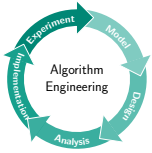
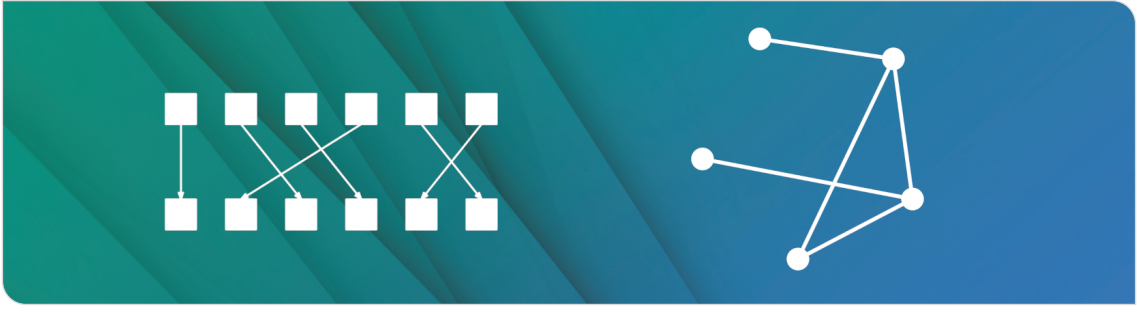
Karlsruhe Institute of Technology



ShockHash: Towards Optimal-Space Minimal Perfect Hashing Beyond Brute-Force

ALENEX 2024, Alexandria

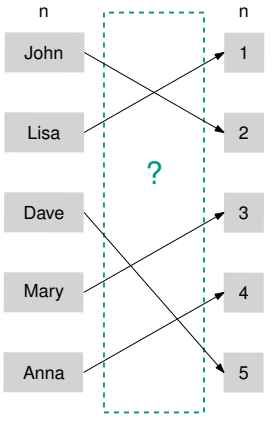
Hans-Peter Lehmann, Peter Sanders, Stefan Walzer | Jan 8, 2024



Minimal Perfect Hashing

- Static set of n keys
- Bijectively map keys to the first n integers

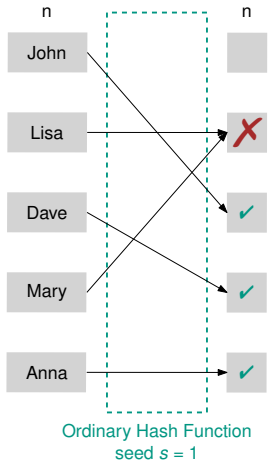
- Applications: databases, hash tables, AMQ, retrieval
- Get as close as possible to the space lower bound



Minimal Perfect Hash Function

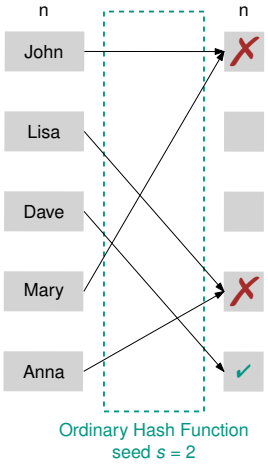
Brute-Force Construction

- Try ordinary hash functions
- Perfect hash function data structure: store successful seed s
- Expected tries: $n^n/n! \approx e^n$
  $\Rightarrow\approx n\log_2(e)$ bits (this is optimal)



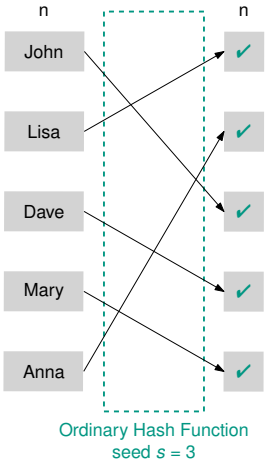
Brute-Force Construction

- Try ordinary hash functions
- Perfect hash function data structure: store successful seed s
- Expected tries: $n^n/n! \approx e^n$
  $\Rightarrow\approx n\log_2(e)$ bits (this is optimal)



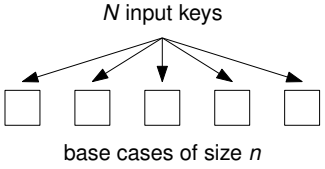
Brute-Force Construction

- Try ordinary hash functions
- Perfect hash function data structure: store successful seed s
- Expected tries: $n^n/n! \approx e^n$
  $\Rightarrow\approx n\log_2(e)$ bits (this is optimal)



