

# MorphisHash: Improving Space Efficiency of ShockHash for Minimal Perfect Hashing

Stefan Hermann 

Karlsruhe Institute of Technology, Germany

---

## Abstract

A minimal perfect hash function (MPHF) maps a set of  $n$  keys to unique positions  $\{1, \dots, n\}$ . Representing an MPHF requires at least  $\log_2(e) \approx 1.443$  bits per key. ShockHash is a technique to construct an MPHF and requires just slightly more space. It gives each key two random candidate positions. If each key can be mapped to one of its two candidate positions such that there is exactly one key mapped to each position, then an MPHF is found. If not, ShockHash repeats the process with a new set of random candidate positions. ShockHash has to store how many repetitions were required and for each key to which of the two candidate positions it is mapped. However, when a given set of candidate positions can be used as MPHF then there is not only one but multiple ways of mapping the keys to one of their candidate positions such that the mapping results in an MPHF. This redundancy makes up for the majority of the remaining space overhead in ShockHash. In this paper, we present MorphisHash which almost completely eliminates this redundancy. Our theoretical result is that MorphisHash saves  $\Theta(\ln(n))$  bits in expectation compared to ShockHash. This corresponds to a factor of 20 less space overhead in practice. Just like ShockHash, MorphisHash can be used as a building block within RecSplit to obtain MorphisHash-RS. When compared for same space consumption, MorphisHash-RS can be constructed up to 21 times faster than ShockHash-RS. The technique to accomplish this might be of a more general interest to compress data structures.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Data compression; Information systems  $\rightarrow$  Point lookups; Theory of computation  $\rightarrow$  Bloom filters and hashing; Mathematics of computing  $\rightarrow$  Random graphs

**Keywords and phrases** compressed data structure, perfect hashing, random graph, pseudoforest, component

**Digital Object Identifier** 10.4230/LIPIcs.ESA.2025.9

**Related Version** *Extended Version*: <https://arxiv.org/abs/2503.10161> [11]

**Supplementary Material** *Software*: <https://github.com/stefanfred/MorphisHash> [10]  
archived at `swb:1:dir:72bf7952109795567ca30d31efc4c557b44dfc17`

**Funding** This work was supported by funding from the pilot program Core Informatics at KIT (KiKIT) of the Helmholtz Association (HGF).


**Acknowledgements** I thank Stefan Walzer for proofreading an earlier version of this paper.

## 1 Introduction

Given a set  $S$  of  $n$  keys, a minimal perfect hash function (MPHF) maps each key to a unique position in  $[n] := \{1, \dots, n\}$ . MPHFs have a wide range of applications including compressed full-text indexes [2], computer networks [18], databases [6], prefix-search data structures [1], language models [24], bioinformatics [7, 23], and Bloom filters [5]. Different techniques exist for constructing an MPHF. They offer a variety of trade-offs between construction time, space consumption and query time. The space lower bound of an MPHF is  $\log_2(e) \approx 1.443$  bits per key [19].



© Stefan Hermann;  
licensed under Creative Commons License CC-BY 4.0  
33rd Annual European Symposium on Algorithms (ESA 2025).

Editors: Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman; Article No. 9; pp. 9:1–9:16  
Leibniz International Proceedings in Informatics  
 LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**ShockHash.** Our technique builds on ShockHash [15, 16]. Similar to Cuckoo hashing [22], each key is given two candidate positions using respective hash functions  $h_{s,0} : S \rightarrow [n]$  and  $h_{s,1} : S \rightarrow [n]$  with seed  $s$ . ShockHash finds a seed  $s$  such that all keys can be mapped to one of their candidate positions and there is exactly one key mapped to each position. ShockHash needs to store the seed  $s$ , once found. Additionally, it needs to store for each key  $k \in S$  if the candidate position  $h_{s,0}(k)$  or  $h_{s,1}(k)$  is used. This can be represented using a function  $f : S \rightarrow \{0, 1\}$ . Such a mapping can be stored efficiently using a retrieval data structure [8] which requires about 1 bit per key. A key  $k \in S$  is queried using  $h_{s,f(k)}(k)$ .

A different perspective on ShockHash is that with each seed  $s$  it samples a random graph. The  $[n]$  possible output positions are the nodes of that graph. The keys are the edges, connecting the nodes of their respective candidate positions. A seed is accepted if the graph can be oriented, i.e. each edge is given a direction, such that the indegree of each node is 1. This is possible if and only if the graph is a *pseudoforest* – a graph where each component is a cycle with trees branching from it. The edges of the cycle of each component are oriented either all “clockwise” or “counterclockwise”. The edges in the trees are uniquely oriented away from their cycle. Hence, the indegree of each node is 1. ShockHash arbitrarily chooses one of two possible orientations of each cycle and stores the according orientation of each edge in a retrieval structure. Hence, for each cycle there is one bit of redundancy.

**Contribution.** In this paper we introduce MorphisHash which exhausts this remaining redundancy. MorphisHash is a recursive acronym: **M**orphisHash is an **o**verloaded **r**etrieval structure for **p**erfect **h**ashing using **S**hock**H**ash. Our key observation is that the possible orientations of a pseudoforest can be described as the solution of a linear equation system. A retrieval structure that stores the edge orientations can also be described as the solution of a linear equation system. This allows us to concatenate the equation systems using matrix multiplication. We achieve compression by reducing the dimensionality of the solution space of the combined equation system. Our theoretical insight is that a random pseudoforest has  $\Theta(\ln(n))$  components in expectation and MorphisHash can convert this into  $\Theta(\ln(n))$  bits of expected space savings compared to ShockHash by utilizing the freedom of choosing the orientation of each component’s cycle. Our experiments show that MorphisHash has about a factor of 20 less space overhead than ShockHash at the cost of roughly 4 times more construction time.

**Partitioning.** Note that within this paper,  $n$  denotes the input size for one instance of MorphisHash. In Section 5, an MPHf is obtained by splitting a large input key set into a linear number of MorphisHash instances, each of size  $n$ , and concatenating them afterwards. Partitioning is required mainly for keeping construction times feasible. Furthermore, our per instance space savings translate to linear space savings in terms of the large input.

**Outline.** We begin in Section 2 with related space efficient PHF construction techniques. We present MorphisHash in Section 3 and analyze it in Section 4. We explain implementation details in Section 5. Finally, Section 6 discusses experiments and the paper is concluded with Section 7. Our compression technique might be of a more general interest and we give further examples in the full version of this paper [11].

## 2 Related Work

We provide a brief overview of other space efficient PHF techniques. For a detailed survey of state-of-the-art techniques we refer to [14].

**RecSplit.** RecSplit [9, 3] first hashes the keys into partitions of about 2000 keys. RecSplit then finds a seed of a hash function that splits the partition into smaller subsets of equal size. This is applied recursively resulting in a tree-like structure. Once sufficiently small subsets are obtained, RecSplit uses brute-force search to find an MPHf within each leaf. Very recently a significant improvement to RecSplit has been made with the introduction of CONSENSUS [17]. Instead of allowing arbitrarily large seeds, CONSENSUS-RecSplit uses a fixed number of bits for each seed and backtracks in the splitting tree if a seed cannot be found within the allowed space. CONSENSUS-RecSplit is currently the most space efficient technique with just 0.001 bits per key overhead.

**PHOBIC.** Another PHF construction technique is PHOBIC [12]. Again, the keys are hashed into partitions of about 2000 keys. Within each partition the keys are hashed to buckets which have an average size of about 10. For each bucket, PHOBIC uses brute force search to find a seed of a hash function such that all keys of that bucket are hashed to positions in  $[n]$  to which no keys of previous buckets are hashed to. The buckets are inserted in non-increasing order of size because it is much easier to insert the larger buckets when the output domain is almost empty. This effect is utilized further by intentionally making some of the buckets larger. PHOBIC has fast queries at the cost of more space overhead.

