Combined Search and Encoding for Seeds, with an Application to Minimal Perfect Hashing

Hans-Peter Lehmann
Karlsruhe Institute of Technology, Germany

Peter Sanders
Karlsruhe Institute of Technology, Germany

Stefan Walzer
Karlsruhe Institute of Technology, Germany

Jonatan Ziegler
Karlsruhe Institute of Technology, Germany

Abstract

Randomised algorithms often employ methods that can fail and that are retried with independent randomness until they succeed. Randomised data structures therefore often store indices of successful attempts, called seeds. If n such seeds are required (e.g., for independent substructures) the standard approach is to compute for each $i \in [n]$ the smallest successful seed S_i and store $\vec{S} = (S_1, \ldots, S_n)$.

The central observation of this paper is that this is not space-optimal. We present a different algorithm that computes a sequence $\vec{S'} = (S'_1, \ldots, S'_n)$ of successful seeds such that the entropy of $\vec{S'}$ undercuts the entropy of $\vec{S}$ by $\Omega(n)$ bits in most cases. To achieve a memory consumption of $\mathrm{OPT} + \varepsilon n$, the expected number of inspected seeds increases by a factor of $\mathcal{O}(1/\varepsilon)$.

We demonstrate the usefulness of our findings with a novel construction for minimal perfect hash functions that, for n keys and any $\varepsilon \in [n^{-3/7}, 1]$, has space requirement $(1+\varepsilon)\mathrm{OPT}$ and construction time $\mathcal{O}(n/\varepsilon)$. All previous approaches only support $\varepsilon = \omega(1/\log n)$ or have construction times that increase exponentially with $1/\varepsilon$. Our implementation beats the construction throughput of the state of the art by more than two orders of magnitude for $\varepsilon \leq 3\%$.

2012 ACM Subject Classification Theory of computation → Randomness, geometry and discrete structures; Theory of computation → Data compression; Theory of computation → Data structures design and analysis

Keywords and phrases Random Seed, Encoding, Bernoulli Process, Backtracking, Perfect Hashing

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.109

Related Version *Extended Version*: `https://arxiv.org/abs/2502.05613` [24]

Supplementary Material *Software (Source code)*: `https://github.com/ByteHamster/ConsensusRecSplit` [23], archived at `swh:1:dir:eeef738e12bea287eae54a77ce81241587a91767`
Software (Comparison with competitors): `https://github.com/ByteHamster/MPHF-Experiments` [17], archived at `swh:1:dir:7b02eda8ffa5a9d61a086ff2fc34fb0fbdc3eff4`

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500). This work was also supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF).

Acknowledgements This paper is based on the bachelor's thesis of Jonatan Ziegler [34].

1 Introduction

The construction of randomised data structures can sometimes fail, so the data structures need to store indices of successful attempts called *seeds*.[1] In this paper, we present a technique that finds and encodes successful seeds in a space-efficient and time-efficient way.

When searching a single seed, we may imagine an infinite sequence of i.i.d. Bernoulli random variables, called a *Bernoulli process*, that indicate which natural numbers lead to success and that can be inspected one by one. To model the task of finding a *sequence* of seeds we use a *sequence* of Bernoulli processes as follows. In this paper $[n] \coloneqq \{1, \dots, n\}$ and $\mathbb{N}_0 \coloneqq \mathbb{N} \cup \{0\}$.

▶ **Definition 1** (Seed Search and Encoding Problem (SSEP)). *Let $n \in \mathbb{N}$ and $p_1, \dots, p_n \in (0, 1]$. Given for each $i \in [n]$ a Bernoulli process $\vec{B}^{(i)} = (B^{(i)}_j)_{j \in \mathbb{N}_0}$ with parameter p_i, the task is to craft a bitstring M that encodes for each $i \in [n]$ a successful seed $S_i \in \mathbb{N}_0$, i.e. $B^{(i)}_{S_i} = 1$.*

Our primary goals are to minimise the length of M in bits, and to minimise for each $i \in [n]$ the number T_i inspected Bernoulli random variables from $\vec{B}^{(i)}$. Moreover, the time to decode S_i given $p_1, \dots, p_n$, M and $i \in [n]$ should be constant. The optimal values for the expectation of $|M|$ and T_i are the following (for a proof see the full version of this paper [24]).

$$M^{\mathrm{OPT}} = \sum_{i=1}^{n} \log_2(1/p_i) \text{ and } T_i^{\mathrm{OPT}} = 1/p_i \text{ for } i \in [n]. \tag{1}$$

1.1 Ad-Hoc Solutions

We briefly consider two natural strategies for solving the SSEP to build intuition. The obvious strategy (MIN) is to encode the smallest successful seeds, i.e. define

$$S_i \coloneqq \min\{j \in \mathbb{N}_0 \mid B^{(i)}_j = 1\} \text{ and } \vec{S}_{\mathrm{MIN}} \coloneqq (S_1, \dots, S_n) \tag{MIN}$$

and let M be an optimal encoding of $\vec{S}_{\mathrm{MIN}}$. Perhaps surprisingly, this requires $\Omega(n)$ bits more than the optimum if $p_1, \dots, p_n$ are bounded away from 1.

▶ **Lemma 2** (MIN is fast but not space-efficient, proof in full version [24]). *If M encodes $\vec{S}_{\mathrm{MIN}}$ then $\mathbb{E}[T_i] = T_i^{\mathrm{OPT}}$ for all $i \in [n]$ but $\mathbb{E}[|M|] \geq M^{\mathrm{OPT}} + \sum_{i=1}^{n} 1 - p_i$.*

Another simple strategy (UNI) is to encode the smallest seed that is simultaneously successful for all Bernoulli processes, i.e. we define

$$S_* = \min\{j \in \mathbb{N}_0 \mid \forall i \in [n] : B^{(i)}_j = 1\} \text{ and } S_i = S_* \text{ for all } i \in [n]. \tag{UNI}$$

This yields a bitstring M of minimal length but requires total time $\sum_{i=1}^{n} T_i = \exp(\Omega(n))$ if $p_1, \dots, p_n$ are bounded away from 1.

▶ **Lemma 3** (UNI is space-efficient but slow, proof in full version [24]). *If M encodes S_* then $\mathbb{E}[|M|] = M^{\mathrm{OPT}} + \mathcal{O}(1)$ but $\mathbb{E}[\sum_{i=1}^{n} T_i] \geq \prod_{i=1}^{n} 1/p_i \gg \sum_{i=1}^{n} T_i^{\mathrm{OPT}}$.*

[1] Throughout this paper, we use the *Simple Uniform Hashing Assumption*, as do many works before us [5, 29, 28, 22, 21, 14, 7]. This is equivalent to saying that we assume access to an infinite sequence of uniformly random and independent hash functions, each identified by its index (seed). The assumption is an adequate model for practical hash functions and can be justified in theory using the "split-and-share" trick [6].

1.2 The Consensus Algorithm

We present an algorithm for the SSEP designed to simultaneously get close to both optima. We call it *Combined Search and Encoding for Successful Seeds*, or Consensus for short. We summarise the guarantees of the stronger of two variants here.

▶ **Theorem 4** (Main result, informal version of Theorem 7). *For any $\varepsilon > 0$ the full Consensus algorithm with high probability solves the SSEP with $|M| \leq M^{\mathrm{OPT}} + \varepsilon n + \mathcal{O}(\log n)$ and $\mathbb{E}[T_i] = \mathcal{O}(T_i^{\mathrm{OPT}}/\varepsilon)$ for all $i \in [n]$. Each seed S_i is a bitstring of fixed length found at a predetermined offset in M.*

Note that if the geometric mean of $p_1, \ldots, p_n$ is at most $\frac{1}{2}$ then $M^{\mathrm{OPT}} \geq n$ and the leading term εn in our space overhead is at most $\varepsilon M^{\mathrm{OPT}}$.[2]

Intuition. First reconsider the MIN-strategy. It involves the sequence $\vec{S}_{\mathrm{MIN}} = (S_1, \ldots, S_n)$ with some entropy $H(\vec{S}_{\mathrm{MIN}})$. There are two issues with storing $\vec{S}_{\mathrm{MIN}}$. The superficial issue is that each number in the sequence follows a geometric distribution, and it is not obvious how to encode $\vec{S}_{\mathrm{MIN}}$ in space close to $H(\vec{S}_{\mathrm{MIN}})$ while preserving fast random access to each component.[3] The deeper issue is that $\vec{S}_{\mathrm{MIN}}$ is not what a space-optimal approach should encode in the first place. This is demonstrated by the fact that even an optimal encoding of $\vec{S}_{\mathrm{MIN}}$ needs space $H(\vec{S}_{\mathrm{MIN}})$, which UNI undercuts by $\Omega(n)$ bits.

Consensus deals with both issues simultaneously. The idea is to store for each $i \in [n]$ a natural number $1 \leq \sigma_i \leq 2^{\varepsilon}/p_i$. This allows for fixed width encoding of σ_i using $\varepsilon + \log_2 1/p_i$ bits. If we simply used $S_i = \sigma_i$ then the expected number of successful choices for σ_i would be $2^{\varepsilon} > 1$. The problem would be, of course, that the actual number of successful choices might be zero for many $i \in [n]$. So instead we interpret σ_i as a *seed fragment* and define S_i to be a concatenation of seed fragments up to and including σ_i. This way, if we run out of choices for σ_i, we can hope that a different successful choice for earlier seed fragments will allow for a successful choice for σ_i as well. We can construct the sequence $(\sigma_1, \ldots, \sigma_n)$ using backtracking. We explain our algorithm in detail in Section 2. The main technical challenge is to bound the additional cost this brings.