Supporting Large Input Sets

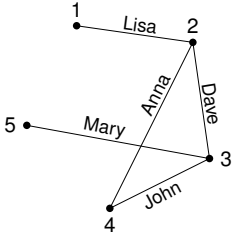
- Brute-force of e^n seeds not practical for large n
- Use as building block
- Partition input to smaller sets
- Also used in RecSplit [EGV20]



ShockHash

- Sample random graphs
- Store choice between two candidates [DHSW22a, LSW23b]

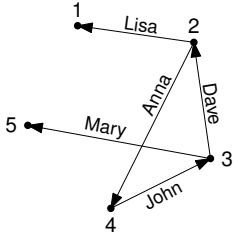
x	$h_0(x)$	$h_1(x)$
John	3	4
Lisa	2	1
Dave	2	3
Mary	5	3
Anna	2	4



ShockHash

- Sample random graphs
- Store choice between two candidates [DHSW22a, LSW23b]

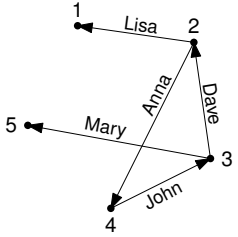
x	$h_0(x)$	$h_1(x)$
John	③	4
Lisa	2	①
Dave	②	3
Mary	⑤	3
Anna	2	④



ShockHash

- Sample random graphs
- Store choice between two candidates [DHSW22a, LSW23b]

x	$h_0(x)$	$h_1(x)$
John	③	4
Lisa	2	①
Dave	②	3
Mary	⑤	3
Anna	2	④

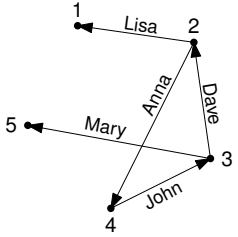


Retrieval	
John	0
Lisa	1
Dave	0
Mary	0
Anna	1

ShockHash

- Sample random graphs
- Store choice between two candidates [DHSW22a, LSW23b]

x	$h_0(x)$	$h_1(x)$
John	③	4
Lisa	2	①
Dave	②	3
Mary	⑤	3
Anna	2	④

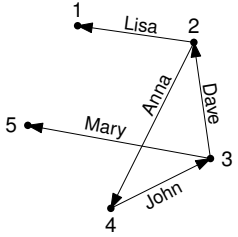


Retrieval	
John	0
Lisa	1
Dave	0
Mary	0
Anna	1

ShockHash

- Sample random graphs
- Store choice between two candidates [DHSW22a, LSW23b]

x	$h_0(x)$	$h_1(x)$
John	③	4
Lisa	2	①
Dave	②	3
Mary	⑤	3
Anna	2	④

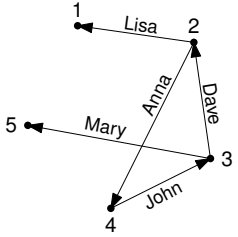


Retrieval	
John	0
Lisa	1
Dave	0
Mary	0
Anna	1

ShockHash

- Sample random graphs
- Store choice between two candidates [DHSW22a, LSW23b]
- Problem: Unlikely to work for $> n/2$ edges [PR04], here we use n

x	$h_0(x)$	$h_1(x)$
John	③	4
Lisa	2	①
Dave	②	3
Mary	⑤	3
Anna	2	④

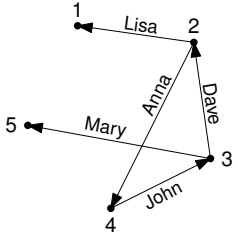


Retrieval	
John	0
Lisa	1
Dave	0
Mary	0
Anna	1

ShockHash

- Sample random graphs
- Store choice between two candidates [DHSW22a, LSW23b]
- Problem: Unlikely to work for $> n/2$ edges [PR04], here we use n
- **ShockHash**: Do it anyway, try many seeds

x	$h_0(x)$	$h_1(x)$
John	③	4
Lisa	2	①
Dave	②	3
Mary	⑤	3
Anna	2	④



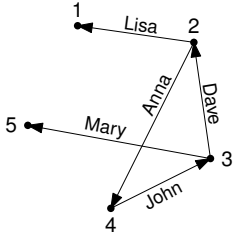
Retrieval	
John	0
Lisa	1
Dave	0
Mary	0
Anna	1

