcal{S}$ is a measure of the number of elements in the set, denoted by

$$|\mathcal{S}| \,. \tag{1}$$

The cardinality of a *finite set* is a non-negative integer and counts the number of elements in the set, e.g.,

$$|\{1, 3, 5\}| = 3 \,.$$

The abstract data type of the immutable *positive* approximate set[**?**] is given by the following definition.

Definition 2.2. *The abstract data type of the* positive approximate set *over a countably infinite universe $\mathcal{U}$ has the following operations defined:*

$$\in\colon\ \mathcal{P}(\mathcal{U}) \times \mathcal{U} \mapsto \{\textbf{true}, \textbf{false}\} \,. \tag{2}$$

Let an element that is selected uniformly at random from the universe $\mathcal{U}$ be denoted by X. *A set $\mathcal{S}^+(\varepsilon)$ is a* positive approximate set *of a set $\mathcal{S}$ with a false positive rate ε if the following conditions hold:*

(i) *$\mathcal{S}$ is a subset of $\mathcal{S}^+$. This condition guarantees that no* false negatives *may occur.*

(ii) *If* X *is* not *a member of $\mathcal{S}$, it is a member of $\mathcal{S}^+$ with a probability ε,*

$$\mathrm{P}\Big[\mathrm{X} \in \mathcal{S}^+ \,\Big|\, \mathrm{X} \notin \mathcal{S}\Big] = \varepsilon \,. \tag{3}$$

The optimal space complexity of *countably infinite* positive approximate sets is given by the following postulate.

Postulate 2.1. *The* optimal *space complexity of a data structure implementing the* positive approximate set *over* countably infinite *universes is independent of the type of elements and depends only the false positive rate ε as given by*

$$-\log_2 \varepsilon \text{ bits/element} \,. \tag{4}$$

The immutable abstract data type of the *oblivious set*[**?**] is given by the following definition.

Definition 2.3. *Assuming that the only information about a set of interested $\mathcal{S}$ is given by another set $\check{\mathcal{S}}$, $\check{\mathcal{S}}$ is an* oblivious positive set *of $\mathcal{S}$ with a false positive rate ε if the following conditions hold:*

(i) *$\check{\mathcal{S}}$ is a positive approximate set of $\mathcal{S}$ with a false positive rate ε.*

(ii) *The false positives are uniformly distributed over $\mathcal{U} \setminus \mathcal{S}$.*

(iii) *There is no efficient way to enumerate the elements in $\check{\mathcal{S}}$.*[2]

[2] That is, the true positives and false positives.

(iv) Any estimator of the cardinality of $\mathcal{S}$ may only be able determine an approximate upper and lower bound, where the uncertainty may be traded for space-efficiency.

The absolute space efficiency of a data structure implementing an oblivious positive set consisting of m true positives with a false positive rate ε and an entropy β is given by

$$\mathtt{E}(\varepsilon, m, \beta) = \mathrm{E}\left[\frac{-(m + \mathrm{X}) \log_2 \varepsilon}{\ell(\mathtt{make\_cipher\_set}(m + \mathrm{X}, \varepsilon))}\right], \tag{5}$$

where

$$\mathrm{X} \sim \mathrm{DU}(0, 2^{\beta} - 1)\,. \tag{6}$$

The relative efficiency of the *optimal* positive oblivious set with entropy β to the *optimal* positive approximate set ($\beta = 0$) has an expectation given by

$$\mathtt{RE}(\,\cdot\,, m, \beta) = 2^{-\beta} \sum_{k=0}^{2^{\beta}-1} \left(1 + \frac{k}{m}\right)^{-1}\,. \tag{7}$$

For a fixed β, as $m \to \infty$ the relative efficiency goes to 1.

3 Perfect hash filters that trade time for space

$\mathtt{ph}_{\mathcal{S}} \colon \mathcal{U} \mapsto [0, \ldots, k]$
$\mathrm{f} \colon \mathbb{N} \mapsto \{0, 1, \ldots, N\}^{\lceil t \rceil}$
Compose two functions
$\mathtt{ph}_{\mathcal{S}} \colon \mathcal{U} \mapsto \{0, 1, \ldots, N\}^{\lceil t \rceil} = \mathrm{f} \circ \mathtt{ph}_{\mathcal{S}}$

4 A C++ implementation

Random positive approximate sets are a well-defined concept. Any data structure T and set of functions dependent upon T required by the concept is an implementation of the model. Any generic algorithm (generic programming) parameterized by R which assumes R models a random approximate set may be applied to the data structure T.