### 3 MorphisHash

ShockHash samples random graphs until stumbling on a pseudoforest. The only remaining degree of freedom when orienting the pseudoforest is that each component contains a cycle and there are two ways to orient each cycle. We address this remaining redundancy with MorphisHash.

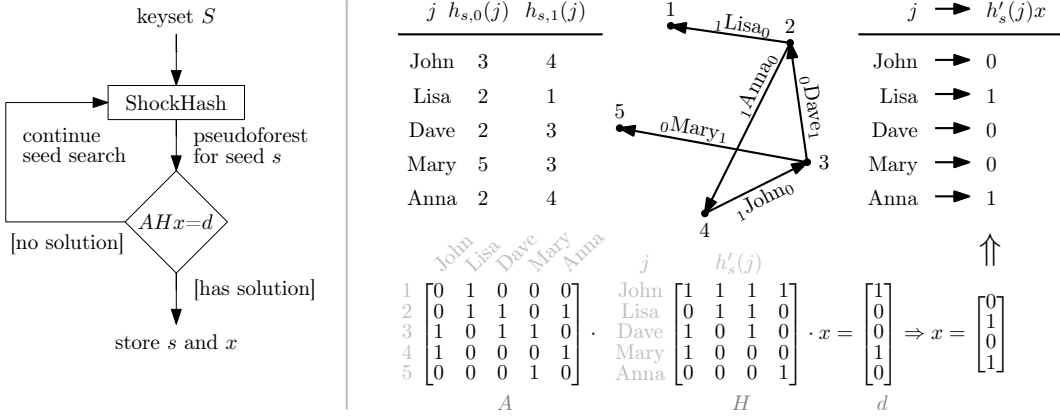
The first ingredient of MorphisHash is the insight that all allowed orientations of a graph can be expressed as an affine subspace of  $\mathbb{F}_2^S$ , where  $S$  is the set of  $n$  keys. To show this, we define  $y \in \mathbb{F}_2^S$  as the vector representing the orientation of each edge such that an edge  $j \in S$  is oriented to node  $h_{s,y_j}(j)$ . We now consider a given pseudoforest and one possible orientation  $y$ . We can flip the orientation of a cycle by adding a vector  $v \in \mathbb{F}_2^S$  to  $y$  where  $v_j = 1$  if and only if  $j$  is part of that cycle. This can be done for each cycle independently. Clearly, the dimension of this subspace is equal to the number of components. Part of this section is to describe the linear equation system of which the solution space is our desired affine subspace.

The second ingredient is a 1-bit retrieval data structure. The retrieval structure works by storing a bit vector  $x \in \mathbb{F}_2^b$ , where the parameter  $b \in \mathbb{N}_0$  is discussed in detail later. The orientation of an edge  $j \in S$  is described using  $h'_s(j)x$ , where  $h'_s : S \rightarrow \mathbb{F}_2^{1 \times b}$  is a hash function and  $s$  is the ShockHash seed. Using linear equations that involve hash functions is a common technique for retrieval data structures [8].

The beauty of MorphisHash is that we can concatenate both linear equation systems using a simple matrix multiplication to find a retrieval structure which directly orients the edges correctly. We can then decrease the number of bits  $b$  that the retrieval structure is allowed to use which reduces the dimension of the solution space and therefore extracts the remaining redundancy.

We now show the equation system that describes the allowed edge orientations. As a first step we show that the constraints of an MPHf can be weakened in the following sense:

► **Lemma 1.** *A function  $f : S \rightarrow [n]$  with  $|S| = n$  is an MPHf (i.e. a bijection) if and only if for all  $i \in [n]$  we have that  $|\{f^{-1}(i)\}|$  is uneven.*



■ **Figure 1** Left: MorphisHash uses ShockHash as a black box. Right: An example. ShockHash has found a pseudoforest using the candidate positions  $h_{s,0}$  and  $h_{s,1}$  as shown in the table and also illustrated as a graph. The description  $1\text{Lisa}_0$  indicates that  $h_{s,1}$  is the left candidate and  $h_{s,0}$  the right. This can be used to find  $d$ . For example,  $d_3 = 0^{\text{Dave}} + 0^{\text{Mary}} + 1^{\text{John}} + 1^{\text{offset}} = 0$ . The incidence matrix  $A$  and some random matrix  $H$  is shown. A solution  $x$  is found resulting in the orientations as shown both in the right table and using arrows in the graph. The solution  $x$  describes the orientation of 5 edges but only requires 4 bits to store. The figure is derived from [16, Fig. 1].

**Proof.** Clearly, if  $f$  is bijective then  $|\{f^{-1}(i)\}| = 1$  is uneven. If  $f$  is not bijective then there is at least one  $i$  such that  $|\{f^{-1}(i)\}| = 0$  which is even. ◀

**Linear Equations in Graphs.** This allows us to count the indegree of each node using  $\mathbb{F}_2$ : If the indegree of all nodes is  $1_{\mathbb{F}_2}$  then the orientations result in a valid MPHf. Recall the definition of  $y \in \mathbb{F}_2^S$  as the vector representing the orientation of each edge such that an edge  $j \in S$  is oriented to node  $h_{s,y_j}(j)$ . A different perspective is that an edge  $j$  is oriented towards a node  $i$  if and only if  $(h_{s,1}(j) = i \wedge y_j = 1) \vee (h_{s,0}(j) = i \wedge y_j + 1 = 1)$ , which will be useful in the following equation. We define  $A \in \mathbb{F}_2^{n \times S}$  as the incidence matrix of the graph. We also define  $d \in \mathbb{F}_2^n$  where  $d_i = |\{j \mid h_{s,0}(j) = i\}| + 1$  counts the number of edges (+1) that are mapped to position  $i$  using the  $h_{s,0}$  candidate function. Finally, this allows us to count the indegree of a node  $i$  as

$$\sum_{h_{s,1}(j)=i} y_j + \sum_{h_{s,0}(j)=i} (1 + y_j) = \left( \sum_{\substack{h_{s,0}(j)=i \\ \oplus h_{s,1}(j)=i}} y_j \right) + d_i + 1 = A_i y + d_i + 1$$

The  $\oplus$  operator is logical XOR. According to Lemma 1 setting  $A_i y + d_i + 1 = 1$  such that the indegree of each node  $i$  is uneven results in a valid orientation of all edges. The complete equation system therefore simplifies to  $Ay = d$ , which has a solution if and only if the graph is a pseudoforest. Note that in case of a loop, the respective column in the incidence matrix is a zero column.

**The Retrieval Data Structure.** To store the orientation of the edges we cannot use  $y$  directly because we do not know the index of a key during query time. We therefore employ the idea of a retrieval structure. Our retrieval structure consists of a bit vector  $x \in \mathbb{F}_2^b$ , where  $b$  is a tuning parameter. The orientation of edge  $j$  is then described using a scalar product  $y_j = h'_s(j)x$  where  $h'_s : S \rightarrow \mathbb{F}_2^{1 \times b}$  is a hash function and  $s$  is the ShockHash seed. MorphisHash needs to find  $x$  such that all edges are properly oriented. The above linear

equation is given for each key and can therefore be written using a matrix  $H$ . Each row  $H_j$  is the hash  $h'_s(j)$  of the respective key  $j$ . We have  $y = Hx$  and substitute it into  $Ay = d$  resulting in  $AHx = d$ . If a solution  $x$  for  $AHx = d$  exists then the graph is a pseudoforest,  $y = Hx$  is a valid orientation and  $x$  requires exactly  $b$  bits to store using a bit string. We discuss the selection of  $b$  both in theory (Section 4) and practice (Section 6). Querying our MPHf for a key  $j$  is now straightforward  $h_{s,h'_s(j)x}(j)$ .

