Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 155 (2007) 386–393
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

Decompositions of graphs of functions and fast iterations of lookup tables☆

Boaz Tsaban

Department of Mathematics, The Weizmann Institute of Science, Rehovot 76100, Israel

Received 9 October 2005; received in revised form 27 June 2006; accepted 29 June 2006
Available online 17 August 2006

Abstract

We show that every function f implemented as a lookup table can be implemented such that the computational complexity of evaluating $f^m(x)$ is small, independently of m and x. The implementation only increases the storage space by a small *constant* factor.

Keywords: Fast forward functions; Fast forward permutations; Cycle decomposition; Cycle structure

1. Introduction and motivation

According to Naor and Reingold [2], a function $f : \{0, \ldots, N-1\} \to \{0, \ldots, N-1\}$ is *fast forward* if for each natural number m which is polynomial in N, and each $x = 0, \ldots, N-1$, the computational complexity of evaluating $f^m(x)$—the mth iterate of f at x—is small (polynomial in $\log N$). This is useful in simulations and cryptographic applications, and for the study of dynamic-theoretic properties of the function f.

Originally this notion was studied in the context of pseudorandomness, where N is very large—see [2,3,1]. Here we consider the remainder of the scale, where N is not too large, so that the function $f : \{0, \ldots, N-1\} \to \{0, \ldots, N-1\}$ is or can be implemented by a lookup table of size N. Implementations as lookup tables are standard for several reasons, e.g., in the case where the evaluation $f(x)$ is required to be efficient, or in the case that f is a random function, so that f has no shorter definition than just specifying its values for all possible inputs. We describe a simple way to implement a given function f such that it becomes fast forward. The implementation only increases the storage space by a small constant factor.

The case that f is a permutation is of special importance and is easier to treat. This is done in Section 2. In Section 3 we treat the general case.

2. Making a permutation fast forward

We recall two definitions from [3].

☆ Supported by the Koshland Fellowship.
E-mail address: boaz.tsaban@weizmann.ac.il
URL: http://www.cs.biu.ac.il/∼ tsaban.

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2006.06.006

Definition 1. Assume that f is a permutation on $\{0,\dots,N-1\}$. The *ordered cycle decomposition* of f is the sequence $(C_0,\dots,C_{\ell-1})$ consisting of all (distinct) cycles of f, such that for each $i, j \in \{0,\dots,\ell-1\}$ with $i<j$, $\min C_i < \min C_j$. The *ordered cycle structure* of f is the sequence $(|C_0|,\dots,|C_{\ell-1}|)$.

The ordered cycle decomposition of f can be computed in time N: find C_0, the cycle of 0. Then find C_1, the cycle of the first element not in C_0, etc. In particular, the ordered cycle *structure* of f can be computed in time N.

Definition 2. Assume that $(m_0, m_1, \dots, m_{\ell-1})$ is the ordered cycle structure of a permutation f on $\{0,\dots,N-1\}$. For each $i=0,\dots,\ell-1$, let $s_i = m_0+\cdots+m_i$. The *fast forward permutation coded by* $(m_0, m_1, \dots, m_{\ell-1})$ is the permutation π on $\{0,\dots,N-1\}$ such that for each $x \in \{0,\dots,N-1\}$,

$$\pi(x) = s_i + (x - s_i + 1 \bmod m_{i+1}) \quad \text{where } s_i \leqslant x < s_{i+1}.$$

In other words, π is the permutation whose ordered cycle decomposition is

$$\pi = (\underbrace{0\dots s_0-1}_{m_0})(\underbrace{s_0\dots s_1-1}_{m_1})(\underbrace{s_1\dots s_2-1}_{m_2})\cdots(\underbrace{s_{\ell-2}\dots N-1}_{m_{\ell-1}}).$$

The assignment $x \mapsto i(x)$ such that $s_{i(x)} \leqslant x < s_{i(x)+1}$ can be implemented (in time N) as a lookup table of size N. As

$$\pi^m(x) = s_{i(x)} + (x - s_{i(x)} + m \bmod (s_{i(x)+1} - s_{i(x)})),$$

π is fast forward.

Coding 3. To code a given permutation f on $\{0,\dots,N-1\}$ as a fast forward permutation, do the following.