We show a simple implementation.

4.1 Subtypes: dynamic polymorphism

TODO: break up each function/class/maybe method into separate listings and then talk about each separately.

Open set of data types, closed set of operations.

We may also use *dynamic polymorphism*. A dynamic polymorphic C++ interface of the *random approximate set* abstract data type is detailed next.

Assuming the elements of an approximate set are *non-enumerable*, i.e., only membership tests may be performed using the member-of $\in \colon \mathcal{U} \times \mathcal{P}(\mathcal{U})$ interface, set-theoretic operations may be implemented by storing each approximate set *as-is* and composing them together. For example, if we are given two approximate sets $\mathcal{S}^1$ and $\mathcal{S}^2$, the union $\mathcal{S}^1 \cup \mathcal{S}^2$ is implemented by performing membership tests on both $\mathcal{S}^1$ and $\mathcal{S}^2$ and returning **true** if either membership test returns **true**, i.e.,

$$x \in \left(\mathcal{S}^1 \cup \mathcal{S}^2\right) \equiv \left(x \in \mathcal{S}^1\right) \vee \left(x \in \mathcal{S}^2\right)\,. \tag{8}$$

Any set-theoretic composition may be implemented as a combination of unions and complements as described in **??**. A C++ implementation for taking the union of two approximate sets is given by **??** and a C++ implementation for taking the complement of an approximate set is given by **??**.

The following C++ implementation of a Perfect Hash Filter of type X models a first-order rate-distorted set $\mathrm{R}_{\varepsilon}^{\omega}$ of type X where $\varepsilon = 2^{-K}$ and ω is the product of ε and the possible rate-distortion of the (rate-distorted) perfect hash function.

```
template <
        // Expected false positive rate ε = 2^{-K}.
        int K,
        // Models a function of the type string_view ↦ int.
        template PerfectHashFn,
        bool Complement = false
>
class perfect_hash_filter
{
public:
        template <typename Set>
        perfect_hash_filter(
                Set A,   // A is objective set to approximate
                float r, // r is load factor
                uint32_t seed)
{
        h = make_hash_fn(seed);
        ph_ = build_perfect_hash_fn(h_, A, r);

        auto N = ceil(size(A) / r);
        hs_.resize(N);
        for (auto x : A)
        hs_[ph_(seed, x)] = h_(x) % K;
{

auto contains(X x) const
{
        return hs_[ph_(x)] == h_(x) % K;
}

private:
        PerfectHashFn<hash_fn> ph_;
        hash_fn h_;
        vector<uint32_t> hs_;
};

template <int K, typename PerfectHashFn>
auto fpr(perfect_hash_filter<K, PerfectHashFn, false>)
{
        return power<K,2>{}; // 2^{-K}
}
```

```
template <int K, typename PerfectHashFn>
auto fnr(perfect_hash_filter<K, PerfectHashFn, false> A)
{
        return rate_distortion(A.hash_fn())*(1-fpr(A));
}

template <int K, typename PerfectHashFn>
auto fpr(perfect_hash_filter<K, PerfectHashFn, true>)
{
        return rate_distortion(A.hash_fn())*(1-fpr(A)); ???
}

template <int K, typename PerfectHashFn>
auto fnr(perfect_hash_filter<K, PerfectHashFn, true>) { return power<-K,2>{}; }
```

The complement unary operator is given next.

```
template <typename K, bool Complement>
perfect_hash_filter<K,!Complement> operator~(perfect_hash_filter<K,Complement> a)
{
        return perfect_hash_filter<K,!Complement>(a);
}
```

The perfect hash filter, as a *first-order* rate-distorted set, is closed under complements, i.e., it remains a first-order approximation after complementation.

If Container models the concept of a PackedContainer, Container<Integer> requires $O(m/rlog2N)$ bits.

5 Abstract data type of random approximate sets

A *type* is a set and the elements of the set are called the *values* of the type. An *abstract data type* is a type and a set of operations on values of the type. For example, the *integer* abstract data type is defined by the set of integers and standard operations like addition and subtraction.

A *data structure* is a particular way of organizing data and may implement one or more abstract data types. Two different data structures that implement the abstract data type should be *exchangable* in a computer program without changing the outward behavior of the program.