If the system  $AHx = d$  does not have a solution for a sampled graph this can have two reasons. (1) The graph is no pseudoforest (2) A possible orientation of the pseudoforest is not within the solution space of the retrieval structure. If there is no solution we reject the seed and ShockHash continues with searching for a new seed.

The already existing step in ShockHash of checking whether a seed is a pseudoforest is therefore redundant. However, solving an equation system is computationally more expensive than using the original ShockHash pseudoforest check. We therefore leave the original pseudoforest check as a filter and only need to solve  $AHx = d$  a few times in practice. MorphisHash is illustrated using an example in Figure 1.

**Bipartite MorphisHash.** A variant of ShockHash is bipartite ShockHash. Assume that  $n$  is even, the extension for uneven numbers can be found in the original paper [15]. In bipartite ShockHash, the ranges of the two hash functions are made disjoint using  $h_{s_0,0} : S \rightarrow \{1, \dots, \frac{n}{2}\}$  and  $h_{s_1,1} : S \rightarrow \{\frac{n}{2} + 1, \dots, n\}$ . The seed is also split in two independent parts  $s_0$  and  $s_1$ . Seeds where not all positions are hit by at least one candidate position are filtered out. Bipartite ShockHash only checks if pairs of  $s_0$  and  $s_1$  that passed the filter result in a pseudoforest. MorphisHash uses ShockHash as a black box and can be applied on the bipartite case just as well as on the non-bipartite case. In an obvious manner, we refer to bipartite MorphisHash if bipartite ShockHash is used.

## 4 Analysis

In this section we analyze non-bipartite MorphisHash for large  $n$ . Our analysis is split in two parts. First, we analyze the number of components of a random pseudoforest. We use this in the second part, to show the space efficiency of MorphisHash compared to ShockHash. Note that we employ the common simple uniform hashing assumption [21], which assumes that hash functions behave like truly random functions.

### 4.1 The Number of Components in a Random Pseudoforest

In a pseudoforest each component is a tree with one additional edge. An equivalent view is that each component of the pseudoforest is a cycle with trees branching from it.

Our first result uses the following graph model. Let  $R = [n]^{[n]}$  be the set of all functions from  $[n]$  to  $[n]$ . Every  $r \in R$  corresponds to a directed graph  $G_r = ([n], \{(v, r(v)) \mid v \in [n]\})$ . All  $G_r$  are pseudoforests, because if we start from any node and follow its edges we will eventually end up in a cycle. Furthermore, there can only be one cycle in each component because each node has an out-degree of one. In the following we refer to all  $G_r$  as *maximal directed pseudoforests*. All edges of the pseudoforest which are not part of the cycle are pointed towards the cycle and all edges in the cycle are all directed in the same direction. We will first need the following results.

► **Lemma 2** ([25]). *Let  $T(n, i)$  be the number of undirected forests having node set  $[n]$  with  $i$  components where nodes  $1, 2, \dots, i$  belong to different trees. We have  $T(n, i) = in^{n-i-1}$ .*

► **Lemma 3.**  $|\{r \in R \mid \text{all components in } G_r \text{ are cycles}\}| = n!$

**Proof.** If each node in  $G_r$  is part of a cycle then the indegree of each node is 1. Hence,  $r$  is a bijection and there are  $n!$  possible bijections. ◀

We can now combine the previous results to analyze how the number of nodes in cycles is distributed in maximal directed pseudoforests.

► **Lemma 4.**  $|\{r \in R \mid G_r \text{ is pseudoforest and has } i \text{ nodes in cycles}\}| = \frac{in^{n-1-i}n!}{(n-i)!}$ .

**Proof.** There are  $\binom{n}{i}$  ways of partitioning the nodes into (1) cycles with a total of  $i$  nodes and (2) trees that are attached to the cycles. According to Lemma 3 there are  $i!$  ways to create cycles with  $i$  nodes. The  $i$  nodes in the cycles are the roots of trees. According to Lemma 2 the number of labeled rooted trees with  $n$  nodes and  $i$  roots is  $in^{n-i-1}$ . This number holds for graphs where each node has one outgoing edge, because all edges are uniquely oriented towards their parent node. We therefore have

$$\binom{n}{i} i! in^{n-i-1} = \frac{n!}{i!(n-i)!} i! in^{n-i-1} = \frac{in^{n-1-i}n!}{(n-i)!} \quad \blacktriangleleft$$

The previous result is based on maximal directed pseudoforests. However, in MorphisHash we sample graphs in a different manner. Let  $Q = [n]^{S \times \{0,1\}}$  be the set of functions from  $S \times \{0,1\}$  to  $[n]$ . Each  $q \in Q$  corresponds to a graph  $G_q = ([n], \{\{q(x,0), q(x,1)\} \mid x \in S\})$ . We refer to the set of all  $G_q$  as the *hashed* graph model. Graphs in this model may have multiple edges and loops. MorphisHash uniformly samples functions  $q \in Q$ . We are interested in the distribution of  $G_q$  conditioned on the event that  $G_q$  is a pseudoforest. In the following, we transfer our result of maximal directed pseudoforests to the hashed graph model.

► **Lemma 5.** Let  $c(G)$  be the number of components of  $G$ ,  $C_n$  an appropriate normalization constant,  $q \sim \mathcal{U}(Q)$  and  $r \sim \mathcal{U}(R)$  then

$$\begin{aligned} p_n(i) &:= \mathbb{P}[G_q \text{ has } i \text{ nodes in cycles} \mid G_q \text{ is pseudoforest}] \\ &= C_n \mathbb{E}[2^{-c(G_r)} \mid G_r \text{ has } i \text{ nodes in cycles}] \frac{i}{n^i(n-i)!} \end{aligned}$$

**Proof.** We begin with a relation between maximal directed pseudoforests  $G_r$  and hashed pseudoforests  $G_q$ . A hashed pseudoforest  $G_q$  is related to a maximal directed pseudoforest  $G_r$  if we can obtain  $G_r$  by orienting the edges of  $G_q$ . Let  $a_{k,d,l}$  be the number of maximal directed pseudoforests and  $b_{k,d,l}$  the number of hashed pseudoforest with  $i$  nodes in cycles,  $k$  components,  $d$  double edges and  $l$  loops. Within this subset, each cycle of the hashed graph can be oriented in two possible ways to obtain a maximal directed pseudoforest. However, changing the orientation of cycles that are a loop or double edge does not result in different maximal directed pseudoforests. Hence, we have  $\frac{a_{k,d,l}}{b_{k,d,l}} = 2^{k-d-l}$  orientations which result in different maximal directed pseudoforest.

Let  $c_{k,d,l}$  be the number of elements  $q \in Q$  such that  $G_q$  is a pseudoforest with  $i$  nodes in cycles,  $k$  components,  $d$  double edges and  $l$  loops. We show that  $\frac{c_{k,d,l}}{b_{k,d,l}} = n! 2^{n-d-l}$ . We have to consider the number of functions in  $Q$  that result in the same hashed pseudoforest. The order of the edges can be permuted in  $n!$  ways without changing the underlying pseudoforest, except for double edges. The number of possible permutations decreases by a factor of two for each double edge ( $2^{-d}$ ). Analogously, the nodes within the  $n$  edges can be switched without changing the underlying graph in  $2^n$  ways, except for loops ( $2^{-l}$ ).