(1) Compute the ordered cycle decomposition of f:

$$f = (\underbrace{b_0\dots b_{s_0-1}}_{m_0})(\underbrace{b_{s_0}\dots b_{s_1-1}}_{m_1})(\underbrace{b_{s_1}\dots b_{s_2-1}}_{m_2})\cdots(\underbrace{b_{s_{\ell-2}}\dots b_{N-1}}_{m_{\ell-1}}).$$

(2) Define a permutation σ on $\{0,\dots,N-1\}$ by $\sigma(x)=b_x$ for each $x=0,\dots,N-1$.
(3) Store in memory the following tables: σ, σ^{-1}, the list $s_0,\dots,s_{\ell-1}$ (where $s_k = m_0+\cdots+m_k$ for each k), and the assignment $x \mapsto i(x)$.

Let π be the fast forward permutation coded by $(m_0, m_1, \dots, m_{\ell-1})$. Then

$$f = \sigma \circ \pi \circ \sigma^{-1}.$$

For each m and x, $f^m(x)$ is equal to $\sigma(\pi^m(\sigma^{-1}(x)))$, which is computed by five invocations of the stored lookup tables and five elementary arithmetic operations (addition, subtraction, or modular reduction). We therefore have the following.

Theorem 4. *Every permutation f on $\{0,\dots,N-1\}$ can be coded by four lookup tables of size N each, such that each evaluation $f^m(x)$ can be carried using five invocations of lookup tables and five elementary arithmetic operations, independently of the size of m.*

Remark 5.

(1) For random permutations, $\ell \approx \log N$ and therefore the total amount of memory is about $3N + \log N$.
(2) Instead of storing the assignment $x \mapsto i(x)$, we can compute it online. This is a search in an ordered list and takes $\log_2(\ell)$ in the worst case. For a typical permutation this is about $\log_2(\log(N))$ additional operations in the worst case (e.g., for $N=2^{32}$, this is about four additional operations per evaluation). This reduces the memory to $2N + \log N$.

3. Making an arbitrary function fast forward

We begin with a simple method, and then describe a twist of this method which gives better results.

3.1. The basic approach

The language of graphs will be convenient. For shortness, a (partial) function $f : \{0, \ldots, N-1\} \to \{0, \ldots, N-1\}$ will be called a (*partial*) *function on* $\{0, \ldots, N-1\}$.

Definition 6. Let f be a partial function on $\{0, \ldots, N-1\}$. The *graph of* f is the directed graph $G = \langle V, E\rangle$, where

$$V = \{0, \ldots, N-1\},$$

$$E = \{(x, f(x)) : x \in \mathrm{dom}(f)\}.$$

The *orbit* of an element $v \in V$ is the *maximal simple* tour $(v, v_1, v_2, \ldots, v_k)$ in G. Note that either $f(v_k)$ is undefined, or else $f(v_k) \in \{v, v_1, v_2, \ldots, v_k\}$. In the latter case, we say that the orbit is a *ρ-orbit*.

Any subgraph of a partial function f on $\{0, \ldots, N-1\}$ is the graph of some restriction of f, and in particular is the graph of some partial function g on $\{0, \ldots, N-1\}$.

Definition 7. Assume that f is a function on $\{0, \ldots, N-1\}$. The *ordered orbit decomposition* of f is the sequence $(C_0, \ldots, C_{\ell-1})$ defined by

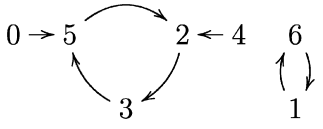
(1) C_0 is the orbit of 0.
(2) For $k > 0$, if $V \neq C_0 \cup C_1 \cup \cdots \cup C_{k-1}$, then C_k is the orbit of the least element of $V \setminus (C_0 \cup \cdots \cup C_{k-1})$ in the subgraph induced by G on the vertices in $V \setminus (C_0 \cup \cdots \cup C_{k-1})$.
(3) ℓ is the least k such that $V = C_0 \cup \cdots \cup C_{k-1}$.

The *ordered orbit structure* of f is the sequence $(|C_0|, \ldots, |C_{\ell-1}|)$.

Note that the ordered orbit decomposition of a permutation is just its ordered cycle decomposition. Assume that $(C_0, \ldots, C_{\ell-1})$ is the ordered orbit decomposition of f. Clearly, $(C_0, \ldots, C_{\ell-1})$ can be reconstructed from the concatenated sequence $C_0C_1 \cdots C_{\ell-1}$ together with the ordered orbit structure $(|C_0|, \ldots, |C_{\ell-1}|)$ of f. To reconstruct f from $(C_0, \ldots, C_{\ell-1})$, we need in addition the following information.