1.3 Motivation: Seed Sequences in Randomised Data Structures

Randomised data structures frequently use seeds to initialise pseudo-random number generators or as additional inputs to hash functions. Since a seed normally counts how often a construction has been restarted, a seed $S \in \mathbb{N}_0$ implies that work $\Omega(S+1)$ has been carried out, meaning any reasonably fast algorithm yields seeds that are only a few bits to a few bytes in size. This does not, however, imply that seeds contribute only negligibly to overall memory consumption. That is because some data structures partition their input into many small parts called *buckets* and store a seed for each of these buckets. The seeds may even make up the majority of the stored data. This is the case for many constructions of perfect hash functions, but also occurs in a construction of compressed static functions [2], which, in turn, enable space-efficient solutions to the approximate membership problem and the relative membership problem. The way we phrased the SSEP implicitly assumes that buckets are handled independently. We discuss a slight generalisation in the full version [24].

[2] It is possible to get $\mathbb{E}[|M|] \leq M^{\mathrm{OPT}} + \varepsilon \min(M^{\mathrm{OPT}}, n) + \mathcal{O}(\log n)$ and $\mathbb{E}[T_i] = \mathcal{O}(T_i^{\mathrm{OPT}}/\varepsilon)$ in general. This requires work if many p_i are close to 1 such that $M^{\mathrm{OPT}} = o(n)$. In that case we can "summarise" subsequences of $p_1, \ldots, p_n$ using essentially the UNI strategy and then apply our main theorem to a shorter sequence $\tilde{p}_1, \ldots, \tilde{p}_{n'}$ with $n' \leq M^{\mathrm{OPT}} + \mathcal{O}(1)$. We decided against integrating this improvement into our arguments as it interferes with clarity.

[3] The standard solution [32] using Golomb-Rice coding [11, 31] comes with $\Omega(n)$ bits of overhead.

Minimal Perfect Hash Functions. As an application of CONSENSUS we discuss the construction of minimal perfect hash functions (MPHFs). An MPHF on a set X of keys maps these keys to a range $[|X|]$ without collisions. The function does not necessarily have to store X explicitly. The space lower bound is $\mathrm{OPT}_{\mathrm{MPHF}} = |X| \log_2(e) - \mathcal{O}(\log |X|)$ bits assuming the universe of possible keys has size $\Omega(|X|^2)$ [1, 26, 25, 9]. While a succinct construction has long been known [12], its space overhead is fixed to $\varepsilon = \Theta(\frac{\log^2 \log n}{\log n})$ and large in practice. Many recent practical approaches for constructing MPHFs involve a brute force search for seed values that map certain subsets of the keys in favourable ways [30, 7, 14, 22, 21]. Those that do, all combine aspects of the MIN-strategy (store smallest working seeds) and the UNI-strategy (let a single seed do lots of work) and achieve a space budget of $(1+\varepsilon) \cdot \mathrm{OPT}_{\mathrm{MPHF}}$ only in time $|X| \cdot \exp(\Omega(1/\varepsilon))$ (see Section 5.1). In this paper we break this barrier for the first time. While we give the details in Section 5, the clean version of our result is the following.

▶ **Theorem 5** (Bucketed CONSENSUS-RecSplit). *For any $\varepsilon \in [n^{-3/7}, 1]$ there is an MPHF data structure with space requirement $(1 + \mathcal{O}(\varepsilon))\mathrm{OPT}_{\mathrm{MPHF}}$, expected construction time $\mathcal{O}(n/\varepsilon)$ and expected query time $\mathcal{O}(\log 1/\varepsilon)$.*

1.4 Overview of the Paper

The structure of the paper is as follows. In Section 2, we explain CONSENSUS in detail, as well as state the main theoretic results. In Sections 3 and 4 we then prove the results. We give an application of CONSENSUS to minimal perfect hashing in Section 5 and also briefly evaluate our practical implementation. Finally, we conclude the paper in Section 6.

2 Combined Search and Encoding for Successful Seeds

In this section we present two variants of our CONSENSUS algorithm. The first aims for conceptual clarity but produces seeds of length $\Omega(n)$ bits that cannot be decoded efficiently. The second is more technical but solves this problem.

In the following we always assume that $n \in \mathbb{N}$, $p_1, \ldots, p_n \in (0, 1]$ and $\varepsilon \in (0, 1]$ are given and $\vec{B}^{(i)} = (B_j^{(i)})_{j \in \mathbb{N}_0}$ is a Bernoulli process with parameter p_i for all $i \in [n]$.

A Branching Process. Let $k_1, \ldots, k_n \in \mathbb{N}$ be a sequence of numbers.[4] Consider the infinite tree visualised in Figure 1. It has $n+2$ layers, indexed from 0 to $n+1$. The root in layer 0 has an infinite number of children and nodes in layer $1 \leq i \leq n$ have k_i children each, with 0-based indexing in both cases. Edges between layer i and $i+1$ are associated with the variables $B_0^{(i)}, B_1^{(i)}, B_2^{(i)}, \ldots$ from top to bottom where $B_j^{(i)} = 0$ indicates a *broken* edge depicted as a dotted line. For convenience, we make another definition. For any $0 \leq \ell \leq n$ and any sequence $(\sigma_0, \sigma_1, \ldots, \sigma_\ell) \in \mathbb{N}_0 \times \{0, \ldots, k_1 - 1\} \times \ldots \times \{0, \ldots, k_\ell - 1\}$ consider the node reached from the root by taking in layer $0 \leq i \leq \ell$ the edge to the σ_i-th child. We define $\sigma_0 \circ \ldots \circ \sigma_\ell$ to be the index of the reached node within layer ℓ.

The Simplified CONSENSUS Algorithm. In CONSENSUS, we search for a non-broken path from the root of the tree to a leaf node using a DFS, see Algorithm 1. Note that such a path exists with probability 1 because the root has infinitely many children. The path is given by

[4] It may help to imagine that $k_i \approx 2^\varepsilon / p_i$.

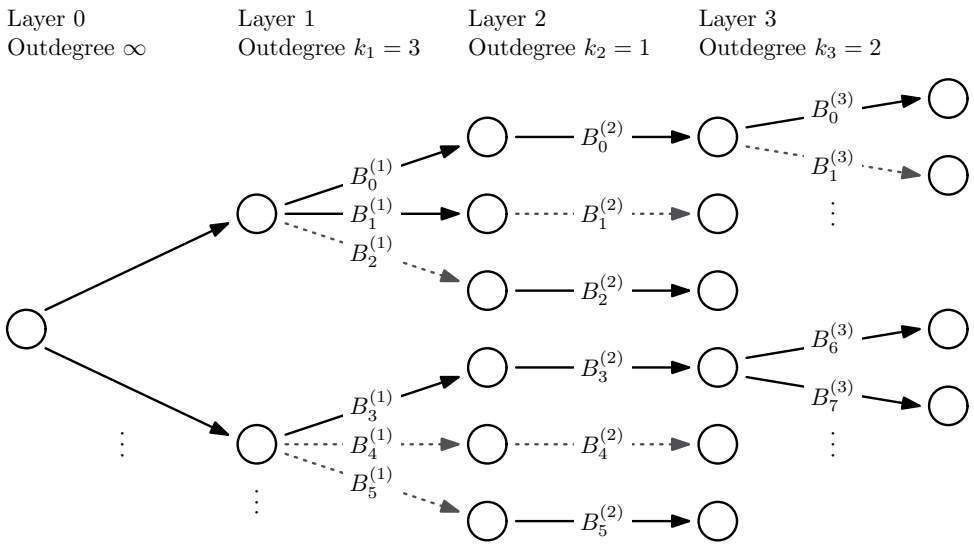

Figure 1 A visualisation of the tree underlying the search of the simplified CONSENSUS algorithm.

the sequence of choices $(\sigma_0, \dots, \sigma_n) \in \mathbb{N}_0 \times \{0, \dots, k_1 - 1\} \times \dots \times \{0, \dots, k_n - 1\}$ made in each layer. The returned sequence can be encoded as a bitstring M. Then $S_i \coloneqq \sigma_0 \circ \dots \circ \sigma_i$ is a successful seed for each $i \in [n]$ by construction. The merits of the approach are summarised in the following theorem.

▶ **Theorem 6** (Performance of Simplified CONSENSUS). *Given $n \in \mathbb{N}$, $p_1, \dots, p_n \in (0, 1]$ and $\varepsilon \in (0, 1]$ let $k_1, \dots, k_n \in \mathbb{N}$ be a sequence of numbers satisfying*[5] $\prod_{j=1}^{i} p_j k_j 2^{-\varepsilon} \in [1, 2]$ *for all $i \in [n]$. Then Algorithm 1 solves the SSEP with $\mathbb{E}[T_i] = \mathcal{O}(T_i^{\mathrm{OPT}}/\varepsilon)$ and $\mathbb{E}[|M|] \leq M^{\mathrm{OPT}} + \varepsilon n + \log_2 1/\varepsilon + \mathcal{O}(1)$.*

Full CONSENSUS Algorithm. The main issues with the simplified CONSENSUS algorithm are that $(\sigma_0, \dots, \sigma_n)$ is awkward to encode in binary and that using $\sigma_0 \circ \dots \circ \sigma_i$ directly as the seed S_i makes for seeds of size $\Omega(n)$ bits, which are inefficient to compute and handle. We solve these issues with three interventions.