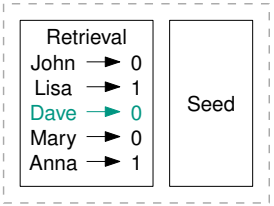
Seed

ShockHash

- Sample random graphs
- Store choice between two candidates [DHSW22a, LSW23b]
- Problem: Unlikely to work for $> n/2$ edges [PR04], here we use n
- **ShockHash**: Do it anyway, try many seeds

x	$h_0(x)$	$h_1(x)$
John	③	4
Lisa	2	①
Dave	②	3
Mary	⑤	3
Anna	2	④



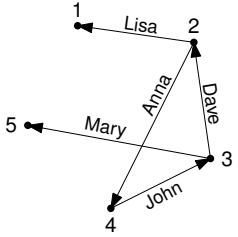


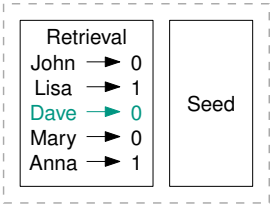
ShockHash data structure

ShockHash

- Sample random graphs
- Store choice between two candidates [DHSW22a, LSW23b]
- Problem: Unlikely to work for $> n/2$ edges [PR04], here we use n
- **ShockHash**: Do it anyway, try many seeds
- Orientability check?
  Success probability?
  Space usage?

x	$h_0(x)$	$h_1(x)$
John	③	4
Lisa	2	①
Dave	②	3
Mary	⑤	3
Anna	2	④

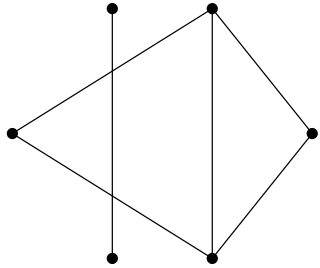




ShockHash data structure

Orientability Check

- 1-orientable if each component contains no more edges than nodes
  - Here: Tree with one additional edge
- Can be checked in linear time using connected components algorithms
- $\Rightarrow$ We check the 2^n different states of the retrieval data structure in linear time



Success Probability

$$\mathbb{P} \geq \frac{\qquad\qquad}{n^{2n}}$$

x	$h_0(x)$	$h_1(x)$
John	?	?
Lisa	?	?
Dave	?	?
Mary	?	?
Anna	?	?

Success Probability

$$\mathbb{P} \geq \frac{\qquad\qquad}{n^{2n}}$$

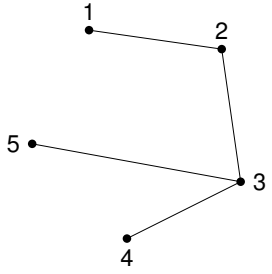
x	$h_0(x)$	$h_1(x)$
John	?	?
Lisa	?	?
Dave	?	?
Mary	?	?
Anna	?	?

Success Probability

- Labeled trees (Cayley's formula)

$$\mathbb{P} \geq \frac{n^{n-2}}{n^{2n}}$$

x	$h_0(x)$	$h_1(x)$
John	3	4
Lisa	2	1
Dave	2	3
Mary	5	3
Anna	?	?

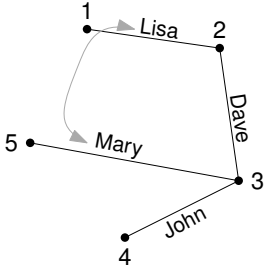


Success Probability

- Labeled trees (Cayley's formula)
- Table rows can be in any order

$$\mathbb{P} \geq \frac{n^{n-2}\ (n-1)!}{n^{2n}}$$

x	$h_0(x)$	$h_1(x)$
John	3	4
Lisa	2	1
Dave	2	3
Mary	5	3
Anna	?	?

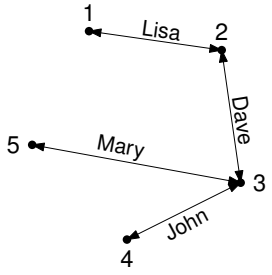


Success Probability

- Labeled trees (Cayley's formula)
- Table rows can be in any order
- Hash values can be in any order

$$\mathbb{P} \geq \frac{n^{n-2}\ (n-1)!\ 2^{n-1}}{n^{2n}}$$

x	$h_0(x)$	$h_1(x)$
John	3	4
Lisa	2	1
Dave	2	3
Mary	5	3
Anna	?	?