By definition we have  $|Q_i| = \sum_{k,d,l} c_{k,d,l}$  and  $|R_i| = \sum_{k,d,l} a_{k,d,l}$ , where  $Q_i \subset Q$  and  $R_i \subset R$  are the pseudoforests with  $i$  nodes in cycles. For brevity let

$$p_{k,d,l} := \frac{a_{k,d,l}}{|R_i|} = \mathbb{P}[G_r \text{ has } k \text{ components, } d \text{ double edges and } l \text{ loops} \mid G_r \text{ has } i \text{ nodes cycles}]$$

We have

$$\begin{aligned} p_n(i) &= \frac{|Q_i|}{\sum_j |Q_j|} = \frac{|R_i|}{\sum_j |Q_j|} \frac{\sum_{k,d,l} c_{k,d,l}}{|R_i|} = \frac{|R_i|}{\sum_j |Q_j|} \sum_{k,d,l} \frac{a_{k,d,l} c_{k,d,l}}{a_{k,d,l} |R_i|} \\ &= \frac{|R_i|}{\sum_j |Q_j|} \sum_{k,d,l} p_{k,d,l} \frac{c_{k,d,l}}{a_{k,d,l}} = \frac{|R_i|}{\sum_j |Q_j|} \sum_{k,d,l} p_{k,d,l} \frac{c_{k,d,l}}{b_{k,d,l}} \frac{b_{k,d,l}}{a_{k,d,l}} \\ &= \frac{|R_i|}{\sum_j |Q_j|} \sum_{k,d,l} p_{k,d,l} n! 2^{n-d-l} 2^{-k+d+l} = \frac{|R_i| n! 2^n}{\sum_j |Q_j|} \sum_{k,d,l} p_{k,d,l} 2^{-k} \\ &= \frac{|R_i| n! 2^n}{\sum_j |Q_j|} \mathbb{E}[2^{-c(G_r)} \mid G_r \text{ has } i \text{ nodes in cycles}] \\ &= C_n \mathbb{E}[2^{-c(G_r)} \mid G_r \text{ has } i \text{ nodes in cycles}] \frac{i}{n^i (n-i)!} \end{aligned}$$

Where we used Lemma 4 for  $|R_i| \cdot \sum_j |Q_j|$  is the number of all hashed pseudoforests and thus only depends on  $n$ .  $C_n$  is chosen such that the probabilities add up to 1.  $\blacktriangleleft$

The next step is to analyze  $\mathbb{E}[2^{-c(G_r)} \mid G_r \text{ has } i \text{ nodes in cycles}]$ . To this end, we first require some definitions and more general results.

► **Lemma 6.** *The number of components of a pseudoforest with  $n$  nodes and  $i$  nodes in cycles follows the same distribution as the number of components of a graph of  $i$  nodes where each component is a cycle.*

Note that this refers to both graph models  $G_r$  and  $G_q$ .

**Proof.** The order in which the edges of a graph are sampled does not change the distribution of the graph. Given a graph of  $n$  nodes and  $i$  edges such that all edges are part of cycles, the probability that the remaining  $n - i$  edges form trees with the root being part of the existing cycles is independent of how the nodes of the cycles are connected to form any number of components.  $\blacktriangleleft$

**Configuration Model.** The *configuration model* [20] can be used to describe distributions of random graphs. In the model, each node is given a fixed number of half-edges. The graph is obtained by repeatedly connecting half-edges by uniformly sampling from the set of all remaining half-edges.

► **Lemma 7.** *The distribution of the hashed graph  $G_q$  with  $q \sim \mathcal{U}(Q)$  is equal to the distribution of the following graph.*

*The graph is obtained in two steps. First, the degree of each node is revealed by distributing  $2n$  half-edges. In a second step, the edges of the graph are obtained in a sequence of  $n$  rounds. In each round an unmatched half-edge  $i \in [2n]$  is chosen arbitrarily and matched to a distinct unmatched half-edge  $j \in [2n]$ , chosen uniformly at random. The choice of  $i$  may depend on the set of half-edges matched previously.*

We refer to ShockHash [15, Lemma 5] for a proof. We require an analogous result for  $G_r$ .



► **Lemma 8.** *The distribution of the maximal directed pseudoforest  $G_r$  with  $r \sim \mathcal{U}(R)$  is equal to the distribution of the following directed graph.*

*The number of outgoing edges of each node is fixed to 1. The graph is obtained in two steps. First, the indegree of each node is revealed by distributing  $n$  incoming half-edges. In a second step, the edges of the graph are obtained in a sequence of  $n$  rounds. In each round an unmatched outgoing half-edge at node  $i \in [n]$  is chosen arbitrarily and matched to a distinct unmatched incoming half-edge  $j \in [n]$ , chosen uniformly at random. The choice of  $i$  may depend on the set of half-edges matched previously.*

Again, the proof is analogous to ShockHash [15, Lemma 5].

► **Lemma 9.** *For  $i \in \mathbb{N} : \prod_{k=2}^i (1 - \frac{1}{k}) = \frac{1}{i}$*

**Proof.** By induction  $\prod_{k=2}^1 (1 - \frac{1}{k}) = 1$  and

$$\prod_{k=2}^{i+1} (1 - \frac{1}{k}) = (1 - \frac{1}{i+1}) \prod_{k=2}^i (1 - \frac{1}{k}) = (1 - \frac{1}{i+1}) \frac{1}{i} = \frac{1}{i+1} \quad \blacktriangleleft$$

We now have the available tools to analyze the number of components conditioned on the number of nodes in cycles.

► **Lemma 10.**  $\frac{1}{4}i^{-1/2} \leq \mathbb{E}[2^{-c(G_r)} \mid G_r \text{ has } i \text{ nodes in cycles}] \leq i^{-1/2}$ , where  $r \sim \mathcal{U}(R)$  and  $c(G_r)$  is the number of components of  $G_r$ .

Note that the following proof has similarities to ShockHash [15, Lemma 6].

**Proof.** We consider the number of components of a maximal directed pseudoforest  $G_r^i$  with  $i$  nodes,  $R = [i]^{[i]}$ ,  $r \sim \mathcal{U}(R)$ , conditioned on the event that each component of  $G_r^i$  is a cycle. According to Lemma 6 the number of components of  $G_r^i$  follows the same distribution as the number of components of a maximal directed pseudoforest with  $n$  nodes and  $i$  nodes in cycles. We proceed as described in Lemma 8 by revealing the graph in a sequence of rounds. First, the indegree of each node is revealed. Each component is a circle and therefore each node has indegree one. We choose the outgoing half-edge at an arbitrary node  $x$ . The outgoing half-edge is matched with one of the  $i$  incoming half-edges. Let  $y$  be the node at the incoming half-edge. There are two cases.

1. With probability  $\frac{1}{i}$  we have  $x = y$ . In this case, a loop is created, a cycle is closed and the next outgoing half-edge to match is chosen arbitrarily.
2. Otherwise we merge the nodes to a single one which does not change the number of components. The outgoing half-edge at node  $y$  is matched next.

