Learned Monotone Minimal Perfect Hashing

Paolo Ferragina\textsuperscript{1}, Hans-Peter Lehmann\textsuperscript{2}, Peter Sanders\textsuperscript{2}, and Giorgio Vinciguerra\textsuperscript{1}
Monotone Minimal Perfect Hash Functions (MMPHFs)

Given a set \( S \) of \( n \) keys from a universe \([u] = \{0, \ldots, u - 1\}\)

Construct a hash function that maps keys \( \in S \) to their rank, and keys \( \notin S \) to an arbitrary value.
Why are MMPHFs interesting?

Minimal Perfect Hash Functions (MPHF)

- Take \( \geq 1.44 \) bits/key
- But no ranks

MMPHFs [SODA 09]

- Return ranks of keys in \( S \) in \( \mathcal{O}(\log \log \log u) \) bits/key

Order-Preserving MPHFs [TOIS 91]

- Return ranks of keys in \( S \) in \( \Omega(\log n) \) bits/key

Exploit lex order

Any order

Rank data structures

Compressed \( S \)

Index on \( S \)

Applications of MMPHFs in databases, pattern matching, and search engines
Key tool: Retrieval data structures

• Associate given $r$-bit values to keys in $S$, and retrieve them in $O(1)$ time

• Take $rn$ bits + small overhead
  - $o(n)$ bits in theory Dietzfelbinger and Pagh [ICALP 08], Porat [CSR 09]
  - $< 0.01 rn$ bits in practice with BuRR Dillinger et al. [SEA 22]
Known approaches for MMPHFs

1. Form equal-size buckets and store local ranks with a retrieval data structure
2. Build a (relative) rank data structure on the bucket delimiters $D$ to route keys in $S$ to buckets

**Space:** $\mathcal{O}(\log \log \log u)$ bits/key

**Queries:** $\mathcal{O}(\log \log u)$ time

---

This is optimal, Assadi et al. [SODA 23]
Our MMPHF

$n$ input keys

Universe $[u]$

Monotone function (discussed later)

$n$ buckets
Our MMPHF

$n$ input keys

Universe $[u]$  

Monotone function (discussed later)

$n$ buckets

Global ranks

$2n + o(n)$ bits

\[
\begin{array}{cccccccc}
0 & 1 & 1 & 3 & 4 & 4 & 5 & 6 & 6 & 9 & 10 & 11 \\
\end{array}
\]
Our MMPHF

- **n** input keys
- Universe \([u]\)
- Monotone function (discussed later)
- **n** buckets
- Global ranks

2\(n + o(n)\) bits

- Retrieval data structures storing local ranks for buckets of size \(\geq 2\)
How to map to buckets

• Suppose the input integers are **uniform**

• Map $x$ to bucket number $\left\lfloor \frac{x}{n} \right\rfloor$, i.e. a linear mapping

**Theorem 1.**
Our MMPHF on **uniform** integers needs $n(2.915 + o(1))$ bits on average and can be queried in $\mathcal{O}(1)$ time.

• This breaks the lower bound of $\Omega(n \log \log \log u)$ bits for a MMPHF

• Learning and leveraging the input data smoothness: **LeMonHash**
How to map non-uniform data

- Learn a **piecewise linear $\varepsilon$-approximation** of the function keys $\rightarrow$ ranks [Ferragina, V. [VLDB 20]]
  - $|\text{Rank estimate} - \text{True rank}| \leq a$ given integer $\varepsilon$
  - The more the data is smooth the smaller is the number of segments
- Rank estimate for a key = bucket index
LeMonHash bounds

Theorem 1.
LeMonHash on uniform integers takes $n(2.915 + o(1))$ bits on average and can be queried in $\mathcal{O}(1)$ time.

Theorem 2.
LeMonHash takes $n\left(\log(2\epsilon + 1) + 2 + o(1)\right) + \mathcal{O} \left( m \log \frac{u}{m} \right)$ bits in the worst case and can be queried in $\mathcal{O} ( \log \log u )$ time.
Handling variable-length strings

\[ |LCP| = 1 \]

- salmon
- sawfish
- scallops
- sea_lion
- sea_snake
- sea_spider

\[ w \]-bit chunks

\[ [2^w] \]

Global ranks

- salmon
- sawfish
- scallops
- sea_lion
- sea_snake
- sea_spider

#strings \( \geq t \) \( \Rightarrow \) handle recursively thus creating a tree

Many optimisations in the paper
Experiments: space vs query throughput

String data

Query MKeys/s

Existing Pareto front:
- **Hollow-trie** approaches
- **ZFast / PaCo**
- **LCP** approaches

LeMonHash:
- On *string data*, space within 13% of the best competitors, and up to 3× faster queries than the larger competitor
- On *integer data*, dominates in space-time all competitors (except for the space on fb)
- Improved construction throughput by up to 2×
Conclusion

LeMonHash: New MMPHF that learns and leverages data smoothness

• Can break the superlinear lower bound on MMPHFs’ space

• In practice: on most datasets, dominates all competitors on space usage, query and construction throughput, simultaneously

Open problems

1. Strengthen our pessimistic bounds

2. Extend to nonlinear models