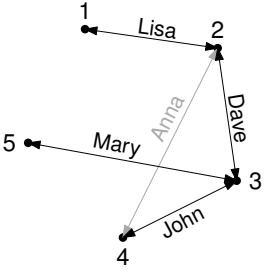


Success Probability

- Labeled trees (Cayley's formula)
- Table rows can be in any order
- Hash values can be in any order
- Last edge can be anything

$$\mathbb{P} \geq \frac{n^{n-2} \quad (n-1)! \quad 2^{n-1} \quad n^2}{n^{2n}}$$

x	$h_0(x)$	$h_1(x)$
John	3	4
Lisa	2	1
Dave	2	3
Mary	5	3
Anna	?	?



Success Probability

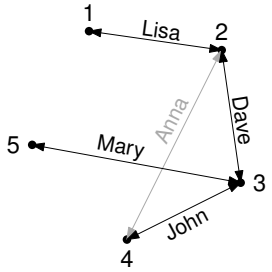
- Labeled trees (Cayley's formula)
- Table rows can be in any order
- Hash values can be in any order
- Last edge can be anything

$$\mathbb{P} \geq \frac{n^{n-2}\ (n-1)!\ 2^{n-1}\ n^2}{n^{2n}}$$

$$= \frac{n!}{n^n} \cdot \frac{2^{n-1}}{n}$$

Brute force

x	$h_0(x)$	$h_1(x)$
John	3	4
Lisa	2	1
Dave	2	3
Mary	5	3
Anna	?	?



Success Probability

- Labeled trees (Cayley's formula)
- Table rows can be in any order
- Hash values can be in any order
- Last edge can be anything

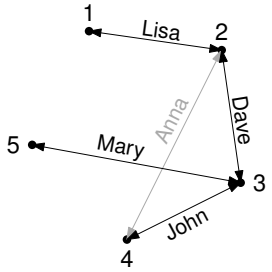
$$\mathbb{P} \geq \frac{n^{n-2} \;\; (n-1)! \;\; 2^{n-1} \;\; n^2}{n^{2n}}$$

$$= \frac{n!}{n^n} \cdot \frac{2^{n-1}}{n}$$

Brute force

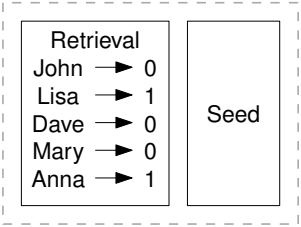
Almost 2^n times higher probability

x	$h_0(x)$	$h_1(x)$
John	3	4
Lisa	2	1
Dave	2	3
Mary	5	3
Anna	?	?



Space Usage

- Record choices in retrieval:
  BuRR [DHSW22b] solves equation system
  $f(x) = A \cdot h(x)$ to achieve $\approx n$ bits
- Expected space for seed:
  $\log_2\left(\frac{n}{2^{n-1}} \cdot \frac{n^n}{n!}\right) \approx n\log_2(e) - n$ bits
- Together: $\approx n\log_2(e)$ bits (optimal!)

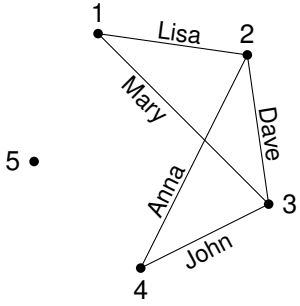


ShockHash data structure

Same space as brute-force but nearly 2^n times faster!

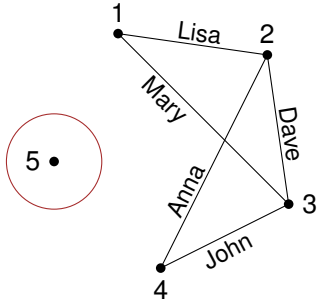
Filtering

- First implementation dominated by orientability check
  $\Rightarrow$Filter seeds that can't work
- Efficiently in registers
- Filter passed with probability only 0.84^n
- Main ingredient for making ShockHash feasible in practice



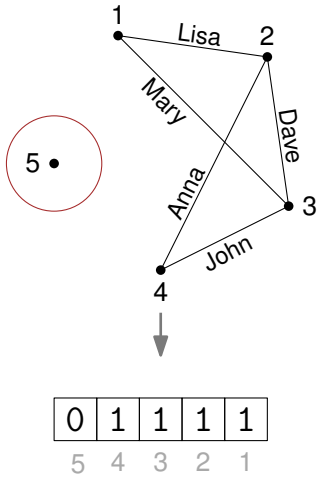
Filtering

- First implementation dominated by orientability check
  $\Rightarrow$Filter seeds that can't work
- Efficiently in registers
- Filter passed with probability only 0.84^n
- Main ingredient for making ShockHash feasible in practice