1. We introduce a word size $w \in \mathbb{N}$ and encode σ_0 using w bits (hoping that it does not overflow, see below).
2. We demand that $k_1, \dots, k_n \in \mathbb{N}$ are powers of two, i.e. $k_i = 2^{\ell_i}$ for $\ell_i \in \mathbb{N}_0$. With our definition of "$\circ$", the binary representation of the number $\sigma_0 \circ \dots \circ \sigma_i$ now simply arises by concatenating the binary representations of $\sigma_0, \dots, \sigma_i$ (where σ_j is encoded using ℓ_j bits and σ_0 is encoded using w bits).
3. Instead of defining S_i to be $\sigma_0 \circ \dots \circ \sigma_i$, we define S_i to be the w-bit suffix of $\sigma_0 \circ \dots \circ \sigma_i$. We will show that this truncation is unlikely to cause collisions, i.e. any two sequences $(\sigma_0, \dots, \sigma_i)$ and $(\sigma'_0, \dots, \sigma'_i)$ we encounter have distinct suffixes $S_i \neq S'_i$. Intuitively, this guarantees that the search algorithm will always see "fresh" random variables and does not "notice" the change.

The full algorithm is given as Algorithm 2, using notation defined in Figure 2. Our formal result is as follows.

[5] Ideally we would want $k_j \coloneqq \frac{2^{\varepsilon}}{p_j}$ but the integer constraint on the k_j forces us to be more lenient.

Algorithm 1 Simplified CONSENSUS.

Input: $n, (p_1, \dots, p_n), (k_1, \dots, k_n)$ and $B_0^{(i)}, B_1^{(i)}, \dots \sim \mathrm{Ber}(p_i)$ for all $i \in [n]$.
Output: $(\sigma_0, \dots, \sigma_n)$ with $0 \le \sigma_i < k_i$ such that $B^{(i)}_{\sigma_0 \circ \sigma_1 \circ \dots \circ \sigma_i} = 1$ for all $i \in [n]$.
Algorithm Simplified-CONSENSUS:

```
for σ_0 ∈ {0, 1, 2, ...} do
    (σ_1, i) ← (0, 1)
    while i > 0 do
        if B^(i)_{σ_0 ∘ ... ∘ σ_i} = 1 then
            if i = n then
                return M = (σ_0, ..., σ_n)
            i ← i + 1
            σ_i ← 0
        else // backtrack and/or next seed
            while i > 0 and σ_i = k_i − 1 do
                i ← i − 1
            if i > 0 then
                σ_i ← σ_i + 1
```

Algorithm 2 Full CONSENSUS.

Input: $n, w, (p_1, \dots, p_n), (\ell_1, \dots, \ell_n)$ and $B_0^{(i)}, \dots, B_{2^w-1}^{(i)} \sim \mathrm{Ber}(p_i)$ for all $i \in [n]$.
Output: either $\perp$ or M as in Figure 2 such that $B_{S_i}^{(i)} = 1$ for all $i \in [n]$.
Algorithm Full-CONSENSUS:

```
for σ_0 ∈ {0, 1, ..., 2^w − 1} do // bounded
    (σ_1, i) ← (0, 1)
    while i > 0 do
        if B^(i)_{S_i} = 1 then // S_i as in Figure 2
            if i = n then
                return M = (σ_0, ..., σ_n)
            i ← i + 1
            σ_i ← 0
        else // backtrack and/or next seed
            while i > 0 and σ_i = 2^{ℓ_i} − 1 do
                i ← i − 1
            if i > 0 then
                σ_i ← σ_i + 1
return ⊥ // can fail
```

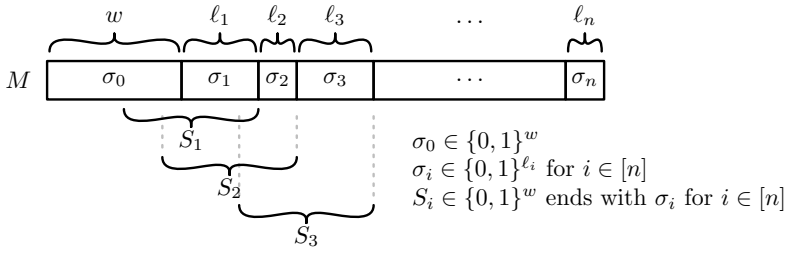

Figure 2 The bitstring M returned by the full CONSENSUS algorithm and the notation used to refer to relevant substrings.

▶ **Theorem 7** (Performance of Full CONSENSUS). *Let $c > 0$ be an arbitrary constant. Given $n \in \mathbb{N}$, $p_1, \dots, p_n \in (n^{-c}, 1]$ and $\varepsilon \in (n^{-c}, 1]$, let $\ell_1, \dots, \ell_n \in \mathbb{N}_0$ be the unique sequence of numbers such that*

$$\sum_{j=1}^{i} \ell_j \coloneqq \left\lceil \varepsilon i + \sum_{j=1}^{i} \log_2(1/p_i) \right\rceil \text{ for } i \in [n].$$

Assume we run Algorithm 2 with these parameters and $w \ge (2 + 7c) \log_2 n$.[6] Then there is an event E with $\Pr[E] = 1 - \mathcal{O}(n^{-c})$ that implies that the run solves the SSEP, producing a bitstring M with $|M| = \lceil M^{\mathrm{OPT}} + \varepsilon n + w \rceil$ containing $S_i \in \{0,1\}^w$ at offset $\sum_{j=1}^{i} \ell_j$ for all $i \in [n]$. Moreover, $\mathbb{E}[T_i \mid E] = \mathcal{O}(T_i^{\mathrm{OPT}}/\varepsilon)$ for all $i \in [n]$.

[6] We did not attempt to improve the constant "$2 + 7c$" and expect that $w = 64$ is enough in practice.

In particular the result guarantees that S_i can be inferred efficiently given M and i provided that $\sum_{j=1}^{i} \log_2(1/p_i)$ can be computed efficiently given i. In practice it is fine to use approximations of p_i.[7] Note that the expectation of T_i is only bounded conditioned on a good event E, which is defined in Section 4. To obtain an unconditionally bounded expectation, Algorithm 2 should be modified to give up early in failure cases.[8]

Special Cases and Variants. An important special case is $p_1 = \ldots = p_n = p$. With $b = \log_2 1/p$ the space lower bound is then simply $M^{\mathrm{OPT}} = bn$ bits. If we apply Theorem 7 with $\varepsilon \in (0,1)$ then the lengths $\ell_1, \ldots, \ell_n$ of the seed fragments $\sigma_1, \ldots, \sigma_n$ are all either $\lceil b+\varepsilon \rceil$ or $\lfloor b+\varepsilon \rfloor$ with an average approaching $b+\varepsilon$. Assuming $b \notin \mathbb{N}$ it may be tempting to use $\varepsilon = \lceil b \rceil - b$ such that all seed fragments have the same length $b+\varepsilon \in \mathbb{N}$. Similarly, we could pick ε such that $b+\varepsilon$ is a "nice" rational number, e.g. $b+\varepsilon = b' + \frac{1}{2}$ for $b' \in \mathbb{N}$ such that $\ell_1, \ldots, \ell_n$ alternate between $b'+1$ and b'.

In some cases it may be reasonable to choose $\varepsilon > 1$. Even though Consensus ceases to be more space efficient than MIN, it still has the advantage that seeds are bitstrings of fixed lengths in predetermined locations. We decided against formally covering this case in Theorem 7, but argue for the following modification in the full version [24] of this paper.

▶ Remark 8. The guarantees of Theorem 7 also hold for $\varepsilon \in [1, \log n]$, except that

$$\mathbb{E}[T_i \mid E] = T_i^{\mathrm{OPT}}(1 + \exp(-\Omega(2^{\varepsilon})) + n^{-c}) \qquad \text{(instead of } \mathbb{E}[T_i \mid E] = \mathcal{O}(T_i^{\mathrm{OPT}}/\varepsilon)\text{)}.$$

It may also be worthwhile to consider variants of Consensus that use different values of the overhead parameter ε in different parts of M to minimise some overall cost. Such an idea makes sense in general if we augment the SSEP to include specific costs c_i associated with testing the Bernoulli random variables belonging to the ith family.

3 Analysis of Simplified Consensus

We begin by reducing the claim of Theorem 6 to a purely mathematical statement given in Lemma 9, which we prove in Section 3.2.

3.1 Reduction of Theorem 6 to Lemma 9

Recall the tree defined in Section 2 and illustrated in Figure 1. For any $i \in [n+1]$ consider a node v in layer i. Let q_i be the probability that there is an unbroken path from v to a node in layer $n+1$. (This probability is indeed the same for all nodes in layer i.) In the last layer, we have $q_{n+1} = 1$ by definition. For all $1 \le i \le n$ we have $q_i = 1 - (1 - p_i q_{i+1})^{k_i}$ since $p_i q_{i+1}$ is the probability that an unbroken path via a specific child of v exists, and since there is an independent chance for each of the k_i children. The technical challenge of this section is to show that the q_i are not too small.

▶ **Lemma 9.** *Under the conditions of Theorem 6 we have $q_i = \Omega(\varepsilon)$ for all $i \in [n]$.*

This lemma implies Theorem 6 as follows.

[7] Our implementation uses fixed point arithmetic (integers counting "microbits") to avoid issues related to floating point arithmetic not being associative and distributive.

