Experimental Variations of a Theoretically Good Retrieval Data Structure

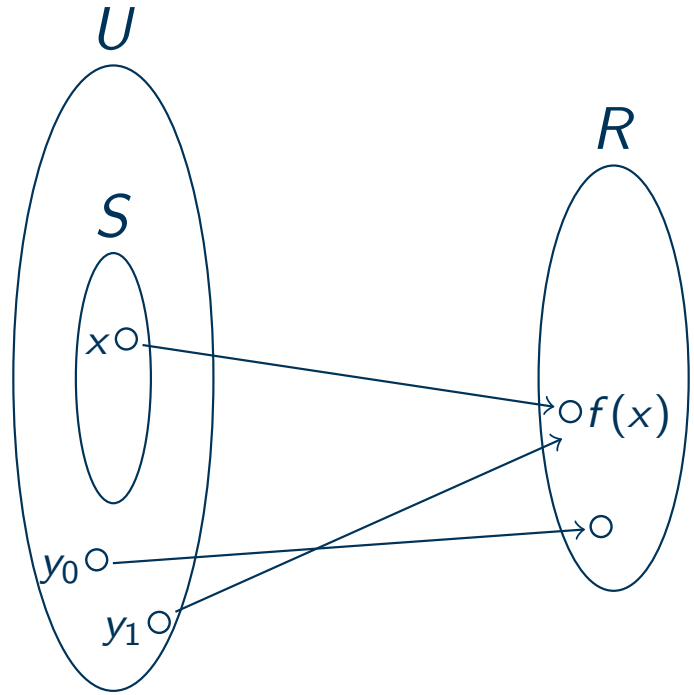
Martin Aumüller, Martin Dietzfelbinger, Michael Rink

Technische Universität Ilmenau

09.09.2009

Retrieval Problem

- given an universe U with a key set $S \subseteq U$ and some range R
- given a mapping $f : S \to R$
- Problem: provide a data structure (DS) that
  - on input $x \in S$ calculates $f(x)$
  - on input $y \in U - S$ calculates an arbitrary value from R



- Typical Solution: minimal perfect hashing (MPH)

Solutions (Theory)

- let $|S| = n$, $R = \{0,1\}^r$ and $n \gg \log\log|U|$
- Model: unit cost RAM, key fits in a single machine word
- $\mathcal{S}$: space [bits], $\mathcal{T}$: time [word ops]
- Solutions:

based on MPH	$\mathcal{S}$	$\mathcal{T}_{\text{eval}}$	$E(\mathcal{T}_{\text{con}})$
[HT01]	$1.44n + nr$	$O(1)$	$O(n)$
[BPZ07]	$3.62n$	$O(1)$	$O(n)$

some alternatives	$\mathcal{S}$	$\mathcal{T}_{\text{eval}}$	$E(\mathcal{T}_{\text{con}})$
[BPZ07]	$1.23n$	$O(1)$	$O(n)$
[DP08]	$(1+\varepsilon)n$	$O(\log(1/\varepsilon))$	$O(n)$
[Por09]	$nr + o(n)$	$O(1)$	$O(n)$

- Here: experiments to turn the approach from [DP08] into an operational DS
- Motivation: Can we beat the MPH approach for small r?

Retrieval Data Structure

- The retrieval data structure (RDS) from [DP08] uses 2 algorithms / sub-data structures:
  - PRIMARY [DP08]: builds a RDS with $\mathcal{S} = (1+\varepsilon)nr,\ \mathcal{T}_{\text{eval}} = O(\log(1/\varepsilon)),\ E(\mathcal{T}_{\text{con}}) = \Omega(n^2)$ only succeeds whp
  - BACKUP [BPZ07]: builds a RDS with $\mathcal{S} \approx 1.23nr, \quad \mathcal{T}_{\text{eval}} = O(1), \quad E(\mathcal{T}_{\text{con}}) = O(n)$
- Theory: Combining these two algorithms allows to build a RDS with asymptotic optimal space in expected linear time (details in a few moments).