In both cases, the distribution of the remaining graph is that of  $G_r^{i-1}$ . Because of this independence, we can multiply the expectation values to obtain the recurrence

$$\mathbb{E}[2^{-c(G_r^i)}] = \frac{1}{2} \frac{1}{i} \mathbb{E}[2^{-c(G_r^{i-1})}] + \left(1 - \frac{1}{i}\right) \mathbb{E}[2^{-c(G_r^{i-1})}] = \left(1 - \frac{1}{2i}\right) \mathbb{E}[2^{-c(G_r^{i-1})}]$$

With the base case  $\mathbb{E}[2^{-c_0}] = 1$  the recurrence is solved and upper bounded by

$$\begin{aligned} \mathbb{E}[2^{-c(G_r^i)}] &= \prod_{k=1}^i \left(1 - \frac{1}{2k}\right) = \sqrt{\prod_{k=1}^i \left(1 - \frac{1}{2k}\right)^2} \leq \sqrt{\prod_{k=1}^i \left(1 - \frac{1}{2k}\right) \left(1 - \frac{1}{2k+1}\right)} \\ &= \sqrt{\prod_{k=2}^{2i+1} \left(1 - \frac{1}{k}\right)} = \sqrt{\frac{1}{2i+1}} \leq i^{-1/2} \end{aligned}$$



Where we used Lemma 9. Analogously, a lower bound is

$$\begin{aligned}
 \mathbb{E}[2^{-c(G_r^i)}] &= \sqrt{\left(1 - \frac{1}{2}\right)^2 \prod_{k=2}^i \left(1 - \frac{1}{2k}\right)^2} = \frac{1}{2} \sqrt{\prod_{k=2}^i \left(1 - \frac{1}{2k}\right)^2} \\
 &\geq \frac{1}{2} \sqrt{\prod_{k=2}^i \left(1 - \frac{1}{2k-1}\right) \left(1 - \frac{1}{2k-2}\right)} \geq \frac{1}{2} \sqrt{\prod_{k=2}^{2i} \left(1 - \frac{1}{k}\right)} \\
 &= \frac{1}{2} \sqrt{\frac{1}{2i}} \geq \frac{1}{4} i^{-1/2}
 \end{aligned}$$

A similar result for hashed pseudoforests instead of maximal directed pseudoforests is the following.

► **Lemma 11.**  $\mathbb{P}[c(G_q) \geq \frac{1}{4} \ln(i) \mid G_q \text{ is pseudoforest with } i \text{ nodes in cycles}] \in \Omega_{i \rightarrow \infty}(1)$ , where  $q \sim \mathcal{U}(Q)$  and  $c(G_q)$  is the number of components of  $G_q$ .

**Proof.** The proof is analogous to the previous one. We use Lemma 7 to match a half-edge to one of the remaining  $2i - 1$  half-edges. In each round the number of components increases by one with probability  $\frac{1}{2i-1}$ . Resolving the respective recurrence results in

$$\begin{aligned}
 &\mathbb{E}[c(G_q) \mid G_q \text{ is pseudoforest with } i \text{ nodes in cycles}] \\
 &= \left(\frac{1}{2i-1} + \frac{1}{2i-3} + \frac{1}{2i-5} + \frac{1}{2i-7} + \dots + \frac{1}{3} + 1\right) \\
 &= \frac{1}{2} \left(\frac{1}{2i-1} + \frac{1}{2i-1} + \frac{1}{2i-3} + \frac{1}{2i-3} + \dots + 1 + 1\right) \\
 &> \frac{1}{2} \left(\frac{1}{2i-0} + \frac{1}{2i-1} + \frac{1}{2i-2} + \frac{1}{2i-3} + \dots + \frac{1}{2} + 1\right) \\
 &= \frac{1}{2} H_{2i} > \frac{1}{2} \ln(i)
 \end{aligned}$$

Where  $H_k$  is the  $k$ -th harmonic number. We are interested in a lower bound for the probability that the number of components is at least  $\frac{1}{4} \ln(i)$ . We can use the variance to find such a bound. The probability distribution of the number of components can be described in terms of a Poisson binomial distribution. The variance of a Poisson binomial distribution can be bounded above by the expected value. Clearly,  $H_{2i} < \ln(3i)$  is an upper bound for the variance. Using Cantelli's inequality ( $\mathbb{P}[X \geq \mathbb{E}[X] + x] \geq \frac{x^2}{x^2 + \sigma^2}$ ) we find with  $x = -\frac{1}{4} \ln(i)$  that with probability  $\frac{(\frac{1}{4} \ln(i))^2}{(\frac{1}{4} \ln(i))^2 + \ln(3i)} \in \Omega(1)$  the pseudoforest has at least  $\frac{1}{4} \ln(i)$  components. ◀

The previous result shows that the number of components increases logarithmically in the number of nodes in cycles. The next step is therefore to show that the cycles are sufficiently large.

► **Lemma 12.**  $\mathbb{P}[G_q \text{ has at least } \sqrt{n} \text{ nodes in cycles} \mid G_q \text{ is pseudoforest}] \in \Omega_{n \rightarrow \infty}(1)$ , where  $q \sim \mathcal{U}(Q)$ .

**Proof.** Let  $p_n(i)$  be the probability of sampling a hashed pseudoforest with  $i$  nodes in cycles as determined in Lemma 5. We need to show that the probability of sampling a pseudoforest with at least  $\sqrt{n}$  nodes in cycles is at least a constant factor larger than sampling a pseudoforest with less than  $\sqrt{n}$  nodes in cycles, i.e.  $\left(\sum_{i=\sqrt{n}}^n p_n(i)\right) / \left(\sum_{i=1}^{\sqrt{n}} p_n(i)\right) \in \Omega(1)$ .

A stronger statement is  $\left(\sum_{i=\sqrt{n}}^{2\sqrt{n}} p_n(i)\right) / \left(\sum_{i=1}^{\sqrt{n}} p_n(i)\right) \in \Omega(1)$ . In the following we show  $\frac{p_n(\sqrt{n}+x)}{p_n(\sqrt{n}-x)} \in \Omega(1)$  for all  $0 \leq x \leq \sqrt{n}$ . This pointwise comparison is an even stronger statement. For brevity, we omit rounding of  $\sqrt{n}$ . Using the bounds of Lemma 10 and Stirling's approximation we have:

$$\begin{aligned}
\frac{p_n(\sqrt{n}+x)}{p_n(\sqrt{n}-x)} &\geq \frac{1}{4} \frac{(\sqrt{n}-x)^{1/2}}{(\sqrt{n}+x)^{1/2}} \cdot \frac{\sqrt{n}+x}{\sqrt{n}-x} \cdot \frac{n^{\sqrt{n}-x}}{n^{\sqrt{n}+x}} \cdot \frac{(n-\sqrt{n}+x)!}{(n-\sqrt{n}-x)!} \\
&= \frac{1}{4} \left( \frac{\sqrt{n}+x}{\sqrt{n}-x} \right)^{1/2} \cdot n^{-2x} \cdot \sqrt{\frac{n-\sqrt{n}+x}{n-\sqrt{n}-x}} \cdot \frac{e^{n-\sqrt{n}-x}}{e^{n-\sqrt{n}+x}} \cdot \frac{(n-\sqrt{n}+x)^{n-\sqrt{n}+x}}{(n-\sqrt{n}-x)^{n-\sqrt{n}-x}} \\
&> \frac{1}{4} n^{-2x} \cdot e^{-2x} \cdot \left( \frac{n-\sqrt{n}+x}{n-\sqrt{n}-x} \right)^{n-\sqrt{n}} \cdot ((n-\sqrt{n}+x)(n-\sqrt{n}-x))^x \\
&= \frac{1}{4} e^{-2x} \cdot \left( \frac{1+\frac{x}{n-\sqrt{n}}}{1-\frac{x}{n-\sqrt{n}}} \right)^{n-\sqrt{n}} \cdot \left( \frac{(n-\sqrt{n}+x)(n-\sqrt{n}-x)}{n^2} \right)^x \\
&> \frac{1}{4} e^{-2x} \cdot \frac{e^x}{e^{-x}} \cdot \left( \frac{(n-\sqrt{n}+\sqrt{n})(n-\sqrt{n}-\sqrt{n})}{n^2} \right)^{\sqrt{n}} \\
&= \frac{1}{4} \left( \frac{n(n-2\sqrt{n})}{n^2} \right)^{\sqrt{n}} = \frac{1}{4} \left( 1 - \frac{2}{\sqrt{n}} \right)^{\sqrt{n}} = \frac{1}{4} \frac{1}{e^2}
\end{aligned}$$

Combining the results gives us the following.