[8] To do so we need not decide whether E occurs. We can simply count the number of steps and abort when a limit is reached.

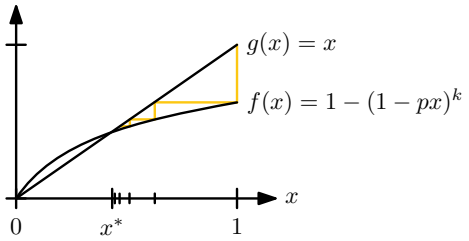

Figure 3 For $p_1 = \ldots = p_n = \frac{1}{4}$ and $k_1 = \ldots = k_n = 5$ (thus $\varepsilon = \log_2(5/4) \approx 0.32$) the values $q_{n+1}, q_n, q_{n-1}, \ldots$ (marked as ticks on the x-axis) arise by iteratively applying $f(x) = 1 - (1 - px)^k$ to the starting value of 1. Each q_i is at least the fixed point $x^* \approx 0.45$ of f.

Proof of Theorem 6. Let T_i for $i \in [n]$ be the number of Bernoulli random variables from the family $(B_j^{(i)})_{j \in \mathbb{N}_0}$ that are inspected. This corresponds to the number of edges between layers i and $i+1$ that are inspected by the DFS. With probability p_i such an edge is unbroken and, if it is unbroken, then with probability q_{i+1} the DFS will never backtrack out of the corresponding subtree and hence never inspect another edge from the same layer. This implies $T_i \sim \text{Geom}(p_i q_{i+1})$ and thus $\mathbb{E}[T_i] = \frac{1}{p_i q_{i+1}} = \mathcal{O}(T_i^{\text{OPT}}/\varepsilon)$ where the last step uses $T_i^{\text{OPT}} = 1/p_i$ and Lemma 9.

Concerning the space to encode $M = (\sigma_0, \ldots, \sigma_n)$, there are two parts to consider. The "fixed width" numbers $(\sigma_1, \ldots, \sigma_n) \in \{0, \ldots, k_1 - 1\} \times \ldots \times \{0, \ldots, k_n - 1\}$ can be encoded using $\lceil \log_2 \prod_{i \in [n]} k_i \rceil$ bits. By the assumption on $(k_i)_{i \in [n]}$ we have $\prod_{i \in [n]} k_i \leq 2 \cdot \prod_{i \in [n]} 2^\varepsilon / p_i$ and can bound the space by

$$1 + \log_2 \prod_{i \in [n]} k_i \leq 1 + \log_2 \left(2 \prod_{i \in [n]} \frac{2^\varepsilon}{p_i} \right) \leq 2 + \varepsilon n + \sum_{i \in [n]} \log_2 \frac{1}{p_i} = M^{\text{OPT}} + \varepsilon n + \mathcal{O}(1).$$

The "variable width" number $1 + \sigma_0 \in \mathbb{N}$ has distribution $1 + \sigma_0 \sim \text{Geom}(q_1)$ because σ_0 is the 0-based index of the first node in layer 1 with an unbroken path to layer $n+1$. Again by Lemma 9 we have $\mathbb{E}[1 + \sigma_0] = \frac{1}{q_1} = \mathcal{O}(1/\varepsilon)$. Using Jensen's inequality [15, 27] and the fact that $\log_2$ is concave, the expected number of bits to encode σ_0 is hence bounded by

$$\mathbb{E}[\lceil \log_2(1 + \sigma_0) \rceil] \leq 1 + \log_2 \mathbb{E}[1 + \sigma_0] = 1 + \log_2(\mathcal{O}(1/\varepsilon)) = \log_2 1/\varepsilon + \mathcal{O}(1).$$

Summing the contributions of both parts yields a bound on $\mathbb{E}[|M|]$ as claimed. ◀

3.2 Proof of Lemma 9

To understand the proof of Lemma 9 it may help to consider a simplified case first. Assume $p_1 = \ldots = p_n = p$ and $k_1 = \ldots = k_n = k$ with $kp = 2^\varepsilon$, i.e. k is slightly larger than $\frac{1}{p}$. We have $q_i = 1 - (1 - pq_{i+1})^k$, which motivates defining $f(x) = 1 - (1 - px)^k$ such that q_i arises by iteratively applying f to an initial value of $q_{n+1} = 1$. From the illustration in Figure 3 we see that q_i can never drop below the largest fixed point x^* of f, and it is not hard to show that $x^* = \Omega(\varepsilon)$.

Our full proof has to deal with a setting that is much less clean. The values p_i and k_i depend on i, yielding a distinct function f_i for each $i \in [n]$ and $p_i k_i \approx 2^\varepsilon$ is only true in the sense of a geometric average over $\Omega(1/\varepsilon)$ consecutive indices. We hence analyse compositions $f_a \circ \ldots \circ f_{b-1}$ for sufficiently large $b - a = \Theta(1/\varepsilon)$. We begin with some definitions.

- Let $f_i(x) = 1 - (1 - p_i x)^{k_i}$. Note that this gives $q_i = (f_i \circ \ldots \circ f_n)(1)$ for $i \in [n+1]$.

- For any set $I = \{a, \dots, b-1\}$ let $P_I = \prod_{i \in I} p_i$, $K_I = \prod_{i \in I} k_i$, and $F_I = f_a \circ \dots \circ f_{b-1}$. Note that the assumption of Theorem 6 gives

$$P_I K_I \cdot 2^{-\varepsilon|I|} = \underbrace{P_{\{1,\dots,b-1\}} K_{\{1,\dots,b-1\}} 2^{-\varepsilon(b-1)}}_{\in[1,2]} \Big(\underbrace{P_{\{1,\dots,a-1\}} K_{\{1,\dots,a-1\}} 2^{-\varepsilon(a-1)}}_{\in[1,2]} \Big)^{-1} \in [\tfrac{1}{2}, 2].$$

- Let $\varepsilon_0 := \lceil 2\varepsilon^{-1} \rceil^{-1}$. Its role is to be a number satisfying $\varepsilon_0 = \Theta(\varepsilon)$, $\varepsilon_0^{-1} \in \mathbb{N}$, and $(2^\varepsilon)^{\varepsilon_0^{-1}} \in [4, 8]$. In particular, if $I = \{a, \dots, b-1\}$ with $b - a = \varepsilon_0^{-1}$ then

$$P_I K_I = \underbrace{P_I K_I 2^{-\varepsilon|I|}}_{\in[\frac{1}{2},2]} \cdot \underbrace{2^{\varepsilon|I|}}_{\in[4,8]} \in [2, 16].$$

- Lastly, let $C = 8$ and $\varepsilon_1 = \varepsilon_0 / C$.

These definitions are designed to make the proof of the following lemma as simple as possible. No attempt was made to optimise constant factors.

▶ **Lemma 10.** *Let $I = \{a, \dots, b-1\} \subseteq [n]$. The following holds.*

(i) *For all $i \in [n]$ and $x \in [0, 1]$: $f_i(x) \geq p_i k_i x - (p_i k_i x)^2/2$.*

(ii) *If $b - a \leq \varepsilon_0^{-1}$ then $F_I(x) \geq P_I K_I x(1 - C|I|x)$ for $x \in [0, \varepsilon_1]$.*

(iii) *If $b - a = \varepsilon_0^{-1}$ then $F_I(\varepsilon_1/2) \geq \varepsilon_1/2$.*

Proof.

(i) e^{-x} satisfies the following bounds: $1 - x \leq e^{-x} \leq 1 - x + x^2/2$ for $x > 0$. We get the left claim by integrating $e^{-x} \leq 1$ for $t \in [0, x]$ to get $1 - e^{-x} \leq x$. Integrating again for $t \in [0, x]$ gives the right claim. Applying to $f_i(x)$ we get

$$f_i(x) = 1 - (1 - p_i x)^{k_i} \geq 1 - e^{-p_i k_i x} \geq 1 - (1 - p_i k_i x + (p_i k_i x)^2/2) = p_i k_i x - (p_i k_i x)^2/2.$$

(ii) We use induction on $b - a$. For $b - a = 0$ we have $I = \emptyset$, $F_I = \mathrm{id}$ as well as $P_I = K_I = 1$ and the claim holds. Let now $I = \{a, \dots, b-1\}$ with $b - a \leq \varepsilon_0^{-1}$and $I' = \{a + 1, \dots, b - 1\}$. We assume the claim holds for I'. Let $x \in [0, \varepsilon_1]$.

$$\begin{aligned}
F_I(x) = (f_a \circ F_{I'})(x) &\overset{\text{monotonic}}{=} f_a(F_{I'}(x)) \overset{\text{induction}}{\geq} f_a(P_{I'} K_{I'} x(1 - C|I'|x)) \\
&\overset{\text{(i)}}{\geq} p_a k_a P_{I'} K_{I'} x(1 - C|I'|x) - (p_a k_a P_{I'} K_{I'} x(1 - C|I'|x))^2/2 \\
&= P_I K_I x(1 - C|I'|x) - (P_I K_I x(1 - \underbrace{C|I'|x}_{\leq C\varepsilon_0^{-1}\varepsilon_1 = 1 < 2}))^2/2 \quad \text{and hence } (1 - C|I'|x)^2 \leq 1 \\
&\geq P_I K_I x(1 - C|I'|x) - (P_I K_I x)^2/2 = P_I K_I x(1 - x(C|I'| + \underbrace{P_I K_I}_{\leq 16 = 2C}/2)) \\
&\geq P_I K_I x(1 - x(C|I'| + C)) = P_I K_I x(1 - C|I|x).
\end{aligned}$$