PRIMARY Algorithm

- Goal: realize a given function $f : S \to \{0,1\}^r$
- Assume we have a function ξ that maps each $x \in S$ to a random binary vector $\boldsymbol{v}_x$ of weight k and length $m = (1+\varepsilon)n$.

$$\xi : x_0 \mapsto \begin{bmatrix}1 & 0 & 1 & 0 & 0 & 1\end{bmatrix} = \boldsymbol{v}_{x_0}$$

- Construct an $n \times m$ matrix $\boldsymbol{M}$ using the vectors $\boldsymbol{v}_x$ as rows.
- Build a vector $\boldsymbol{f}$ of the function values.
- Solve linear system $\boldsymbol{M} \cdot \boldsymbol{a} = \boldsymbol{f}$.

$$\overbrace{\begin{matrix} x_0: \\ x_1: \\ x_2: \\ x_3: \end{matrix}\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}}^{\boldsymbol{M}} \cdot \overbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}}^{\boldsymbol{a}} = \overbrace{\begin{bmatrix} f(x_0) \\ f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix}}^{\boldsymbol{f}}$$

PRIMARY Algorithm

- [Cal97]: $\boldsymbol{M}$ has full row rank whp if
  - ξ is fully random on S
  - $k \geq 3$, n sufficiently large, $m = (1+\varepsilon) \cdot n$
  - $\varepsilon > \varepsilon(k)$ where $\varepsilon(k) \to 0$ exponentially fast in k
- If a solution $\boldsymbol{a}$ exists, build an RDS for S by storing $\boldsymbol{a}$ and ξ.
- Retrieve $f(x)$ via XORing the entries of $\boldsymbol{a}$ which corresponds the non-zero entries of $\boldsymbol{v}_x$.

RDS: ξ,

a_0	a_1	a_2	a_3	a_4	a_5

evaluation: $x_0 \mapsto \boldsymbol{v}_{x_0} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$

return $f(x_0) = a_0 \oplus a_2 \oplus a_5$

PRIMARY Algorithm — Solution Strategy

- Repeat trying ξ's until $\boldsymbol{M}$ has full row rank.
  (failure probability: $O(n^{-c})$, for $c > 0$ arbitrary)
- Then Gauss-Jordan elimination yields a pseudoinverse $\boldsymbol{C}$ such that $\boldsymbol{C} \cdot \boldsymbol{M}$ has all unit vectors as columns.
- Since $\boldsymbol{C}$ is invertible it follows that each solution of:

$$(\boldsymbol{C} \cdot \boldsymbol{M}) \cdot \boldsymbol{a} = \boldsymbol{C} \cdot \boldsymbol{f} = \boldsymbol{g}$$

  is a solution of the original system $\boldsymbol{M} \cdot \boldsymbol{a} = \boldsymbol{f}$.

$$\overbrace{\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}}^{\boldsymbol{C} \cdot \boldsymbol{M}} \cdot \overbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}}^{\boldsymbol{a}} = \overbrace{\begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix}}^{\boldsymbol{g}} \Rightarrow \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} g_0 & +a_3 & +a_5 \\ g_3 & +a_3 & +a_5 \\ g_1 & +a_3 & \\ & a_3 & \\ g_2 & & +a_5 \\ & & a_5 \end{bmatrix}$$

- Only n entries of $\boldsymbol{a}$ are significant. $\Rightarrow$ Shrink solution vector to length n at the cost of $\lceil \log \binom{m}{n} \rceil$ extra bits.

Retrieval with Linear Construction Time

- Problem: Solving the large linear system needs too much time!
- Solution: Partition S into small subsets $S_i := \{x \in S \mid h'(x) = i\}$ using a hash function $h' : U \to [n/n']$.
- Apply PRIMARY algorithm to each S_i.
- If PRIMARY fails or S_i is "too large" then keys in S_i are handled via BACKUP algorithm.
- [DP08] suggest: $n' = E(|S_i|) = \frac{1}{2}\sqrt{\log n}$
  - A matrix fits in $O(1)$ words.
  - linear construction time (preprocessed lookup tables)
  - Size of tables, number of elements handled by BACKUP is $o(n)$.
  - Note: if $n' = \frac{1}{2}\sqrt{\log n} = 6$, then $n = 2^{144}$

Preliminary Considerations

- Q: How practical is it to use precomputed lookup tables, i.e. is there a chance to beat the $1.23nr$ bound given by BACKUP?
- $n' = E(|S_i|)$ small $\Rightarrow$ many of the subsets will be larger than n' $\Rightarrow$ keys go to BACKUP

Preliminary Considerations

- The number of $n' \times m'$, $m' = (1+\varepsilon)n'$, binary matrices with k ones per row is $\binom{m'}{k}^{n'}$.
  $\Rightarrow$ Very small ε lead to unrealistic large auxiliary tables.

  $$(n', m', k) = (7, 9, 3), \binom{m'}{k}^{n'} = 2.95 \cdot 10^{13}, (1+\varepsilon) = \tfrac{9}{7} > 1.28$$

- Way out: Compute pseudoinverses only during the construction of the RDS.
- Main task: Find suitable parameter configurations that optimize space and time requirements.

Small Matrices — Compression

- π: number of binary $n' \times m'$ matrices $\boldsymbol{M}$ with row weight k with a pseudoinverse $\boldsymbol{C}$
- Problem: π large even for small n'.