► **Theorem 13.**  $\mathbb{P}[c(G_q) \geq \frac{1}{8} \ln(n) \mid G_q \text{ is pseudoforest}] \in \Omega_{n \rightarrow \infty}(1)$ , where  $c(G_q)$  is the number of components of  $G_q$  and  $q \sim \mathcal{U}(Q)$ .

**Proof.** According to Lemma 12 at least  $\sqrt{n}$  nodes are in cycles with constant probability. Using Lemma 11 this results in  $\frac{1}{4} \ln(\sqrt{n}) = \frac{1}{8} \ln(n)$  components with constant probability. ◀

► **Corollary 14.**  $\mathbb{E}[c(G_q) \mid G_q \text{ is pseudoforest}] \in \Theta_{n \rightarrow \infty}(\ln(n))$ , where  $c(G_q)$  is the number of components of  $G_q$  and  $q \sim \mathcal{U}(Q)$ .

**Proof.** Use Theorem 13 for the lower bound. There can be at most all  $n$  nodes in cycles of the pseudoforest and we can apply Lemma 11 for the upper bound. ◀

## 4.2 Space Savings of MorphisHash

We now show that MorphisHash can convert each component into space savings compared to ShockHash. As a first step we transition the previous graph results into the world of matrices.

► **Lemma 15.** *The defect of the incidence matrix  $A$  of a pseudoforest is at least the number of the pseudoforests components.*

**Proof.** For each component  $C$ , summing up the rows  $A_j$  of the respective nodes  $j \in C$  results in zero rows because for each component, the two endpoints of each edge are included in the summed up rows. The rows summed up for each zero row are disjoint and therefore in particular linearly independent combinations. ◀

► **Lemma 16** ([4]). *The probability that a random square matrix (of any field) with  $n$  rows and columns has full rank approaches 1 for large  $n$ .*

► **Lemma 17.** *The probability that a random rectangular matrix (of any field) with  $n$  rows and  $m \in o(n)$  columns has full rank approaches 1 for large  $n$ .*

**Proof.** A necessary condition that a square matrix with  $n$  rows has full rank is that the first  $m$  columns have full rank. The probability that this rectangular submatrix has full rank is therefore bounded below by Lemma 16. ◀

We now show our main result.

► **Theorem 18.**  $\mathbb{P}[\exists x : AHx = d \mid G_q \text{ is pseudoforest}] \in \Omega_{n \rightarrow \infty}(1)$ , where  $q \sim \mathcal{U}(Q)$ , incidence matrix  $A$  and vector  $d$  of  $G_q$  are as described in the algorithm,  $b = n - \frac{1}{9} \ln(n)$  and  $H \sim \mathcal{U}(\mathbb{F}_2^{n \times b})$ .

**Proof.** In Theorem 13 we showed that a hashed pseudoforest has at least  $\frac{1}{8} \ln(n)$  components with constant probability. According to Lemma 15 a direct consequence is that the incidence matrix  $A$  of a random pseudoforest has a defect of at least  $\frac{1}{8} \ln(n)$  with constant probability. According to Lemma 17, the probability that  $H$  has full rank is at least constant. The system  $AHx = d$  has a solution if there is a vector  $y$  which solves  $Ay = d$  and simultaneously  $Hx = y$ . Such a vector  $y$  exists if the two solution spaces have to intersect, which happens once  $\text{def}(A) + \text{rank}(H) - n > 0$ . Since  $A$  and  $H$  are uncorrelated we have with constant probability that both  $H$  has full rank  $b$  and simultaneously  $A$  has at least a defect of  $\frac{1}{8} \ln(n)$ . With  $b = n - \frac{1}{9} \ln(n)$  we have  $\text{def}(A) + \text{rank}(H) - n \geq \frac{1}{8} \ln(n) + (n - \frac{1}{9} \ln(n)) - n > 0$  with constant probability. ◀

We use the above result to measure the space savings compared to ShockHash.

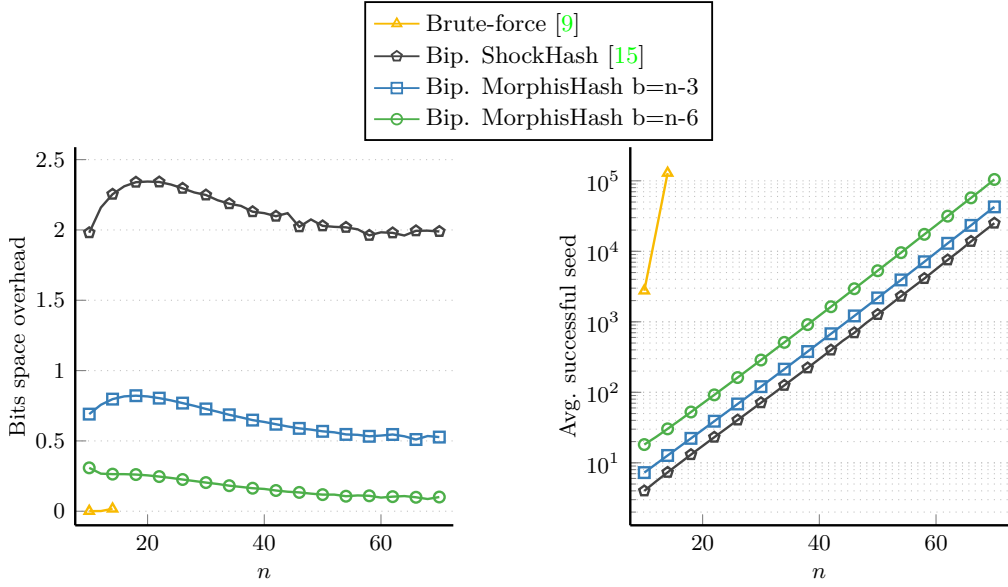
► **Corollary 19.** *Compared to ShockHash, MorphisHash is at least  $\frac{1}{9} \ln(n) - \mathcal{O}(1)$  bits more space efficient in expectation while requiring a constant factor more time.*

**Proof.** ShockHash requires at least  $n$  bits to store the orientation of all keys in a retrieval structure. MorphisHash has to store the solution vector  $x$  instead, requiring exactly  $b$  bits. According to Theorem 18 we can choose  $b = n - \frac{1}{9} \ln(n)$  to obtain the desired space savings. However, there is a constant probability that a seed pair has to be rejected because there is no solution for  $x$ . MorphisHash therefore has to check a constant factor more seeds in expectation, consequently increasing the expected space required to store the seed by a constant number of bits. Analogously, the construction time grows by a constant factor in expectation. The time required to solve an expected constant number of equation systems is dominated by the seed search. ◀

## 5 Partitioning

The time required to find a seed in MorphisHash and ShockHash grows exponentially with  $n$ . To keep construction feasible for a large number of keys, we first partition the input into equally sized subsets of manageable size. For consistency, we use  $n$  for the size of those subsets, and refer to them as *base cases* in the following. MorphisHash is then applied on those base cases. Note that the following two partitioning schemes are also applied on ShockHash and we refer to their paper for more details [15, Section 7].