(iii) By applying (ii) and using $C\varepsilon_0^{-1}\varepsilon_1 = 1$ we get

$$F_I(\varepsilon_1/2) \overset{\text{(ii)}}{\geq} \underbrace{P_I K_I}_{\geq 2} \tfrac{\varepsilon_1}{2} \underbrace{(1 - C\varepsilon_0^{-1} \tfrac{\varepsilon_1}{2})}_{= \frac{1}{2}} \geq \varepsilon_1/2.$$ ◀

The proof of Lemma 9 is now immediate:

Proof of Lemma 9. Let $i \in [n]$. Partition $I = \{i, \dots, n\}$ into contiguous intervals $I_1, \dots, I_k$ such that $F_I = F_{I_1} \circ \dots \circ F_{I_k}$ with $|I_1| \leq \varepsilon_0^{-1}$ and $|I_j| = \varepsilon_0^{-1}$ for $j \in \{2, \dots, k\}$. Then by repeated applications of Lemma 10 (iii), one application of Lemma 10 (ii) and the monotonicity of the relevant functions we get

$$q_i = F_I(1) \geq F_I(\varepsilon_1/2) = F_{I_1}(F_{I_2}(\dots(F_{I_k}(\varepsilon_1/2))\dots)) \overset{\text{(iii)}}{\geq} F_{I_1}(\varepsilon_1/2)$$
$$\overset{\text{(ii)}}{\geq} P_{I_1} K_{I_1} \frac{\varepsilon_1}{2}(1 - C|I_1|\frac{\varepsilon_1}{2}) = \underbrace{P_{I_1} K_{I_1} 2^{-\varepsilon|I_1|}}_{\geq \frac{1}{2}} \underbrace{2^{\varepsilon|I_1|}}_{\geq 1} \frac{\varepsilon_1}{2}(1 - \underbrace{C|I_1|\frac{\varepsilon_1}{2}}_{\leq \frac{1}{2}}) \geq \frac{\varepsilon_1}{8}. \quad \blacktriangleleft$$

4 Analysis of Full Consensus

Full Consensus performs a DFS in a tree as depicted in Figure 1 in much the same way as simplified Consensus, with two differences. First, the root node only has 2^w children (not infinitely many) and the probability that an unbroken path exists is therefore strictly less than 1. Secondly, when examining the outgoing edge of index σ_i of some node in layer i, then its status is no longer linked to B^i_Σ for $\Sigma = \sigma_0 \circ \dots \circ \sigma_i$. Instead Σ is divided as $\Sigma = \text{prefix}(\Sigma) \circ \text{suffix}(\Sigma)$ where $\text{suffix}(\Sigma) \in \{0,1\}^w$ and $B^i_{\text{suffix}(\Sigma)}$ is used. To prepare for the proof of Theorem 7, we present a coupling to, and a claim about simplified Consensus.

A Coupling between Simplified and Full Consensus. Consider the following natural coupling. Let $B^{(i)}_j$ for $i \in [n]$ and $j \in \mathbb{N}_0$ be the random variables underlying simplified Consensus and $\tilde{B}^{(i)}_S$ for $i \in [n]$ and $S \in \{0,1\}^w$ the random variables underlying full Consensus. The coupling should ensure that $B^{(i)}_\Sigma = \tilde{B}^{(i)}_{\text{suffix}(\Sigma)}$ for all pairs (i, Σ) such that simplified Consensus inspects $B^{(i)}_\Sigma$ and does not at an earlier time inspect $B^{(i)}_{\Sigma'}$ with $\text{suffix}(\Sigma') = \text{suffix}(\Sigma)$. In other words, for any pair (i, S), if simplified consensus inspects at least one variable $B^{(i)}_\Sigma$ with $\text{suffix}(\Sigma) = S$ then $\tilde{B}^{(i)}_S$ equals the earliest such variable that is inspected.

This implies that the runs of simplified and full Consensus behave identically as long as, firstly, σ_0 never reaches 2^w and, secondly, simplified Consensus never *repeats a suffix*, by which we mean inspecting $B^{(i)}_\Sigma$ for some pair (i, Σ) such that $B^{(i)}_{\Sigma'}$ with $\text{suffix}(\Sigma) = \text{suffix}(\Sigma')$ was previously inspected. In the following, by a *step* of Consensus we mean one iteration of its main while-loop, which involves inspecting exactly one random variable.

▷ Claim 11. Simplified Consensus repeats a suffix in step t with probability at most $\frac{t}{2^w}$.

Proof. Assume simplified Consensus is about to inspect $B^{(i)}_{\Sigma^*}$. Call $S \in \{0,1\}^w$ a *blocked suffix* if there exists $\Sigma \in \{0,1\}^*$ such that $S = \text{suffix}(\Sigma)$ and $B^{(i)}_\Sigma$ was previously inspected. This necessitates $\text{prefix}(\Sigma) < \text{prefix}(\Sigma^*)$. Clearly, step t repeats a suffix if $\text{suffix}(\Sigma^*)$ is blocked. A crucial observation is that the DFS has exhaustively explored all branches of the tree associated with any prefix less than Σ^* and the order in which this exploration has been carried out is no longer relevant. The symmetry between all nodes of the same layer therefore implies that every suffix is blocked with the same probability. Since at most t suffixes can be blocked at step t, the probability that Σ^* is blocked (given our knowledge of (i, Σ^*)) is at most $\frac{t}{2^w}$. ◁

Proof of Theorem 7. First note that the choice of $(\ell_1, \dots, \ell_n)$ made in Theorem 7 amounts to a valid choice of $(k_1 = 2^{\ell_1}, \dots, k_n = 2^{\ell_n})$ for applying Theorem 6. Indeed, for each $i \in [n]$ there exists some $\text{err}_i \in [1, 2)$ related to rounding up such that

$$\prod_{j=1}^{i} p_j k_j 2^{-\varepsilon} = \Big(\prod_{j=1}^{i} p_j\Big) 2^{\sum_{j=1}^{i} \ell_j} 2^{-\varepsilon i} = \Big(\prod_{j=1}^{i} p_j\Big) 2^{\lceil \varepsilon i + \sum_{j=1}^{i} \log_2(1/p_i)\rceil} 2^{-\varepsilon i}$$
$$= \Big(\prod_{j=1}^{i} p_j\Big) 2^{\varepsilon i + \sum_{j=1}^{i} \log_2(1/p_i)} \cdot \mathrm{err}_i 2^{-\varepsilon i} = \mathrm{err}_i \in [1,2]$$

as required. The number $T = \sum_{i=1}^{n} T_i$ of steps taken by simplified Consensus satisfies $\mathbb{E}[T] = \mathcal{O}(\sum_{i=1}^{n} \frac{1}{p_i \varepsilon}) = \mathcal{O}(n^{1+2c})$. By Markov's inequality T exceeds its expectation by a factor of n^c with probability at most n^{-c} so with probability at least $1 - \mathcal{O}(n^{-c})$ simplified Consensus terminates within $T_{\text{limit}} = n^{1+3c}$ steps.

By Claim 11 and a union bound the probability that simplified Consensus repeats a suffix within its first T_{limit} steps is at most $\sum_{t=1}^{T_{\text{limit}}} t/2^w = \mathcal{O}(T_{\text{limit}}^2/2^w) = \mathcal{O}(n^{2+6c}/2^w)$. By choosing $w \geq (2+7c)\log_2 n$ this probability is $\mathcal{O}(n^{-c})$. This choice also guarantees that σ_0 never exceeds 2^w during the first T_{limit} steps. Let now $E = \{T \leq T_{\text{limit}}\} \cap \{\text{no repeated suffix in the first } T_{\text{limit}} \text{ steps}\}$. By a union bound $\Pr[E] = 1 - \mathcal{O}(n^{-c})$ and by our coupling E implies that simplified and full Consensus explore the same sequence of nodes in their DFS and terminate with the same sequence $\sigma_0, \ldots, \sigma_n$ with $\sigma_0 < 2^w$. In particular full Consensus succeeds in that case as claimed.

The updated bound on $|M|$ simply reflects that σ_0, which in simplified Consensus had expected length $\log_2 1/\varepsilon + \mathcal{O}(1)$ now has fixed length $w = \mathcal{O}(\log n)$. Lastly, if T_i and $\tilde{T}_i$ denote the number of Bernoulli trials from the i-th family inspected by simplified and full Consensus, respectively, then conditioned on E we have $T_i = \tilde{T}_i$ so

$$\mathbb{E}[\tilde{T}_i \mid E] = \mathbb{E}[T_i|E] \leq \mathbb{E}[T_i]/\Pr[E] = \mathcal{O}(\mathbb{E}[T_i]) = \mathcal{O}(T_i^{\text{OPT}}/\varepsilon). \quad \blacktriangleleft$$

5 Application to Minimal Perfect Hashing

In this section, we apply the Consensus idea to the area of minimal perfect hash functions (MPHFs). Recall from Section 1.3 that an MPHF for a given set X of n keys maps each key in X to a unique integer from $[n]$.

5.1 Existing Strategies for Constructing MPHFs

We begin with a brief overview of established approaches with a focus on those we need. Detailed explanations are found in [18, 19].