See section 4 to see how a C++ interface for the random approximate set data type may be defined. A popular implementation of an algorithm and data structure that generates approximate sets consistent with the model is known as the *Bloom filter*.

6 Deterministic algorithm

By **????**, each independent observation of $\mathcal{S}^{\pm}$ generates an uncertain number of false positives and false negatives. In practice, this may or may not be the case.

Suppose we have a deterministic algorithm that generates approximate sets with a specified expected false positive rate ε and false negative rate ω. The algorithm, being deterministic, necessarily generates a certain set $\mathcal{X}$ given an objective set $\mathcal{S}$. That is to say, the algorithm is a *total function*, $\mathrm{f}\colon\ \mathcal{P}(\mathcal{U}) \mapsto \mathcal{P}(\mathcal{U})$, with an *image*

$$\mathtt{image}(\mathrm{f}) = \{\, \mathrm{f}(\mathcal{A}) \mid \mathcal{A} \in \mathcal{P}(\mathcal{U}) \,\} \subseteq \mathcal{P}(\mathcal{U})\,. \tag{9}$$

Since the function f may map more than one input to the same output, and some sets in the codomain may not be mapped to by any set in the domain, f is (possibly) a non-surjective, non-injective function.

6.1 Algebraic properties

The only thing we can say with certainty *a priori*[3] about the image of f is that its members are subsets of $\mathcal{U}$ and it contains the empty set $\varnothing$ and universal set $\mathcal{U}$.

Definition 6.1. *A σ-algebra is closed under...*

The image of f is not necessarily a σ-algebra. However, the subsets of $\mathcal{U}$ that may be constructed by countable complements, unions, and intersections for elements of the image is by definition a σ-algebra and is denoted by $\sigma(\mathrm{f})$.

Since $\sigma(\mathrm{f})$ is a set of sets closed under unions, intersections, and complements, it is a Boolean algebra defined by the six-tuple $\left(\sigma(\mathrm{f}), \cup, \cap, \bar{}, \varnothing, \mathcal{U}\right)$, e.g., set-theoretic operations over the above Boolean algebra are of the form

$$\sigma(\mathrm{f}) \times \sigma(\mathrm{f}) \mapsto \sigma(\mathrm{f}) \,. \tag{10}$$

Note that neither *positive* nor *negative* approximate sets are closed under complementation and thus are not Boolean algebras, but rather semi-rings under unions and intersections.

Suppose we have two distinct Boolean algebras, $\left(\sigma(\mathrm{f}), \cup, \cap, \bar{}, \varnothing, \mathcal{U}\right)$ and $\left(\sigma(\mathrm{g}), \cup, \cap, \bar{}, \varnothing, \mathcal{U}\right)$, where f and g are distinct deterministic algorithms that generate approximate sets for $\mathcal{P}(\mathcal{U})$. Set-theoretic operations over both Boolean algebras is the Boolean algebra $\left(\Sigma_{f\cup g}, \cup, \cap, \bar{}, \varnothing, \mathcal{U}\right)$, where $\Sigma_{f\cup g} = \sigma(\sigma(\mathrm{f}) \cup \sigma(\mathrm{g}))$. Note that $\sigma(\mathrm{f}), \sigma(\mathrm{g}) \subseteq \Sigma_{f\cup g}$, so if we take the limit $\lim_{n\to\infty} \Sigma_{f_1\cup\cdots\cup f_n}$, where $f_1, \ldots, f_n$ are distinct mappings, $\Sigma_{f_1\cup\cdots\cup f_n}$ converges to $\mathcal{P}(\mathcal{U})$.

In practice, it is often trivial to implement a large family of random approximate set generators with distinct Boolean algebras given a single implementation, e.g., "randomly" seeding a Bloom filter's hash function. As the size of this family goes to infinity, "randomly" selecting one of them each time we generate an approximate set generates a random sample consistent with the probability distribution given by **??**.

6.2 Compatibility with the probabilistic model

How do we reconcile the fact that the algorithm is a *function* with the *probabilistic model*? In this context, the notion of *probability* quantifies our *ignorance*:

1. Given an objective set $\mathcal{S}$, we do not have complete *a priori* knowledge about the set $\mathcal{X}$ it maps to (the approximate set model only provides *a priori* knowledge about the probability distribution $\mathcal{S}^{\pm}$). We acquire *a posterior* knowledge[4] by applying the algorithm and *observing* $\mathcal{X}$ as its output.

