Making Deterministic Signatures Quickly

Milan Ružić
milan@itu.dk

Computational Logic and Algorithms Group
IT University of Copenhagen
Denmark
Basic searching problems

- Membership queries: Is \( x \in S \)?

- Predecessor queries: Find \( \max\{y \in S \mid y \leq x\} \).

- Data structure problem: How to store a set \( S \subset U \) and support inquires about membership (predecessor) plus retrieval of associated data.

- Static vs. dynamic.

- Some concepts that have given good solutions: trees, hashing.
Universe reduction

$U \supseteq S$

$U' \supseteq S'$

$U'' \supseteq S''$
In hashing-based reductions, typically there is a constant number of levels; the focus is on *computation*. Yet, it takes long to find an injective function deterministically.

In the structure of van Embde Boas universe reduction is gradual; the focus is on storing *information* and the steps of the reduction are *adaptive*. 
We give a new method of gradual universe reduction.

The focus is on (simple) computation – the functions have succinct description.

It does not take long to find an injective function for a given set $S$.

Every element of $S$ ultimately gets a unique signature of $O(\log n)$ bits.

Order is not preserved.
Basis of the method

Let $\phi(x) = x \text{ div } 2^s$, $\psi(x) = x \text{ mod } 2^s$.
It only matters that the function $(\phi, \psi)$ is 1-1 on $U$.

The basic reduction function:

$$f(x, a) = \phi(x) + a \cdot \psi(x)$$

Parameter $a$ is to be chosen from $\{1, 2, \ldots, n^c - 1\}$, $c \geq 2$.

The set of bad parameters $A_{bad} = \left\{ \frac{\phi(x_j) - \phi(x_i)}{\psi(x_i) - \psi(x_j)} \mid 1 \leq i < j \leq n \right\}$,
where $S = \{x_1, x_2, \ldots, x_n\}$. 
Choosing a good multiplier

We conduct a type of binary search:

Let $m = \left\{ \{i, j\} : \phi(x_i) < \phi(x_j) \land f(x_i, \mu) > f(x_j, \mu) \right\}$.

Then $m = A_{bad} \cap (0, \mu)$. 
The lower half is selected if \( m < n^2/4 \) (in the first step).

Because \( f \) is linear in \( a \), making a bisection decision on the interval \((l, r)\) can be reduced to making a decision on the interval \((0, r - l)\).

\( m \) can be exactly computed in a way similar to counting the number of inversions in a permutation; a little care has to be taken of duplicate values.

With the exact algorithm \( c \) may be set to 2.
Switching to a real permutation

Define orders $\prec_f$ and $\prec_\phi$ with:

$$x_i \prec_f x_j \iff f(x_i, \mu) < f(x_j, \mu) \lor (f(x_i, \mu) = f(x_j, \mu) \land \phi(x_i) < \phi(x_j)),$$

$$x_i \prec_\phi x_j \iff \phi(x_i) < \phi(x_j) \lor (\phi(x_i) = \phi(x_j) \land x_i \prec_f x_j),$$

and corresponding rank functions $\text{rank}_f, \text{rank}_\phi : S \to [n],$

$\text{rank}_f(x) = |\{x' \in S : x' \prec_f x\}|,$ and

$\text{rank}_\phi(x) = |\{x' \in S : x' \prec_\phi x\}|.$

Let $\pi$ be and a permutation of $[n]$ such that

$\pi(i) = \text{rank}_f(\text{rank}_\phi^{-1}(i)).$

Then, $m = \text{Inv}(\pi).$
We use an approximate formula for inversions:

\[
Inv(\pi) \leq \sum_{i=0}^{n-1} |\pi(i) - i| \leq 2Inv(\pi).
\]

The computation can be parallelized on the word level. The algorithm for finding a good multiplier \(a\) runs in time

\[
O \left( n \log n \left( \frac{1}{K} + \min \left( \log \log n, \frac{\log n \log K}{K} \right) \right) + \log n \right),
\]

where \(K\) is the number of keys that can be packed in a machine word.

Here \(c\) may be set to 3.42.
Top-down composition

With the top-down approach, we use a composition of function $f$ with different parameters for each level of the reduction.

Evaluation is efficient for strings.

In the RAM models it allows batch evaluation on integer arguments; the average time is constant for batches of size $\Omega(\log \frac{w}{\log n})$.

Construction takes time $O(n \log n (\log \log n)^2 + \log n \log w)$ in the word RAM model.

The range of the signatures can be set to $[n^{3.5}]$. 
At most $2n - 2$ edges out of branching nodes.
Results for string approach – RAM

- Signature function can be constructed in time $O(n \log \log n + n \frac{\log^3 n}{w} \log^3 w)$.

- It is possible to evaluate the function in time $O(\log \log \frac{w}{\log n})$.

- This leads to a static dictionary with a lookup time of $O(\log \log n)$ and construction time of $O(n \log \log n)$. 
Results for string approach – I/O

The function description can be computed in I/O cost proportional to the cost of sorting \( \frac{N}{\log n} \) integers of size \( \log n \) bits plus

\[
O \left( \frac{\log^2 n \cdot \log_{M/B}(n/B)}{B} \sum_{s \in S} \log \frac{|s|}{\log n} \right) \text{ I/Os,}
\]

where \( B \) and \( M \) are expressed in bits.

Evaluation of the function on argument \( s \) requires \( O\left(\frac{|s|}{B}\right) \) I/Os.

The bound on the performance of the construction holds under a tall-cache assumption of type \( M > B^{1+\delta} \). The evaluation procedure needs no tall-cache assumption.
Predecessor problem

Query performance in the cache-oblivious model:

\[ O\left(\frac{|s|}{B} + \log |s| + \log \log n\right) \text{ I/Os.} \]
Future work

- Efficient dictionary for polynomial-size universes in the cache-oblivious model.

- Bridging the gap to sorting complexity for strings of length between $\Omega(\log n)$ to $O(\log^{2+\epsilon} n)$ bits in the cache-oblivious model.

- Dynamization.