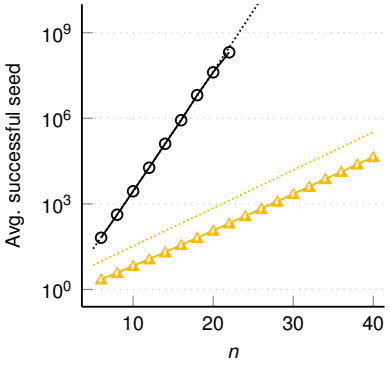


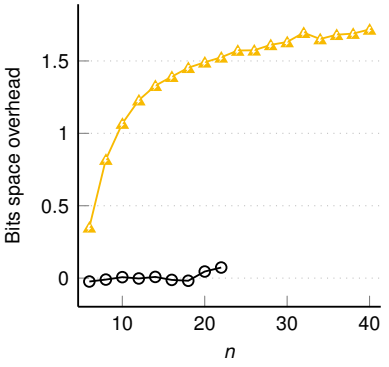
Filtering

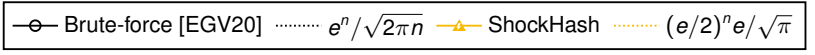
- First implementation dominated by orientability check
  $\Rightarrow$Filter seeds that can't work
- Efficiently in registers
- Filter passed with probability only 0.84^n
- Main ingredient for making ShockHash feasible in practice



Theory and Practice

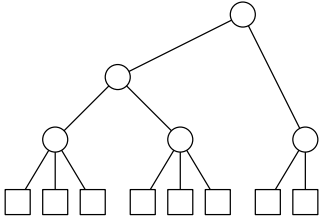




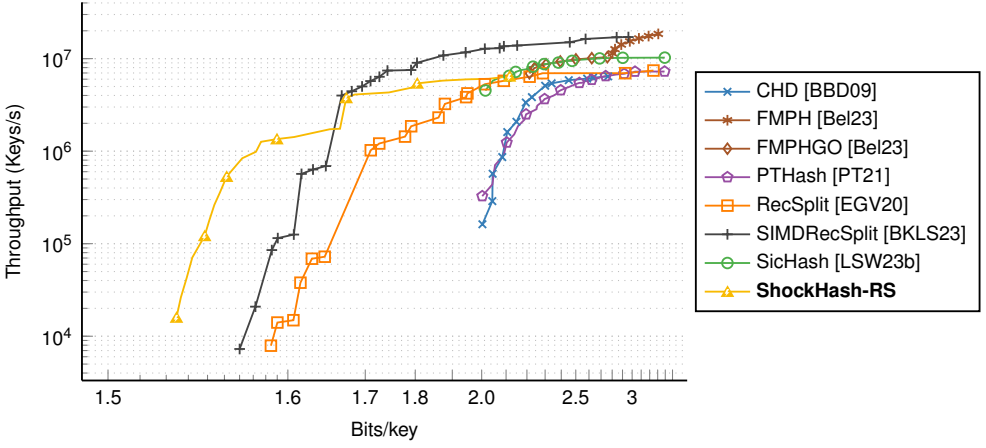


ShockHash-RS

- ShockHash as building block
- RecSplit [EGV20] for partitioning
  $\Rightarrow$ ShockHash-RS
- Practical building block size:
  16 (Brute-force) $\Rightarrow$ 50 (ShockHash)



Construction Throughput



Performance Overview

Method	Space	Constr.	Query
PTHash, c=7.0, α=0.99, C-C	3.524	198 ns	20 ns
SIMDRecSplit, n=14, b=2000	1.585	11 749 ns	108 ns
SIMDRecSplit, n=16, b=2000	1.560	137 902 ns	100 ns
ShockHash-RS, n=30, b=2000	1.583	787 ns	114 ns
ShockHash-RS, n=39, b=2000	1.556	1 805 ns	118 ns
ShockHash-RS, n=58, b=2000	1.523	111 593 ns	115 ns