2. Given a set $\mathcal{X}$ that approximates some unknown objective set $\mathcal{S}$, we do not have complete *a priori* knowledge about $\mathcal{S}$. According to the probabilistic model, the only *a priori* knowledge we have is given by the specified *expected* false positive and false negative rates. We may acquire a posteriori knowledge by applying the algorithm to each set in $\mathcal{P}(\mathcal{U})$ and remembering the sets that map to $\mathcal{X}$.[5] However, since f is (possibly) non-injective, one or more sets may map to $\mathcal{X}$ and thus this process may not completely eliminate uncertainty. Additionally, the domain $\mathcal{P}(\mathcal{U})$ has a cardinality $2^{|\mathcal{U}|}$ and thus exhaustive maps are impractical to compute even for relatively small domains.[6] However, we may still reduce our uncertainty by mapping some subset of interest.

[3] A priori knowledge is independent of experience.

[4] A posteriori knowledge is dependent on experience.

[5] If the approximate set is the result of the union, intersection, and complement of two or more approximate sets, then we must consider the closure.

[6] In the case of *countably infinite* domains, it is not even theoretically possible.

Suppose $\mathcal{U}$ is finite. There are $2^{|\mathcal{U}|}$ subsets of $\mathcal{U}$, and each set in $\mathcal{P}(\mathcal{U}) \setminus \{\varnothing, \mathcal{U}\}$ may map (assuming both false positives and false negatives are possible) to *any* of subset. There are a total of

$$2^{2(|\mathcal{U}|-1)} \tag{11}$$

possible functions that map perform this mapping.[7] Each of these functions is compatible with the probabilistic model. For instance, a Bloom filter (positive approximate set) may have a family of hash function that, for a particular binary coding of the elements of a given universal set, maps *every* element in the universal set to the same hash. Thus, for instance, no matter the objective set $\mathcal{X} \subseteq \mathcal{U}$, it will map to $\mathcal{U}$. The Bloom filter had a theoretically sound implementation, but only after empirical evidence was it discovered that it was not suitable. This is an extremely unlikely outcome in the case of large universal sets, but as the cardinality of the universal set decreases, the probability of such an outcome increases. Indeed, at $|U| = 2$, the probability of this outcome is ?.

Thus, *a priori* knowledge, e.g., a theoretically sound algorithm, is not in practice sufficient (although for large universal sets, the probability is negligible). The suitability of an algorithm can only be determined by acquring *a posterior* knowledge.

We could explore the space of functions in the family and only choose those which, on some sample of objective sets of interest, generates the desired expectations for the false positive and false negative rates with the desired variances. Most of them will if constructed in the right sort of way.

A family of functions that are compatible with the probabilistic model is given by observing a particular realization $\mathcal{X} = \mathcal{S}^{\pm}$ and outputting $\mathcal{X}$ on subsequent inputs of $\mathcal{S}$, i.e., caching the output of a *non-deterministic* process that conforms to the probabilistic model. This is essentially how well-known implementations like the Bloom filter work, where the pseudo-randomness comes from mechanical devices like hash functions that approximate random oracles.

The false positive rate of the approximate set corresponding to objective set $\mathcal{X}$ is given by

$$\hat{\varepsilon}(\mathcal{X}) = \frac{1}{n} \sum_{x \in \overline{\mathcal{X}}} \mathbb{1}_{\mathrm{f}(\mathcal{X})}(x)\,, \tag{12}$$

where $n = |\overline{\mathcal{X}}|$.

Let $\mathcal{U}_p$ denote the set of objective sets with cardinality p. The *mean* false positive rate,

$$\overline{\varepsilon} = \frac{1}{|\mathcal{U}_p|} \sum_{\mathcal{X} \in \mathcal{U}_p} \hat{\varepsilon}(\mathcal{X})\,, \tag{13}$$

is an unbiased estimator of ε and the population variance

$$s_\varepsilon^2 = \frac{1}{|\mathcal{U}_p|} \sum_{\mathcal{X} \in \mathcal{U}_p} \hat{\varepsilon}(\mathcal{X})\,, \tag{14}$$

is an unbiased estimator of $\mathrm{V}[\mathcal{E}_n] = \varepsilon(1-\varepsilon)/n$.