Partitioning. Most approaches randomly partition the input into small *buckets* using a hash function and then construct an MPHF on each bucket separately. The natural way to obtain an MPHF on the entire input involves storing the bucket-MPHFs and prefix sums for the bucket sizes. Storing the prefix sums can be avoided when the partitioning uses a k-perfect hash function, as is done in ShockHash-Flat [21], and as we do in Section 5.3.

Hagerup and Tholey [12] demonstrate that partitioning into tiny buckets of size $\mathcal{O}(\frac{\log n}{\log\log n})$ when suitably combined with exhaustive tabulation of MPHFs on tiny inputs yields an approach with construction time $\mathcal{O}(n)$, query time $\mathcal{O}(1)$ and space overhead $\varepsilon = \Theta(\frac{\log^2 \log n}{\log n})$. Unfortunately the analysis requires astronomical values of $n \geq \max(\exp(\omega(1/\varepsilon)), 2^{150})$ [4] and is restricted to these rather large ε.

Below we explain *monolithic* variants of MPHFs, i.e. constructions not using partitioning, which are typically combined with partitioning to obtain faster *bucketed* variants.

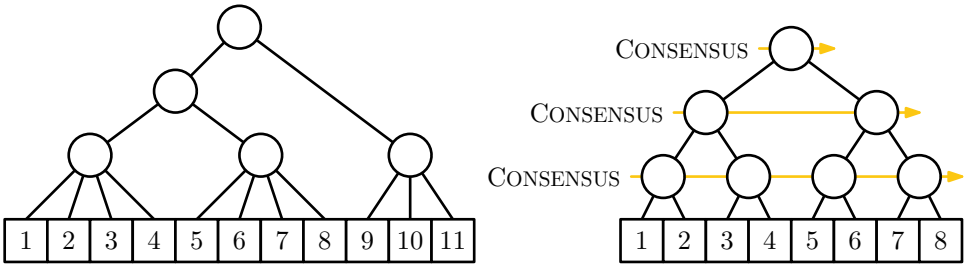

Figure 4 A general splitting tree and a balanced binary splitting tree as used in Section 5.2.

Recursive Splitting. A recursive splitting strategy [7] involves a predefined *splitting tree*, which is a rooted tree with n leaves.[9] See Figure 4 (left) for an example with $n = 11$. An MPHF for an input set of size n is given by a family of functions containing for each internal node v of the tree a *splitting hash function* f_v that maps keys to the children of v. Together, the family $(f_v)_v$ must describe a recursive partitioning of the input set into parts of size 1. The MPHF is queried for a key x by starting at the root. The splitting hash functions are evaluated on x and the resulting path is followed until a leaf is reached, the index of which is returned. Each f_v is identified by a seed that is found using trial and error. In the original paper [7] seeds are stored using Golomb-Rice coding [11, 31]. To keep the space overhead low, the last layer of the splitting tree uses ℓ-way splits for relatively large ℓ (5–16 in practice), which makes splitting hash functions costly to find.

Multiple-Choice Hashing. We randomly assign a set $\{h_1(x), \ldots, h_k(x)\} \subseteq [n(1+\varepsilon)]$ of candidates to each key hoping that a choice $\sigma(x) \in \{1, \ldots, k\}$ can be made for each key such that $x \mapsto h_{\sigma(x)}(x)$ is injective. A perfect hash function is then given by a retrieval data structure that stores σ [4, 10, 20]. A recent variant ShockHash [22, 21] achieves record space efficiency by using the aggressive configuration $\varepsilon = 0$ and $k = 2$ (combined with partitioning and recursive splitting). Many retries are needed until σ exists.

Bucket Placement. A general strategy with many variants assigns a random bucket $f(x) \in \{1, \ldots, b\}$ to each key and considers for each bucket $i \in [b]$ several (hash) functions $g_{i,1}, g_{i,2}, \ldots$ with range $[n]$. A choice $\sigma(i)$ is stored for each bucket such that $x \mapsto g_{f(x),\sigma(f(x))}(x)$ is a bijection. Typically, the values $\sigma(i)$ are chosen greedily in decreasing order of bucket sizes [30, 14, 1], but backtracking has also been considered [33, 8] (in a different sense than Consensus).

5.2 Monolithic Consensus-RecSplit

We now describe a variant of recursive splitting in an idealised setting with $n = 2^d$ that uses Consensus to be particularly space efficient. The idea is simple. The data structure consists of $n - 1$ two-way splitting hash functions, i.e. hash functions with range $\{0, 1\}$, arranged in a balanced binary tree with n leaves, see Figure 4 (right).

This means that during construction for a key set S we have to select the splitting hash functions from top to bottom such that each function splits the part of S arriving at its node perfectly in half. The data structure is entirely given by a sequence of $n - 1$ seeds that identify the splitting hash functions. If we search and encode these seeds using the Consensus algorithm, we obtain the following result.

[9] In the original paper [7], the lowest level is suppressed, with correspondingly changed terminology.

▶ **Theorem 12** (Monolithic CONSENSUS-RecSplit). *Let $n = 2^d$ for $d \in \mathbb{N}$ and $\varepsilon \in [\frac{1}{n}, 1]$. Monolithic-RecSplit is an MPHF data structure with space requirement* $\mathrm{OPT}_{\mathrm{MPHF}} + \mathcal{O}(\varepsilon n + \log^2 n)$*, expected construction time* $\mathcal{O}(\max(n^{3/2}, \frac{n}{\varepsilon}))$ *and query time* $\mathcal{O}(\log n)$*.*

Proof. We use a separate CONSENSUS data structure per level of the splitting tree and construct them from top to bottom. Consider level $\ell \in \{0, \dots, d-1\}$ of the tree, which contains 2^ℓ nodes responsible for $n_\ell = 2^{d-\ell}$ keys each. The probability that a given seed for a splitting hash function amounts to a perfect split for n_ℓ keys is $p(n_\ell) = \binom{n_\ell}{n_\ell/2} \cdot 2^{-n_\ell}$ by a simple counting argument. A standard approximation gives $p(n_\ell) = \Theta(1/\sqrt{n_\ell})$.

By applying Theorem 7 with $p_1 = \ldots = p_{2^\ell} = p(n_\ell)$, $\varepsilon_\ell = \min(1, \varepsilon \cdot n_\ell^{3/4})$ and $w = \mathcal{O}(\log n)$ we obtain a data structure encoding 2^ℓ successful seeds. The expected number of seeds inspected is $\mathcal{O}(\frac{1}{p(n_\ell)\varepsilon_\ell})$ for each node, which corresponds to expected work of $\mathcal{O}(\frac{n_\ell}{p(n_\ell)\varepsilon_\ell}) = \mathcal{O}(n_\ell^{3/2}/\varepsilon_\ell)$ per node when seeds are tested by hashing all n_ℓ relevant keys.[10] For the total expected work w_ℓ in level ℓ there are, up to constant factors, the following two options.

$$\text{if } \varepsilon_\ell = \varepsilon \cdot n_\ell^{3/4} \text{ then } w_\ell = \frac{n_\ell^{3/2}}{\varepsilon n_\ell^{3/4}} \cdot 2^\ell = \frac{n_\ell^{3/4} 2^\ell}{\varepsilon} = \frac{2^{(d-\ell)\cdot 3/4 + \ell}}{\varepsilon} = \frac{n^{3/4} \cdot 2^{\ell/4}}{\varepsilon}.$$

$$\text{if } \varepsilon_\ell = 1 \text{ then } w_\ell = n_\ell^{3/2} \cdot 2^\ell = 2^{(d-\ell)\cdot 3/2 + \ell} = n^{3/2} \cdot 2^{-\ell/2}.$$

The formula for the first case is maximised for $\ell = d-1$ giving $w_{d-1} = \Theta(n/\varepsilon)$. The formula for the second case is maximised for $\ell = 0$ giving $w_0 = n^{3/2}$. Since both describe geometric series we get $\sum_{\ell=0}^{d-1} w_\ell = \mathcal{O}(w_0 + w_{d-1}) = \mathcal{O}(n^{3/2} + n/\varepsilon) = \mathcal{O}(\max(n^{3/2}, n/\varepsilon))$ as desired.

The space requirement for level ℓ is $|M_\ell| = 2^\ell \cdot \log_2(1/p(n_\ell)) + \varepsilon_\ell 2^\ell + \mathcal{O}(\log n)$. To bound the sum $\sum_{\ell=0}^{d-1} |M_\ell|$ we look at the terms in $|M_\ell|$ separately. For the first, we will use that $\prod_{\ell=0}^{d-1} p(2^{d-\ell})^{2^\ell} = \frac{n!}{n^n}$, which can be checked by induction: $d = 0$ is clear. For $d > 0$ we have

$$\prod_{\ell=0}^{d-1} p(2^{d-\ell})^{2^\ell} = p(n) \cdot \prod_{\ell=1}^{d-1} p(2^{d-\ell})^{2^\ell} = p(n) \cdot \Big(\prod_{\ell=1}^{d-1} p(2^{d-\ell})^{2^{\ell-1}} \Big)^2$$

$$= p(n) \cdot \Big(\prod_{\ell=0}^{d-2} p(2^{d-1-\ell})^{2^\ell} \Big)^2 \overset{\text{Ind.}}{=} p(n) \cdot \Big(\frac{(n/2)!}{(n/2)^{n/2}} \Big)^2 = \binom{n}{n/2} 2^{-n} \cdot \Big(\frac{(n/2)!}{(n/2)^{n/2}} \Big)^2 = \frac{n!}{n^n}.$$