Lower bound: 1.443 bits/key

Performance Overview

Method	Space	Constr.	Query
PTHash, c=7.0, α=0.99, C-C	3.524	198 ns	20 ns
SIMDRecSplit, n=14, b=2000	1.585	11 749 ns	108 ns
SIMDRecSplit, n=16, b=2000	1.560	137 902 ns	100 ns
ShockHash-RS, n=30, b=2000	1.583	787 ns	114 ns
ShockHash-RS, n=39, b=2000	1.556	1 805 ns	118 ns
ShockHash-RS, n=58, b=2000	1.523	111 593 ns	115 ns

Lower bound: 1.443 bits/key

Performance Overview

Method	Space	Constr.	Query
PTHash, c=7.0, α=0.99, C-C	3.524	198 ns	20 ns
SIMDRecSplit, n=14, b=2000	1.585	11 749 ns	108 ns
SIMDRecSplit, n=16, b=2000	1.560	137 902 ns	100 ns
ShockHash-RS, n=30, b=2000	1.583	787 ns	114 ns
ShockHash-RS, n=39, b=2000	1.556	1 805 ns	118 ns
ShockHash-RS, n=58, b=2000	1.523	111 593 ns	115 ns

Lower bound: 1.443 bits/key

Performance Overview

Method	Space	Constr.	Query
PTHash, c=7.0, α=0.99, C-C	3.524	198 ns	20 ns
SIMDRecSplit, n=14, b=2000	1.585	11 749 ns	108 ns
SIMDRecSplit, n=16, b=2000	1.560	137 902 ns	100 ns
ShockHash-RS, n=30, b=2000	1.583	787 ns	114 ns
ShockHash-RS, n=39, b=2000	1.556	1 805 ns	118 ns
ShockHash-RS, n=58, b=2000	1.523	111 593 ns	115 ns

Lower bound: 1.443 bits/key

Performance Overview

Method	Space	Constr.	Query
PTHash, c=7.0, α=0.99, C-C	3.524	198 ns	20 ns
SIMDRecSplit, n=14, b=2000	1.585	11 749 ns	108 ns
SIMDRecSplit, n=16, b=2000	1.560	137 902 ns	100 ns
ShockHash-RS, n=30, b=2000	1.583	787 ns	114 ns
ShockHash-RS, n=39, b=2000	1.556	1 805 ns	118 ns
ShockHash-RS, n=58, b=2000	1.523	111 593 ns	115 ns

Lower bound: 1.443 bits/key

Performance Overview

Method	Space	Constr.	Query
PTHash, c=7.0, α=0.99, C-C	3.524	198 ns	20 ns
SIMDRecSplit, n=14, b=2000	1.585	11 749 ns	108 ns
SIMDRecSplit, n=16, b=2000	1.560	137 902 ns	100 ns
ShockHash-RS, n=30, b=2000	1.583	787 ns	114 ns
ShockHash-RS, n=39, b=2000	1.556	1 805 ns	118 ns
ShockHash-RS, n=58, b=2000	1.523	111 593 ns	115 ns

Lower bound: 1.443 bits/key

Conclusion

- ShockHash is a building block for perfect hash functions
- Samples random graphs
- Optimal space consumption but 2^n construction speedup compared to brute-force
- Up to 76 times faster in practice
- /ByteHamster/ShockHash

- Future work
  - Improve query performance
  - Sample bipartite graphs [LSW23a]

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).

References I


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.


Piotr Beling.
Fingerprinting-based minimal perfect hashing revisited.
ACM Journal of Experimental Algorithmics, 2023.


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.

References II


Peter C. Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, and Stefan Walzer.
Fast succinct retrieval and approximate membership using ribbon.
In *SEA*, volume 233 of *LIPIcs*, pages 4:1–4:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.


Peter C. Dillinger, Lorenz Hübschle-Schneider, Peter Sanders, and Stefan Walzer.
Fast succinct retrieval and approximate membership using ribbon.
In *SEA*, volume 233 of *LIPIcs*, pages 4:1–4:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.


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

References III

- Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer.
  Bipartite ShockHash: Pruning ShockHash search for efficient perfect hashing.
  CoRR, abs/2310.14959, 2023.
- Hans-Peter Lehmann, Peter Sanders, and Stefan Walzer.
  Sichash - small irregular cuckoo tables for perfect hashing.
  In *ALENEX*, pages 176–189. SIAM, 2023.
- Rasmus Pagh and Flemming Friche Rodler.
  Cuckoo hashing.
  J. Algorithms, 51(2):122–144, 2004.
- Giulio Ermanno Pibiri and Roberto Trani.
  PTHash: Revisiting FCH minimal perfect hashing.
  In *SIGIR*, pages 1339–1348. ACM, 2021.