Proof. We imagine that the function f caches the output of a *non-deterministic* process that conforms to the probabilistic model. Thus, each time the function maps an objective set $\mathcal{X}$ of cardinality p to its approximation, the algorithm *observes* a realization of $\mathcal{E}_n = \hat{\varepsilon}$. Thus,

$$\overline{\varepsilon} = \frac{1}{|\mathcal{U}_p|} \sum_{\mathcal{X}_\rangle \in \mathcal{U}_p} \hat{\varepsilon}(\mathcal{X}_\rangle) \tag{a}$$

$$= \frac{1}{|\mathcal{U}_p|} \sum_{\mathcal{X}_\rangle \in \mathcal{U}_p} \mathrm{E}\Big[\mathcal{E}_n^{(i)}\Big] = \varepsilon\,. \tag{b}$$

[7] There are a countably infinite number of algorithms that may carry out the computations that implement any particular function.

□

7 Space complexity

If the finite cardinality of a universe is u and the set is *dense* (and the approximation is also dense, i.e., the false negative rate is relatively small), then

$$O(u) \text{ bits} \tag{15}$$

are needed to code the set, which is independent of m, the false positive rate, and the false negative rate.

The lower-bound on the *expected* space complexity of a data structure implementing the *approximate set* abstract data type where the elements are over a *countably infinite* universe is given by the following postulate.

Postulate 7.1. *The* information-theoretic lower-bound *of a data structure that implements the countably infinite* approximate set *abstract data type has an* expected *bit length given by*

$$-(1-\omega)\log_2 \varepsilon \text{ bits/element}, \tag{16}$$

where $\varepsilon > 0$ is the false positive rate and ω is the false negative rate.

The *relative space efficiency* of a data structure X to a data structure Y is some value greater than 0 and is given by the ratio of the bit length of Y to the bit length of X,

$$\mathtt{RE}(X, Y) = \frac{\ell(Y)}{\ell(X)}, \tag{17}$$

where ℓ is the bit length function. If $\mathtt{RE}(X, Y) < 1$, X is less efficient than Y, if $\mathtt{RE}(X, Y) > 1$, X is more efficient than Y, and if $\mathtt{RE}(X, Y) = 1$, X and Y are equally efficient. The absolute space efficiency is given by the following definition.

Definition 7.1. *The absolute space efficiency of a data structure X, denoted by $\mathtt{E}(X)$, is some value between 0 and 1 and is given by the ratio of the bit length of the theoretical lower-bound to the bit length of X,*

$$\mathtt{E}(X) = \frac{\ell}{\ell(X)}, \tag{18}$$

where $\ell(X)$ denotes the bit length of X and ℓ denotes the bit length of the information-theoretic lower-bound. The closer $\mathtt{E}(X)$ is to 1, the more space-efficient the data structure. A data structure that obtains an efficiency of 1 is optimal.[8]

The *absolute* space efficiency of a data structure X implementing an approximate set $\mathcal{S}^{\pm}$ with a false positive rate ε and false negative rate ω is given by

$$\mathtt{E}(X) = \frac{-m(1-\omega)\log_2 \varepsilon}{\ell(X)}, \tag{19}$$

where $m = |\mathcal{S}|$. The most useful sort of asymptotic optimality is with respect to the parameter m.

A well-known implementation of countably infinite *positive approximate set* is the Bloom filter[**?**] which has an expected space complexity given by

$$-\frac{1}{\ln 2}\log_2 \varepsilon \text{ bits/element}, \tag{20}$$

[8] Sometimes, a data structure may only obtain the information-theoretic lower-bound with respect to the limit of some parameter, in which case the data structure *asymptotically* obtains the lower-bound with respect to said parameter.

thus the absolute efficiency of the Bloom filter is $\ln 2 \approx 0.69$. A practical implementation of the *positive approximate set* abstract data type is given by the *Perfect Hash Filter*[**?**], which compares favorably to the Bloom filter in may circumstances.

The *Singular Hash Set*[**?**] is a *theoretical* data structure that optimally implements the abstract data types of the *approximate set* and the *oblivious set*[**?**] abstract data types.

7.1 Space efficiency of *unions* and *differences*

As a way to implement *insertions* and *deletions*, we consider the space efficiency of the set-theoretic operations of unions and differences of approximate sets.