**MorphisHash-RS.** We use RecSplit [9] to recursively split the input into smaller subsets. Once sufficiently small subsets are obtained we apply MorphisHash on those subsets as a base case. RecSplit is space efficient but has significant query time overheads caused by traversing the tree. In MorphisHash-RS we store the solution vector  $x$  of the retrieval structure directly next to the corresponding seed to improve locality.



(a) Idealized space overhead over the lower bound  $\log_2(n^n/n!)$ . For an average successful seed  $s$  we charge  $\log_2(s)$  bits plus the bits required for retrieval i.e.  $n$  for ShockHash and  $b$  for MorphisHash.

(b) Average successful seed. Note that we use Bip. ShockHash and Bip. MorphisHash without the quad split technique (see [15, Section 8.3]).

■ **Figure 2** Space and avg. successful seed for ShockHash, MorphisHash and brute force search.

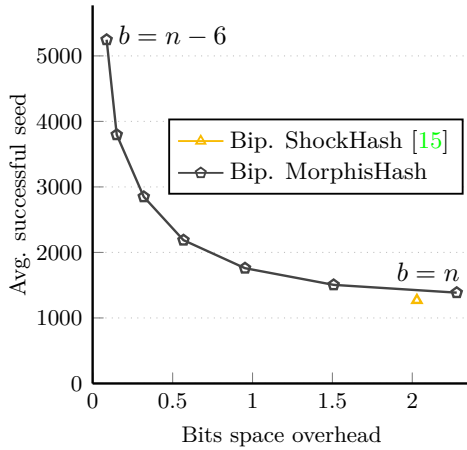
**MorphisHash-Flat.** The input keys are first hashed into buckets. Using thresholds, some keys are bumped such that the bucket does not exceed the desired base case size  $n$ . The bumped keys are then used to fill up buckets which did not reach the desired size. We apply MorphisHash on each bucket.

## 6 Experiments

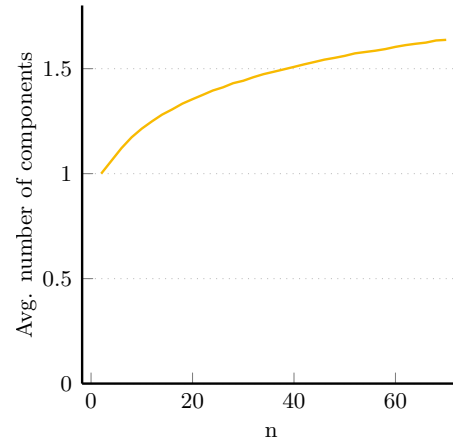
In this section we experimentally evaluate MorphisHash. We show the effect of the new parameter  $b$  introduced by MorphisHash. We then compare MorphisHash-Flat and MorphisHash-RS with state-of-the-art competitors. Our source code is public under the General Public License [10]. We integrate MorphisHash into an existing benchmark framework [13], which was used for the comparison with competitors. The benchmark framework is described in detail in [14]. We perform all experiments on a Core i7-11700 CPU which has 48 KiB L1 and 512 KiB L2 data cache per core. The CPU has a total of 16 MiB L3 cache. The machine has 64 GiB of dual-channel DDR4-3200 RAM. Note that our experiments are at a scale where variances are relatively small, we therefore omit them for better readability.

### 6.1 MorphisHash vs ShockHash without partitioning

In the following we compare Bipartite MorphisHash with Bip. ShockHash. MorphisHash has the additional parameter  $b$ , which determines the size of the retrieval structure. Smaller  $b$  values result in less space overhead at the cost of more seed tests. Note that in theory we choose  $b = n - \frac{1}{9} \ln(n)$  but in practice where  $n$  is small it is more intuitive to work with  $n$  minus a small constant. Figure 2 shows that Bip. MorphisHash has to check roughly a constant factor more seeds than Bip. ShockHash when  $b$  is fixed to  $n$  minus a constant. For



■ **Figure 3** Comparing ShockHash with the MorphisHash trade-off between time and space for fixed  $n = 50$  and  $b = \{n - 6, \dots, n\}$ .



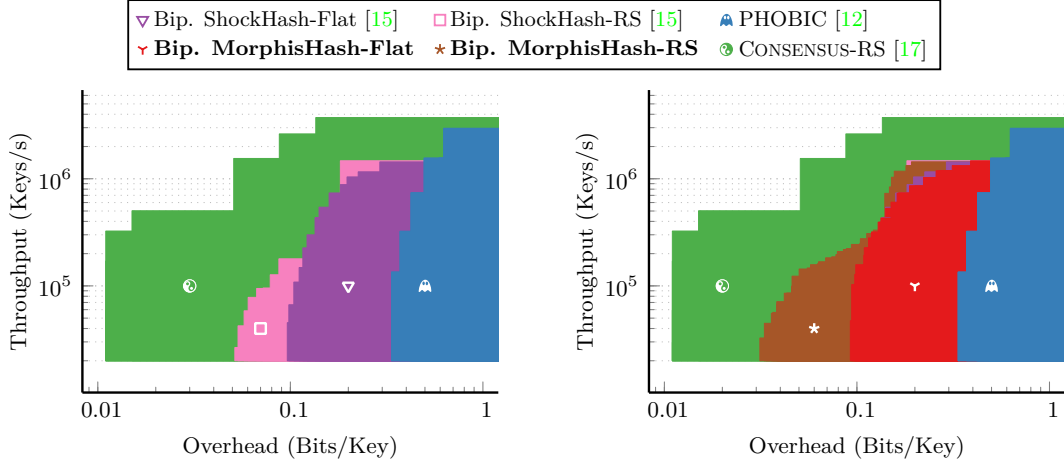
■ **Figure 4** Experimentally measured average number of components when sampling bipartite pseudoforests.

$b = n - 6$  this factor is  $\approx 4$ . At the same time, MorphisHash almost completely eliminates the remaining space overhead. MorphisHash comes below 0.1 bits of space overhead when using  $b = n - 6$  while ShockHash has about 2 bits of space overhead. Thus, MorphisHash has roughly 20 times less space overhead. For  $n = 54$ , MorphisHash has  $\frac{0.11}{54} \approx 0.002$  bits per key space overhead.

Figure 3 gives another perspective. In this plot, we fix  $n$  and vary  $b$ . Interestingly, ShockHash even outperforms MorphisHash for  $b = n$ . This is because MorphisHash requires  $n$  bits for the retrieval structure in this case just like ShockHash. However, there is still the chance that the equation system of MorphisHash has no solution resulting in more retries and therefore in a higher space consumption of the seed and a higher construction time. For smaller  $b$  the space overhead approaches 0. In the extreme case of  $b = 0$  and respective zero dimensional retrieval vector, MorphisHash is equivalent to simple brute force search because all keys can only use the  $h_0$  candidate function. The average successful seed grows rapidly with smaller  $b$  as it is increasingly less likely to stumble upon a pseudoforest which has at least  $n - b$  many components. A pseudoforest with less than  $n - b$  components may still result in a solvable (overdetermined) equation system, but this becomes exponentially less likely with every component below  $n - b$ . Figure 3 uses  $n = 50$ . The expected number of components for  $n = 50$  is 1.56 as shown in Figure 4.

## 6.2 Choosing $b$ in MorphisHash-RS and MorphisHash-Flat

Space improvements in MorphisHash-RS and MorphisHash-Flat can be made either by (1) increasing the base case size  $n$  which reduces the space overhead of the partitioning technique or (2) decreasing  $b$  which reduces the space overhead of MorphisHash. In both cases the construction time increases. We experimented with different values of  $b$  to identify the configurations which dominate the construction time and space trade-off. We determined that  $b = n - 4$  is a good choice for MorphisHash-RS and  $b = n - 2$  for MorphisHash-Flat. MorphisHash-Flat is a less space efficient partitioning technique compared to MorphisHash-RS. Space savings can be made more easily by increasing  $n$  instead of decreasing  $b$ .