Definition 8. The *auxiliary sequence* for an ordered orbit decomposition $(C_0, \ldots, C_{\ell-1})$ of a function f is $(p_0, \ldots, p_{\ell-1})$, where for each $i = 0, \ldots, \ell-1$, p_i is the position of $f(v_i)$ in the concatenated sequence $C_0C_1 \ldots C_{\ell-1}$, v_i being the last element in the sequence C_i.

Example 9. Consider the function f on $\{0, \ldots, 6\}$ whose graph is



The ordered orbit decomposition of f is

$$(C_0, C_1, C_2) = ((0, 5, 2, 3), (1, 6), (4)),$$

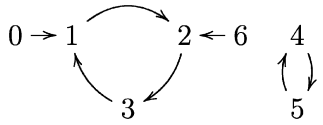
and the ordered orbit structure is $(|C_0|, |C_1|, |C_2|) = (4, 2, 1)$. C_0 and C_1 are ρ-orbits, whereas C_2 is not. The concatenated orbits $C_0C_1C_2$ give $(0, 5, 2, 3, 1, 6, 4)$. Now, 3 is the last element in C_0, and the position of $f(3) = 5$ in the concatenated sequence is 1. 6 is the last element in C_1, and the position of $f(6) = 1$ in the concatenated sequence is 4. Similarly, the position of $f(4) = 2$ is 2, so the auxiliary sequence is $(1, 4, 2)$.

Definition 10. Assume that $(m_0, m_1, \ldots, m_{\ell-1})$ is the ordered orbit structure of a function f on $\{0, \ldots, N-1\}$, and that the auxiliary sequence is $(p_0, \ldots, p_{\ell-1})$. For each $i = 0, \ldots, \ell-1$, let $s_i = m_0 + \cdots + m_i$. The *fast forward function coded by* $(m_0, m_1, \ldots, m_{\ell-1})$ *and* $(p_0, \ldots, p_{\ell-1})$ is the function $\pi : \{0, \ldots, N-1\} \to \{0, \ldots, N-1\}$ whose ordered orbit decomposition is

$$((\underbrace{0 \ldots s_0 - 1}_{m_0}), (\underbrace{s_0 \ldots s_1 - 1}_{m_1}), (\underbrace{s_1 \ldots s_2 - 1}_{m_2}), \ldots, (\underbrace{s_{\ell-2} \ldots N-1}_{m_{\ell-1}})),$$

and whose auxiliary sequence is $(p_0, \ldots, p_{\ell-1})$.

Example 11. The ordered orbit structure of f in Example 9 is (4, 2, 1), and the auxiliary sequence is (1, 4, 2). The fast forward function π corresponding to f is that with the same auxiliary sequence and whose ordered orbit decomposition is ((0, 1, 2, 3), (4, 5), (6)). The graph of π is



Using the auxiliary sequence we have, e.g., that

$$\pi^{10}(6) = \pi^9(2) = \pi^7(1) = 1 + (7 \bmod 3) = 2,$$

as can be verified directly.

Example 11 hints to the following recursive procedure to compute $\pi^m(x)$. Again, let $i(x)$ be such that $s_{i(x)} \leqslant x < s_{i(x)+1}$ for each $x = 0, \ldots, N-1$.

(1) Let $r = m - (s_{i(x)+1} - x)$. (Note that $r < m$.)
(2) If $r < 0$, then $\pi^m(x) = x + m$.
(3) Else:
   (a) If $s_{i(x)} \leqslant p_{i(x)}$ then $C_{i(x)}$ is a ρ-orbit, and therefore

$$\pi^m(x) = p_{i(x)} + (r \bmod (s_{i(x)+1} - p_{i(x)})).$$

   (b) Otherwise, $\pi^m(x) = \pi^r(p_{i(x)})$.

Case (b) is the only case where a recursion is made. Note that in this case, $p_{i(x)} < s_{i(x)}$, i.e. we descend to a previous component. We therefore call this case a *descent*.

For simplicity, use the term *basic operation* for either a basic arithmetic operation, a comparison, or a lookup access. It follows that each descent requires less than 10 basic operations.

Corollary 12. *The complexity of evaluating* $\pi^m(x)$ *is a constant* $c \leqslant 10$ *times the number of descents needed until a* ρ*-orbit is reached.*

Remark 13. In the sequel, we will measure the complexity by the number of descents. The constant c by which this should be multiplied (Corollary 12) can be made smaller by pre-computing lookup tables for $s_{i(x)}$, $p_{i(x)}$, and $s_{i(x)+1} - p_{i(x)}$.