Let $\mathcal{S}_1 = \{x_{j_1}, \ldots, x_{j_m}\}$ and suppose we wish to insert the elements $x_{k_1}, \ldots, x_{k_p}$ into $\mathcal{S}_1$. If X_1 is a mutable object, then an *insertion* operator may be applied on X_1 for each x_{k_i}, $i = 1, \ldots, p$.

Alternatively, if X_1 is immutable, then we may construct another object, X_2, that implements the set $\mathcal{S}_2 = \{x_{k_1}, \ldots, x_{k_p}\}$, and then apply the union function,

$$X_1 \cup X_2 \,. \tag{21}$$

If we replace X_1 and X_2 by objects that implement *positive approximate sets* of $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively, then by **??**, the false positive rate of the resulting approximate set is $\hat{\varepsilon}_1 + \hat{\varepsilon}_2 - \hat{\varepsilon}_1\hat{\varepsilon}_2$.

The space efficiency of this positive approximate set is given by the following theorem.

Theorem 7.1. *Given two countably infinite positive approximate sets $\mathcal{S}_1^+$ and $\mathcal{S}_2^+$ respectively with false positive rates $\hat{\varepsilon}_1$ and $\hat{\varepsilon}_2$, the approximate set $\mathcal{S}_1^+ \cup \mathcal{S}_2^+$, which has an induced false positive rate $\hat{\varepsilon}_1 + \hat{\varepsilon}_2 - \hat{\varepsilon}_1\hat{\varepsilon}_2$, has an expected upper-bound on its absolute efficiency given by*

$$\mathrm{E}(\hat{\varepsilon}_1, \hat{\varepsilon}_2 | \alpha_1, \alpha_2) = \frac{\log_2(\hat{\varepsilon}_1 + \hat{\varepsilon}_2 - \hat{\varepsilon}_1\hat{\varepsilon}_2)}{\alpha_1 \log_2 \hat{\varepsilon}_1 + \alpha_2 \log_2 \hat{\varepsilon}_2} \,, \tag{22}$$

where

$$\begin{aligned} 0 < \alpha_1 &= \frac{|\mathcal{S}_1|}{|\mathcal{S}_1 \cup \mathcal{S}_2|} \leq 1 \,, \\ 0 < \alpha_2 &= \frac{|\mathcal{S}_2|}{|\mathcal{S}_1 \cup \mathcal{S}_2|} \leq 1 \,, \\ 1 &\leq \alpha_1 + \alpha_2 \,. \end{aligned} \tag{23}$$

As $\hat{\varepsilon}_j \to 1$ or $\hat{\varepsilon}_j \to 0$ for $j = 1, 2$, or $(\hat{\varepsilon}_1, \hat{\varepsilon}_2) \to (1, 1)$, the absolute efficiency goes to 0. [9]

Proof. The proof comes in two parts. First, we prove eq. (22), and then we prove the bounds on α_1 and α_2 given by eq. (23).

Let X and Y denote optimally space-efficient data structures that respectively implement positive approximate sets $\mathcal{S}_1^+$ and $\mathcal{S}_2^+$ with false positive rates $\hat{\varepsilon}_1$ and $\hat{\varepsilon}_2$. By **??**, their union has an induced false positive rate given by

$$\hat{\varepsilon}_1 + \hat{\varepsilon}_2 + \hat{\varepsilon}_1\hat{\varepsilon}_2 \,. \tag{a}$$

The information-theoretic lower-bound of the approximate set of $\mathcal{S}_1 \cup \mathcal{S}_2$ with the above false positive rate is given by

$$-|\mathcal{S}_1 \cup \mathcal{S}_2| \log_2(\hat{\varepsilon}_1 + \hat{\varepsilon}_2 + \hat{\varepsilon}_1\hat{\varepsilon}_2) \text{ bits} \,. \tag{b}$$

[9] As $(\hat{\varepsilon}_1, \hat{\varepsilon}_2) \to (0, 0)$, the absolute efficiency depends on the path taken. In most cases, it goes to 0.