The leading terms in $|M_\ell|$ therefore add up to

$$\sum_{\ell=0}^{d-1} 2^\ell \cdot \log_2(1/p(n_\ell)) = \log_2 \Big(1/ \prod_{\ell=0}^{d-1} p(n_\ell)^{2^\ell} \Big) = \log_2 \big(\frac{n^n}{n!} \big) = n \log_2 e - \mathcal{O}(\log n).$$

The overhead terms $\varepsilon_\ell \cdot 2^\ell$ are geometrically increasing in ℓ. This is because

$$\frac{\varepsilon_{\ell+1} \cdot 2^{\ell+1}}{\varepsilon_\ell \cdot 2^\ell} = \frac{\min(1, \varepsilon \cdot n_{\ell+1}^{3/4})}{\min(1, \varepsilon \cdot n_\ell^{3/4})} \cdot 2 = \frac{\min(1, \varepsilon \cdot (2^{d-\ell-1})^{3/4})}{\min(1, \varepsilon \cdot (2^{d-\ell})^{3/4})} \cdot 2 \geq 2^{-3/4} \cdot 2 = 2^{1/4} > 1.$$

The sum of the overhead terms is therefore dominated by the last term $\varepsilon_{d-1} 2^{d-1} \leq \varepsilon n \cdot 2^{-1/4}$ giving $\sum_{\ell=0}^{d-1} \varepsilon_\ell \cdot 2^\ell \leq \varepsilon n / (2^{1/4} - 1) < 6\varepsilon n$. Overall we get as claimed

$$\text{space} = \sum_{\ell=0}^{d-1} |M_\ell| \leq n \log_2 e + 6\varepsilon n + \mathcal{O}(\log^2 n).$$

[10] Dividing the work per node by ε_ℓ gives $-n_\ell^{3/2}/\varepsilon_\ell^2$. Our choice of $\varepsilon_\ell \sim n_\ell^{3/4}$ (as long as $\min(1, \cdot)$ does not kick in) is desired to ensure that using extra bits amounts to roughly the same reduction of work in each level, such that we have a consistent trade-off between time and space.

A query needs to extract one seed from each of the $d = \log_2 n$ CONSENSUS data structures. Since each uses uniform values for $p_1 = \ldots = p_{n_\ell}$, the product $\prod_{j=1}^{i} p_j = p(n_\ell)^i$ is easy to compute in constant time, which is sufficient to find each seed in constant time, so we get a query time of $\mathcal{O}(d)$ in total. ◀

5.3 Bucketed CONSENSUS-RecSplit

We now replace the upper layers of Monolithic CONSENSUS-RecSplit with a minimal k-perfect hash function [13]. This reduces query time as there are fewer layers to traverse and reduces construction time because the splits in the top layers are the most expensive ones to compute. Most importantly, it enables efficiently handling input sizes that are not powers of two.

For any $k \in \mathbb{N}$ a minimal k-perfect hash function for a set S of n keys is a data structure mapping the elements of S to buckets indexed with $[\lceil n/k \rceil]$ such that each bucket $i \in [\lfloor n/k \rfloor]$ receives exactly k keys. If k does not divide n then bucket $\lceil n/k \rceil$ receives between 1 and $k-1$ leftover keys. It has long been known that the space lower bound for such a data structure is $\approx n(\log_2 e - \frac{1}{k}\log_2 \frac{k^k}{k!})$ [1], shown to be $\approx \frac{n}{2k}\log 2\pi k$ in [16]. In the full version of this paper [24] we outline a compact minimal k-perfect hash function with, in expectation, $\mathcal{O}(\frac{n}{k}\log k)$ bits of space, constant query time, and linear construction time.

We call our resulting MPHF Bucketed CONSENSUS-RecSplit. Its performance is as follows (previously stated in Section 1.3).

▶ **Theorem 5** (Bucketed CONSENSUS-RecSplit). *For any $\varepsilon \in [n^{-3/7}, 1]$ there is an MPHF data structure with space requirement $(1 + \mathcal{O}(\varepsilon))\mathrm{OPT}_{\mathrm{MPHF}}$, expected construction time $\mathcal{O}(n/\varepsilon)$ and expected query time $\mathcal{O}(\log 1/\varepsilon)$.*

Proof. The idea is to use our monolithic construction but replace the top-most layers with a k-perfect hash function. More precisely, we choose $k = 2^{\lfloor \frac{4}{3}\log_2 1/\varepsilon \rfloor} = \Theta(\varepsilon^{-4/3})$ and construct a minimal k-perfect hash function F for the input set. As discussed in the full version, the construction time of F is $\mathcal{O}(n)$, the expected query time is $\mathcal{O}(1)$ and the space consumption is $\mathcal{O}(\frac{n}{k}\log k) = \mathcal{O}(\varepsilon n)$ bits, all well within the respective budgets. The resulting $\lfloor n/k \rfloor$ buckets of k keys each (a power of 2) are recursively split like in our monolithic data structure, which takes at most $\mathrm{OPT}_{\mathrm{MPHF}} + \mathcal{O}(\varepsilon n)$ bits of space (the $\mathcal{O}(\log^2 n)$ term is dominated by $\mathcal{O}(\varepsilon n)$). Since we make no assumptions on n there might be up to $k-1$ leftover keys assigned to an extra bucket. For these we construct a separate MPHF F' with range $\{\lfloor n/k \rfloor \cdot k + 1, \ldots, n\}$. Since $k \sim \varepsilon^{-4/3} = \varepsilon \cdot \varepsilon^{-7/3} \leq \varepsilon n$ pretty much any compact (not necessarily succinct) MPHF is good enough here, provided it supports queries in logarithmic time. Queries evaluate F first and, if mapped to a regular bucket, evaluate a sequence of $\log_2 k$ splits, or, if mapped to the extra bucket, evaluate F'. This takes $\mathcal{O}(\log k) = \mathcal{O}(\log 1/\varepsilon)$ time in the worst case.

Concerning construction time, note that the top-most level in use has nodes of size $n_\ell = k \leq \varepsilon^{-4/3}$ and the definition of ε_ℓ we made simplifies to $\varepsilon_\ell = n_\ell^{3/4} \cdot \varepsilon$ without the "$\min(1, \cdot)$". By an argument in Theorem 12, the construction time is then dominated by the bottom layer and bounded by $\mathcal{O}(n/\varepsilon)$ as claimed. ◀

5.4 Experiments

To demonstrate that CONSENSUS-RecSplit is viable in practice, we give an implementation and compare it to other MPHF constructions from the literature.[11] We only include the most space-efficient previous approaches with space consumption below 1.6 bits per key and

[11] We run single-threaded experiments on an Intel i7 11700 processor with a base clock speed of 2.5 GHz. The sizes of the L1 and L2 data caches are 48 KiB and 512 KiB per core, and the L3 cache has a size of

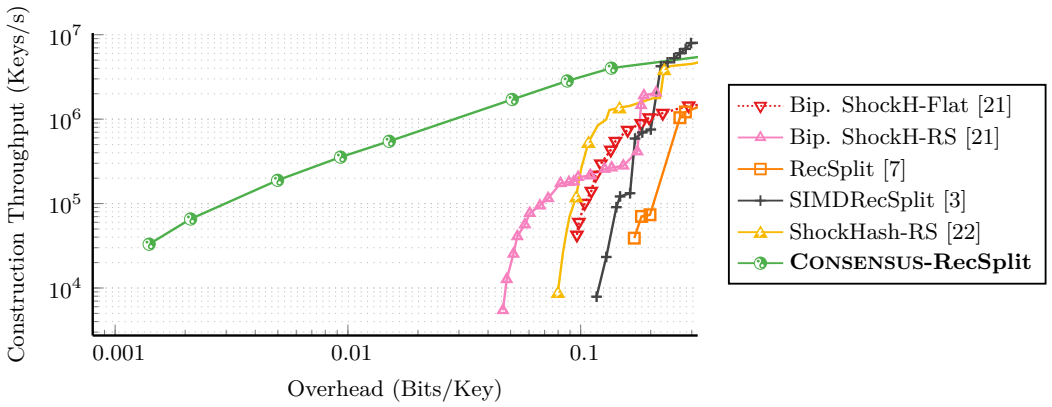

Figure 5 Experimental evaluation of very space-efficient MPHF algorithms, showing the trade-off between space consumption and construction throughput. Because CONSENSUS-RecSplit is focused on space consumption, we only include approaches that achieve below 1.6 bits per key.

refer to [18, 19] for a detailed comparison of state-of-the-art perfect hashing approaches with larger space consumption. Our implementation departs from the algorithmic description in Section 5.3 by using the threshold-based k-perfect hash function [21, 13]. We postpone further tuning and parallelisation efforts to a future paper.

Figure 5 shows the trade-off between space consumption and construction time of CONSENSUS-RecSplit. In the full version [24] we also give a table. The performance of all previous approaches degrades abruptly for smaller overheads because of their exponential dependency on $1/\varepsilon$ while CONSENSUS-RecSplit shows an approximately linear dependence on $1/\varepsilon$ as predicted in Theorem 5. The space lower bound for MPHFs is $\log_2 e \approx 1.443$ bits per key. The previously most space-efficient approach, bipartite ShockHash-RS [21], achieves 1.489 bits per key. In about 15% of the construction time, CONSENSUS-RecSplit achieves a space consumption of just 1.444 bits per key. At a space consumption of 1.489 bits per key, CONSENSUS-RecSplit is more than 350 times faster to construct than the previous state of the art.