(n', m', k)	π	$\pi / n'!$
$(5, 6, 3)$	$1.54 \cdot 10^{6}$	$1.29 \cdot 10^{4}$
$(5, 7, 3)$	$3.58 \cdot 10^{7}$	$5.76 \cdot 10^{5}$
$(6, 7, 3)$	$8.78 \cdot 10^{8}$	$1.22 \cdot 10^{6}$
$(6, 8, 3)$	$2.04 \cdot 10^{10}$	$2.84 \cdot 10^{7}$

- Solution: "normalize" $\boldsymbol{M} \xrightarrow{\text{sort rows}} \boldsymbol{M}_{\text{norm}}$
- Store $(\boldsymbol{M}_{\text{norm}}, \boldsymbol{C})$ via hashing.
  (Lookup needs some pre- and postprocessing.)

Small Matrices — Space

- Even with compression, size $\mathcal{S}_{\mathrm{LT}}$ of lookup table becomes large for $n' \geq 6$.

(n', m', k)	$< \mathcal{S}_{\mathrm{LT}}$	$\pi / n'!$	p_{inv}
$(5, 6, 3)$	0.1MB	$1.29 \cdot 10^4$	0.482
$(5, 7, 3)$	2.7MB	$5.76 \cdot 10^5$	0.683
$(6, 7, 3)$	13.4MB	$1.22 \cdot 10^6$	0.478
$(6, 8, 3)$	312.4MB	$2.84 \cdot 10^7$	0.663

- Problem: The probability p_{inv} that a random (n', m', k)-matrix $\boldsymbol{M}$ has a pseudoinverse, varies.
- Solution: l functions $\xi_0, \xi_1, \ldots, \xi_{l-1}$ for rebuilding $\boldsymbol{M}$
- Space cost: $\lceil \log l \rceil \cdot \lceil n / n' \rceil$ bits

Small Matrices — Space and Time

- With $(n', m', k, l) = (5, 7, 3, 4)$ we get:

$$\mathcal{S} = 1.69nr + 7/5n \ .$$

  - The size of the lookup table is about 3 MB.
  - About 56 percent of the keys are handled via BACKUP.

- Avg construction time is $< 1.26 \cdot 10^{-5} \frac{s}{key}$ (3.2GHz Intel Xeon)

$$\bar{\mathcal{T}}_{\text{con}} \approx 2 \text{ min}, \ n = 10^7 \text{ keys}$$

- Using an additional lookup table for $(6, 7, 3)$-matrices we get:

$$\mathcal{S} = 1.69nr + n \ .$$

  - The sum of sizes of the lookup tables is about 16 MB.
  - About 40 percent of the keys are handled via BACKUP.

- Only slightly better is $(6, 8, 3)$, but has $\mathcal{S}_{\text{LT}} > 300$ MB.
- Limiting Factor: rapidly growing number of pseudoinverses

Large Matrices — Space

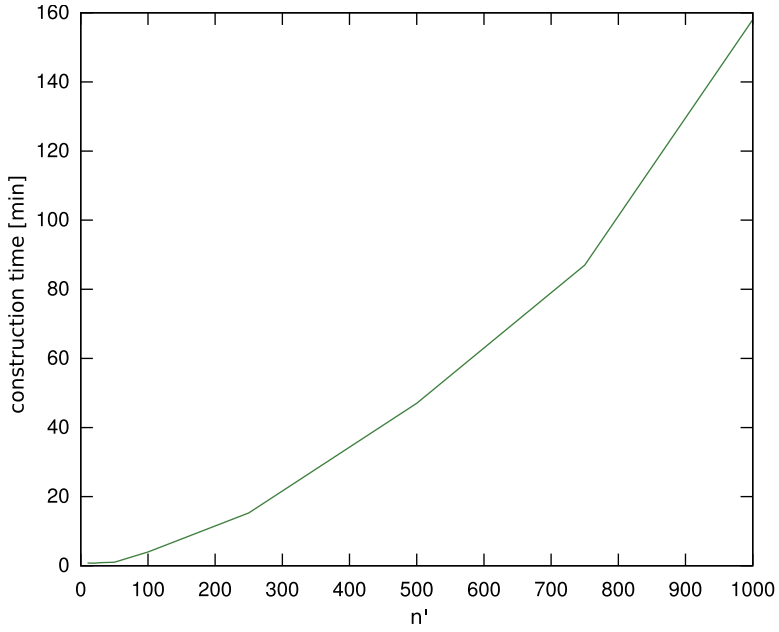
- Idea: Compute pseudoinverses during construction of the RDS.
- For n' from 10 up to 1000 we minimize $\mathcal{S}$ via local linear search.
- Some (local) optima for m':