Since we assume we only have X and Y and it is not possible to enumerate the elements in either, we must implement their union by storing and separately querying both X and Y. Since X and Y are optimal, $\ell(X) = -|\mathcal{S}_1| \log_2 \hat{\varepsilon}_1$ and $\ell(Y) = -|\mathcal{S}_2| \log_2 \hat{\varepsilon}_2$. Making these substitutions yields an absolute efficiency ???

$$\mathtt{E} = \frac{|\mathcal{S}_1 \cup \mathcal{S}_2| \log_2(\hat{\varepsilon}_1 + \hat{\varepsilon}_2 + \hat{\varepsilon}_1\hat{\varepsilon}_2)}{|\mathcal{S}_1| \log_2 \hat{\varepsilon}_1 + |\mathcal{S}_2| \log_2 \hat{\varepsilon}_2} \,. \tag{c}$$

Letting

$$\alpha_1 = \frac{|\mathcal{S}_1|}{|\mathcal{S}_1 \cup \mathcal{S}_2|} \text{ and } \alpha_2 = \frac{|\mathcal{S}_2|}{|\mathcal{S}_1 \cup \mathcal{S}_2|} \,, \tag{d}$$

we may rewrite eq. (c) as

$$\frac{\log_2(\hat{\varepsilon}_1 + \hat{\varepsilon}_2 - \hat{\varepsilon}_1\hat{\varepsilon}_2)}{\alpha_1 \log_2 \hat{\varepsilon}_1 + \alpha_2 \log_2 \hat{\varepsilon}_2} \,. \tag{22 revisited}$$

In the second part of the proof, we prove the bounds on α_1 and α_2 as given by eq. (23). Both α_1 and α_2 must be non-negative since each is the ratio of two positive numbers (cardinalities). If $|\mathcal{S}_1| \ll |\mathcal{S}_2|$, then $\alpha_1 \approx 0$. If $\mathcal{S}_1 \supset \mathcal{S}_2$, then $\alpha_1 = 1$. A similar argument holds for α_2. Finally,

$$\alpha_1 + \alpha_2 = \frac{|\mathcal{S}_1| + |\mathcal{S}_2|}{|\mathcal{S}_1 \cup \mathcal{S}_2|} \tag{e}$$

has a minimum value by assuming that $\mathcal{S}_1$ and $\mathcal{S}_2$ are disjoint sets (i.e., their intersection is the empty set), in which case

$$\alpha_1 + \alpha_2 = \frac{|\mathcal{S}_1| + |\mathcal{S}_2|}{|\mathcal{S}_1| + |\mathcal{S}_2|} = 1 \,. \tag{f}$$

□

See **??** for a contour plot of the expected lower-bound as a function of $\hat{\varepsilon}_1$ and $\hat{\varepsilon}_2$. As $\hat{\varepsilon}_1 \to 0$ or $\hat{\varepsilon}_2 \to 0$, the efficiency goes to 0.

The lower-bound on the efficiency of the union of approximate sets is given by the following corollary.

Corollary 7.1.1. *Given two positive, non-enumerable approximate sets with false positive rates $\hat{\varepsilon}_1$ and $\hat{\varepsilon}_2$, their union is an approximate set that has an efficiency that is expected to be greater than the lower bound given by*

$$\min \mathtt{E}(\hat{\varepsilon}_1, \hat{\varepsilon}_2) = \frac{\log_2(\hat{\varepsilon}_1 + \hat{\varepsilon}_2 - \hat{\varepsilon}_1\hat{\varepsilon}_2)}{\log_2 \hat{\varepsilon}_1\hat{\varepsilon}_2} \,. \tag{24}$$

Corollary 7.1.2. *If $\varepsilon_1 = \varepsilon_2 = \varepsilon$, then the absolute efficiency is given by*

$$\mathtt{E}(\hat{\varepsilon}|\alpha) = \left(1 + \frac{\log_2(2 - \hat{\varepsilon})}{\log_2 \hat{\varepsilon}}\right)\left(1 - \frac{\alpha}{2}\right) \,, \tag{25}$$

where

$$0 \leq \alpha = \frac{|\mathcal{S}_1 \cap \mathcal{S}_2|}{|\mathcal{S}_1 \cup \mathcal{S}_2|} \leq 1 \,, \tag{26}$$

which is a monotonically decreasing function with respect to ε and α with limits given by $\lim_{\hat{\varepsilon}\to 0} \mathtt{E}(\varepsilon) = 1$ and $\lim_{\varepsilon\to 1} \mathtt{E}(\varepsilon) = 0$.

See fig. 1 for a graphic illustration.

Figure 1: Expected lower-bound on efficiency of the union of two approximate sets with the same false positive rate ε, neither of which can be enumerated.