Queries take 150–250 ns depending on k and therefore the depth of the splitting tree (see full version), so their performance is not too far from compact configurations of RecSplit (100 ns) and bipartite ShockHash-RS (125 ns). Future work can improve on the query performance by storing the seeds needed for each query closer to each other. We can adapt bucketed CONSENSUS-RecSplit by constructing one CONSENSUS data structure storing seeds bucket by bucket instead of an independent data structure for each layer. Preliminary experiments show up to 30% faster queries with moderately slower construction at the same space consumption. Splittings higher up in the tree not only have a smaller success probability, but are also more costly to test. We are therefore incentivised to select a different space overhead parameter ε for each splitting within the same CONSENSUS data structure. We leave this generalisation of CONSENSUS to non-uniform ε for future work.

16 MiB. The page size is 4 KiB. We use the GNU C++ compiler version 11.2.0 with optimization flags `-O3 -march=native`. Like previous papers [19, 21, 22, 14, 3], we use 100 million random string keys of uniform random length [10..50] as input. The source code is available on GitHub [17, 23].

6 Conclusion

This paper concerns data structures that store sequences of successful seeds, i.e. values known to influence pseudo-random processes in desirable ways. Normally these seeds are first found independently of each other using trial and error and then encoded separately. In this paper we show that, perhaps surprisingly, this wastes $\Omega(1)$ bits per seed compared to a lower bound.

We present the CONSENSUS approach, which **co**mbi**n**es **s**earch and **en**coding of **su**ccessful **s**eeds and reduces the space overhead to $\mathcal{O}(\varepsilon)$ bits at the price of increasing the required work by a factor of $\mathcal{O}(1/\varepsilon)$.

To demonstrate the merits of our ideas in practice, we apply it to the construction of minimal perfect hash functions. CONSENSUS-RecSplit is the first MPHF to achieve a construction time that only depends linearly on the inverse space overhead. Despite using the same recursive splitting idea as [7], the space overhead is, for the same construction throughput, up to two orders of magnitude smaller in practice.

Future Work. Given our promising results, we intend to continue working on CONSENSUS-RecSplit, in particular looking into parallelisation and improving query times. We also believe that CONSENSUS can be applied to other randomised data structures, including but not limited to other perfect or k-perfect hash functions, retrieval data structures and AMQ-filters. We give a first result on k-perfect hash functions using CONSENSUS in [13].

References

1 Djamal Belazzougui, Fabiano C. Botelho, and Martin Dietzfelbinger. Hash, displace, and compress. In *ESA*, volume 5757 of *Lecture Notes in Computer Science*, pages 682–693. Springer, 2009. doi:10.1007/978-3-642-04128-0_61.

2 Djamal Belazzougui and Rossano Venturini. Compressed static functions with applications. In *SODA*, pages 229–240. SIAM, 2013. doi:10.1137/1.9781611973105.17.

3 Dominik Bez, Florian Kurpicz, Hans-Peter Lehmann, and Peter Sanders. High performance construction of RecSplit based minimal perfect hash functions. In *ESA*, volume 274 of *LIPIcs*, pages 19:1–19:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.19.

4 Fabiano C. Botelho, Rasmus Pagh, and Nivio Ziviani. Practical perfect hashing in nearly optimal space. *Inf. Syst.*, 38(1):108–131, 2013. doi:10.1016/J.IS.2012.06.002.

5 Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *ICALP*, volume 443 of *Lecture Notes in Computer Science*, pages 6–19. Springer, 1990. doi:10.1007/BFB0032018.

6 Martin Dietzfelbinger and Michael Rink. Applications of a splitting trick. In *ICALP (1)*, volume 5555 of *Lecture Notes in Computer Science*, pages 354–365. Springer, 2009. doi:10.1007/978-3-642-02927-1_30.

7 Emmanuel Esposito, Thomas Mueller Graf, and Sebastiano Vigna. RecSplit: Minimal perfect hashing via recursive splitting. In *ALENEX*, pages 175–185. SIAM, 2020. doi:10.1137/1.9781611976007.14.

8 Edward A. Fox, Lenwood S. Heath, Qi Fan Chen, and Amjad M. Daoud. Practical minimal perfect hash functions for large databases. *Commun. ACM*, 35(1):105–121, 1992. doi:10.1145/129617.129623.

9 Michael L. Fredman and J'anos Koml'os. On the size of separating systems and families of perfect hash functions. *SIAM J. Algebraic Discrete Methods*, 5(1):61–68, 1984. doi:10.1137/0605009.

10 Marco Genuzio, Giuseppe Ottaviano, and Sebastiano Vigna. Fast scalable construction of (minimal perfect hash) functions. In *SEA*, volume 9685 of *Lecture Notes in Computer Science*, pages 339–352. Springer, 2016. doi:10.1007/978-3-319-38851-9_23.

11 Solomon W. Golomb. Run-length encodings. *IEEE Trans. Inf. Theory*, 12(3):399–401, 1966. doi:10.1109/TIT.1966.1053907.

12 Torben Hagerup and Torsten Tholey. Efficient minimal perfect hashing in nearly minimal space. In *STACS*, volume 2010 of *Lecture Notes in Computer Science*, pages 317–326. Springer, 2001. doi:10.1007/3-540-44693-1_28.

13 Stefan Hermann, Sebastian Kirmayer, Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. Engineering minimal k-perfect hash functions. In *ESA*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025.

14 Stefan Hermann, Hans-Peter Lehmann, Giulio Ermanno Pibiri, Peter Sanders, and Stefan Walzer. PHOBIC: perfect hashing with optimized bucket sizes and interleaved coding. In *ESA*, volume 308 of *LIPIcs*, pages 69:1–69:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.69.

15 Johan Ludwig William Valdemar Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta mathematica*, 30(1):175–193, 1906. doi:10.1007/BF02418571.

16 Florian Kurpicz, Hans-Peter Lehmann, and Peter Sanders. PaCHash: Packed and compressed hash tables. In *ALENEX*, pages 162–175. SIAM, 2023. doi:10.1137/1.9781611977561.CH14.

17 Hans-Peter Lehmann. ByteHamster/MPHF-Experiments. Software, swhId: swh:1:dir:7b02eda8ffa5a9d61a086ff2fc34fb0fbdc3eff4 (visited on 2025-07-16). URL: https://github.com/ByteHamster/MPHF-Experiments, doi:10.4230/artifacts.23790.

18 Hans-Peter Lehmann. *Fast and Space-Efficient Perfect Hashing*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2024. doi:10.5445/IR/1000176432.

19 Hans-Peter Lehmann, Thomas Mueller, Rasmus Pagh, Giulio Ermanno Pibiri, Peter Sanders, Sebastiano Vigna, and Stefan Walzer. Modern minimal perfect hashing: A survey. *CoRR*, abs/2506.06536, 2025. doi:10.48550/arXiv.2506.06536.

20 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. SicHash - small irregular cuckoo tables for perfect hashing. In *ALENEX*, pages 176–189. SIAM, 2023. doi:10.1137/1.9781611977561.CH15.

21 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. ShockHash: Near optimal-space minimal perfect hashing beyond brute-force. *CoRR*, abs/2310.14959, 2024. doi:10.48550/arXiv.2310.14959.

22 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. ShockHash: Towards optimal-space minimal perfect hashing beyond brute-force. In *ALENEX*, pages 194–206. SIAM, 2024. doi:10.1137/1.9781611977929.15.

23 Hans-Peter Lehmann, Peter Sanders, Stefan Walzer, and Jonatan Ziegler. ByteHamster/ConsensusRecSplit. Software, swhId: swh:1:dir:eeef738e12bea287eae54a77ce81241587a91767 (visited on 2025-07-16). URL: https://github.com/ByteHamster/ConsensusRecSplit, doi:10.4230/artifacts.23789.

24 Hans-Peter Lehmann, Peter Sanders, Stefan Walzer, and Jonatan Ziegler. Combined search and encoding for seeds, with an application to minimal perfect hashing. *CoRR*, abs/2502.05613, 2025. doi:10.48550/arXiv.2502.05613.

25 Harry G. Mairson. The program complexity of searching a table. In *FOCS*, pages 40–47. IEEE, 1983. doi:10.1109/SFCS.1983.76.

26 Kurt Mehlhorn. On the program size of perfect and universal hash functions. In *FOCS*, pages 170–175. IEEE, 1982. doi:10.1109/SFCS.1982.80.

27 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

28 Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008. doi:10.1137/060658400.

29 Anna Pagh, Rasmus Pagh, and Milan Ruzic. Linear probing with constant independence. In *STOC*, pages 318–327. ACM, 2007. doi:10.1145/1250790.1250839.

30 Giulio Ermanno Pibiri and Roberto Trani. PTHash: Revisiting FCH minimal perfect hashing. In *SIGIR*, pages 1339–1348. ACM, 2021. doi:10.1145/3404835.3462849.

31 Robert F. Rice. Some practical universal noiseless coding techniques. *Jet Propulsion Laboratory, JPL Publication*, 1979.

32 Ian H Witten, Alistair Moffat, and Timothy C Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.

33 Wei-Pang Yang and M. W. Du. A backtracking method for constructing perfect hash functions from a set of mapping functions. *BIT*, 25(1):148–164, 1985. doi:10.1007/BF01934995.

34 Jonatan Ziegler. Compacting minimal perfect hashing using symbiotic random search. Bachelor's thesis, Karlsruhe Institute for Technology (KIT), 2025. doi:10.5445/IR/1000177814.