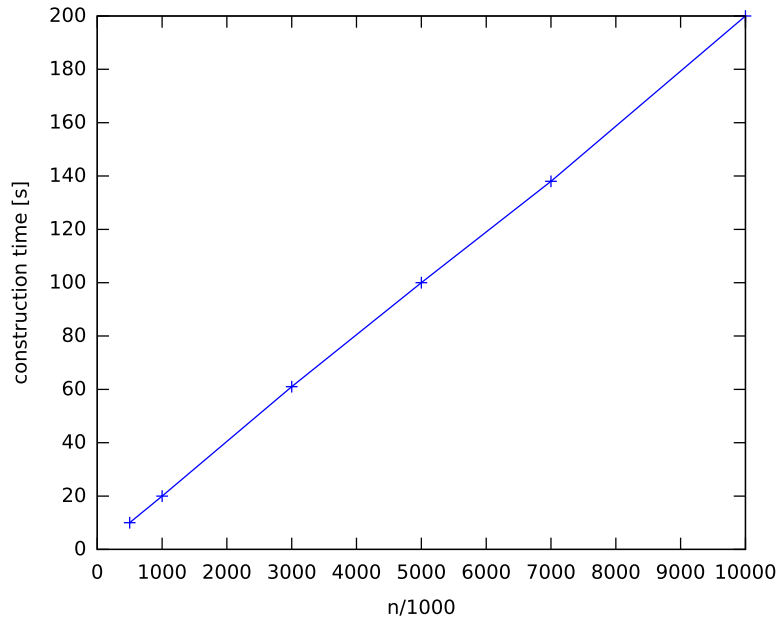
n'	m'	$\mathcal{S}$
20	26	$1.49nr$
50	61	$1.32nr$
100	119	$1.22nr$
200	230	$1.16nr$
500	540	$1.10nr$

  - inserting 7×10^7 keys
  - $k = 5, l = 2$
- beats the $1.23nr$ space bound for $n' \geq 100$ whp
- Note: getting above the standard word length

Large Matrices — Time

- Construction time for RDS $O(n \cdot (n')^2)$:





  - inserting 7×10^7 keys
  - $n' = 100$: $\bar{\mathcal{T}}_{\text{con}} < 4$ min
  - $n' \leq 250$: $\bar{\mathcal{T}}_{\text{con}} < 20$ min
  - inserting $7 \times n$ keys
  - $n' = 100, m' = 119$
- Reducing the overall space usage soon becomes expensive.

Conclusion

- We studied how to transfer the theoretically good RDS from [DP08] into a "real world" data structure.
- Especially we wanted to beat the $1.23nr$ space bound from [BPZ07].
- Result 1: The the theoretical approach with precomputed pseudoinverses cannot be used.
- Reason: Real input data and real machines are not large enough to make the asymptotic estimates valid.

Conclusion and Outlook

- Consequence: straightforward modification
  - Use the same idea of "splitting the key set".
  - Solve larger linear systems (a few hundred equations) without precomputation.
- Result 2: Makes it possible to build RDSs with $\approx 1.1nr$ bits, in time $E(\mathcal{T}_{\text{con}}) = O(n)$, for realistic key set sizes.
- Future Work: Use solution methods that utilize the sparse structure of the matrices better than Gaussian elimination.
  - e.g. Krylov subspace methods (Wiedemann, Lanczos, Conjugate-Gradient,...)
  - Problem: self-orthogonality in $\mathbb{F}_2$

Thank you!

References

- Fabiano C. Botelho, Rasmus Pagh, and Nivio Ziviani, *Simple and space-efficient minimal perfect hash functions*, Proc. 10th WADS, LNCS, vol. 4619, Springer-Verlag, 2007, pp. 139–150.
- Neil J. Calkin, *Dependent sets of constant weight binary vectors*, Combinatorics, Probability & Computing **6** (1997), no. 3, 263–271.
- Martin Dietzfelbinger and Rasmus Pagh, *Succinct data structures for retrieval and approximate membership (extended abstract)*, Proc. 35th ICALP(1), LNCS, vol. 5125, Springer-Verlag, 2008, pp. 385–396.
- Torben Hagerup and Torsten Tholey, *Efficient minimal perfect hashing in nearly minimal space*, Proc. 18th STACS, LNCS, vol. 2010, Springer-Verlag, 2001, pp. 317–326.
- Ely Porat, *An optimal bloom filter replacement based on matrix solving*, CSR, LNCS, vol. 5675, Springer-Verlag, 2009, pp. 263–273.