■ **Figure 5** Dominance maps indicating the approach with the fastest queries, given a specific trade-off between space and construction time with (right) and without (left) MorphisHash on 100 M keys. Space overhead in bits per key over the lower bound of 1.44.

■ **Table 1** Performance of various methods on 100 M keys. In the first part, configurations are chosen such that the construction times are about equal (sorted by space efficiency). In the second part, configurations are chosen such that space consumption is almost equal.

Method	Space (bits/key)	Query (ns/query)	Construction (ns/key)
Consensus-RS, $k=32768$ , $o=0.0025$	1.447	222	6 733
Bip. MorphisHash-RS, base case size $n=52$ , $b=n-4$	1.501	137	6 669
Bip. ShockHash-RS, base case size $n=66$	1.523	147	7 186
Bip. MorphisHash-Flat, base case size $n=88$ , $b=n-2$	1.541	75	6 330
Bip. ShockHash-Flat, base case size $n=96$	1.554	76	6 676
PHOBIC, $\lambda=8.85$ , IC-R	1.749	49	6 426
Bip. ShockHash-RS, base case size $n=128$	1.489	131	172 738
Bip. MorphisHash-RS, base case size $n=64$ , $b=n-4$	1.489	139	8 085

### 6.3 Comparison to Competitors

We compare MorphisHash-RS and MorphisHash-Flat with state-of-the-art competitors. We select the following space efficient competitors based on a recent survey [14]: CONSENSUS-RS[17], ShockHash-Flat, ShockHash-RS and PHOBIC [12]. We test a wide range of configurations for each competitor and compare them in Figure 5. A selection of configurations is shown in Table 1. As can be seen in the plot and the table, MorphisHash-RS is about 0.02 bits per key more space efficient than ShockHash-RS when compared for equal construction time. This corresponds to a reduction of 27% in space overhead. When compared for equal space consumption of 1.489 bits per key, MorphisHash-RS is  $\frac{172\,738}{8\,085} \approx 21$  times faster to construct (see Table 1). For MorphisHash-Flat we select  $b$  less aggressively compared to MorphisHash-RS (Section 6.2) and obtain a space improvement of about 0.01 bits per key. According to Figure 5 MorphisHash dominates ShockHash in the overall space, construction and query time trade-off. The next best competitor in terms of space efficiency is PHOBIC which is a clear winner in terms of query throughput. In the other direction, we have the

recently published CONSENSUS-RS, which can reach space overheads as low as 0.001 bits per key at the cost of additional query time. A negative result regarding non-minimal PHFs can be found in the full version of this paper [11].

## 7 Conclusion and Future Work

MorphisHash almost completely eliminates the remaining redundancy in ShockHash. This is particularly effective when combined with a space efficient partitioning technique. Our compression scheme might be of a more general interest, further examples can be found in the full version of this paper [11].

In future work we plan to improve the space efficiency of partitioning techniques as those are a major source of space overhead. We are hopeful that a partitioning technique that involves the novel CONSENSUS technique puts further trade-offs for MorphisHash into reach.

---

### References

- 1 Djamal Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna. Fast prefix search in little space, with applications. In *ESA (1)*, volume 6346 of *Lecture Notes in Computer Science*, pages 427–438. Springer, 2010. doi:10.1007/978-3-642-15775-2\_37.
- 2 Djamal Belazzougui and Gonzalo Navarro. Alphabet-independent compressed text indexing. *ACM Trans. Algorithms*, 10(4):23:1–23:19, 2014. doi:10.1145/2635816.
- 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 Jean Bourgain, Van H Vu, and Philip Matchett Wood. On the singularity probability of discrete random matrices. *Journal of Functional Analysis*, 258(2):559–603, 2010.
- 5 Andrei Z. Broder and Michael Mitzenmacher. Survey: Network applications of Bloom filters: A survey. *Internet Math.*, 1(4):485–509, 2003. doi:10.1080/15427951.2004.10129096.
- 6 Chin-Chen Chang and Chih-Yang Lin. Perfect hashing schemes for mining association rules. *Comput. J.*, 48(2):168–179, 2005. doi:10.1093/COMJNL/BXH074.
- 7 Victoria G. Crawford, Alan Kuhnle, Christina Boucher, Rayan Chikhi, and Travis Gagie. Practical dynamic de bruijn graphs. *Bioinform.*, 34(24):4189–4195, 2018. doi:10.1093/BIOINFORMATICS/BTY500.
- 8 Peter C Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, and Stefan Walzer. Fast succinct retrieval and approximate membership using ribbon. *arXiv preprint*, 2021. arXiv:2109.01892.
- 9 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.
- 10 Stefan Hermann. MorphisHash - GitHub. <https://github.com/stefanfred/MorphisHash>, 2025.
- 11 Stefan Hermann. MorphisHash: Improving space efficiency of shockhash for minimal perfect hashing. *arXiv preprint*, 2025. doi:10.48550/arXiv.2503.10161.
- 12 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.
- 13 Hans-Peter Lehmann. MPHF Experiments – GitHub. <https://github.com/ByteHamster/MPHF-Experiments>, 2025.
- 14 Hans-Peter Lehmann, Thomas Mueller, Rasmus Pagh, Giulio Ermanno Pibiri, Peter Sanders, Sebastiano Vigna, and Stefan Walzer. Modern minimal perfect hashing: A survey. *arXiv preprint*, 2025. doi:10.48550/arXiv.2506.06536.



- 15 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. ShockHash: Near optimal-space minimal perfect hashing beyond brute-force. *arXiv preprint, invited to Algorithmica*, 2024. doi:10.48550/arXiv.2310.14959.
- 16 Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer. Shockhash: Towards optimal-space minimal perfect hashing beyond brute-force. In *ALENEX*. SIAM, 2024. doi:10.1137/1.9781611977929.15.
- 17 Hans-Peter Lehmann, Peter Sanders, Stefan Walzer, and Jonatan Ziegler. Combined search and encoding for seeds, with an application to minimal perfect hashing. In *ESA, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik*, 2025.
- 18 Yi Lu, Balaji Prabhakar, and Flavio Bonomi. Perfect hashing for network applications. In *ISIT*, pages 2774–2778. IEEE, 2006. doi:10.1109/ISIT.2006.261567.
- 19 Kurt Mehlhorn. On the program size of perfect and universal hash functions. In *FOCS*, pages 170–175. IEEE Computer Society, 1982. doi:10.1109/SFCS.1982.80.
- 20 MARK EJ Newman. *Networks: an introduction*, 2010.
- 21 Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM Journal on Computing*, 38(1):85–96, 2008. doi:10.1137/060658400.
- 22 Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004. doi:10.1016/J.JALGOR.2003.12.002.
- 23 Giulio Ermanno Pibiri. Sparse and skew hashing of k-mers. *Bioinformatics*, 38(Supplement\_1):i185–i194, 2022. doi:10.1093/BIOINFORMATICS/BTAC245.
- 24 Giulio Ermanno Pibiri and Rossano Venturini. Efficient data structures for massive  $N$ -gram datasets. In *SIGIR*, pages 615–624. ACM, 2017. doi:10.1145/3077136.3080798.
- 25 Lajos Takács. On cayley’s formula for counting forests. *Journal of Combinatorial Theory, Series A*, 53(2):321–323, 1990. doi:10.1016/0097-3165(90)90064-4.