We now describe the basic method for coding f as a fast forward function. The running time of this transformation is a small constant multiple of N.

Coding 14. Assume that f is a function on $\{0, \ldots, N-1\}$. Code f as follows.

(1) Compute the ordered orbit decomposition of f:

$$((\underbrace{b_0 \ldots b_{s_0-1}}_{m_0}), (\underbrace{b_{s_0} \ldots b_{s_1-1}}_{m_1}), (\underbrace{b_{s_1} \ldots b_{s_2-1}}_{m_2}), \ldots, (\underbrace{b_{s_{\ell-2}} \ldots b_{N-1}}_{m_{\ell-1}})).$$

(2) Define a permutation σ on $\{0, \ldots, N-1\}$ by $\sigma(x) = b_x$ for each $x = 0, \ldots, N-1$.
(3) Use σ^{-1} to compute the auxiliary sequence $(p_0, \ldots, p_{\ell-1})$.
(4) Store in memory the following tables: σ, σ^{-1}, the list $s_0, \ldots, s_{\ell-1}$ (where $s_k = m_0 + \cdots + m_k$ for each k), the auxiliary sequence $(p_0, \ldots, p_{\ell-1})$, and the assignment $x \mapsto i(x)$ (such that each $x \in C_{i(x)}$).

Note that the code of f defines the fast forward function π coded by $(m_0, \ldots, m_{\ell-1})$ and $(p_0, \ldots, p_{\ell-1})$, and that $f = \sigma \circ \pi \circ \sigma^{-1}$. Thus,

$$f^m(x) = \sigma(\pi^m(\sigma^{-1}(x))$$

for each x and m. Consequently, if the maximal number of descents in π is small, $f^m(x)$ can be evaluated efficiently for all m and x.

Simulations show that for random functions f, the maximal number of descents in the evaluations $f^m(x)$ is around $\log_2 N$. We will give concrete results for a better approach in the sequel.

3.2. An improved approach

There are pathological cases where the number of descents can be N. We exhibit the extreme case, with a hint concerning how it can be avoided.

Example 15. Consider the function $f(k) = \max\{0, k-1\}$:

$$\circlearrowright 0 \leftarrow 1 \leftarrow 2 \leftarrow \cdots \leftarrow (N-2) \leftarrow (N-1)$$

The ordered orbit decomposition of f is $((0), (1), (2), \ldots, (N-1))$, and the auxiliary sequence is $(0, 0, 1, 2, \ldots, N-2)$. The ordered orbit structure is $(1, 1, \ldots, 1)$, and the corresponding fast forward function π is equal to f. Computing $\pi^m(N-1)$ for $m \geqslant N-1$ requires $N-1$ descents.

Now consider the function $g(k) = \min\{k+1, N-1\}$:

$$\circlearrowright (N-1) \leftarrow (N-2) \leftarrow \cdots \leftarrow 2 \leftarrow 1 \leftarrow 0$$

The ordered orbit decomposition of g is $((0, 1, 2, \ldots, N-1))$, and the auxiliary sequence is (0). The ordered orbit structure is (N), and the corresponding fast forward function π is equal to g. No descents at all are required to compute values $\pi^m(x)$.

The following definition captures the improvement made in the second part of the last example.

Definition 16. Assume that f is a function on $\{0, \ldots, N-1\}$. The *greedy orbit decomposition* of f is the sequence $(C_0, \ldots, C_{\ell-1})$ defined as follows, where a *maximal* orbit is an orbit of maximal length, and when there is more than one maximal orbit, we choose the one starting with the least point:

(1) C_0 is the maximal orbit in G.
(2) For $k > 0$, if $V \neq C_0 \cup C_1 \cup \cdots \cup C_{k-1}$, then C_k is the maximal orbit in the subgraph induced by G on the vertices in $V \setminus (C_0 \cup \cdots \cup C_{k-1})$.
(3) ℓ is the least k such that $V = C_0 \cup \cdots \cup C_{k-1}$.

The *greedy orbit structure* of f is the sequence $(|C_0|, \ldots, |C_{\ell-1}|)$.

Remark 17. Given a graph of a function on $\{0, \ldots, N-1\}$, one can attach to each vertex the length of its orbit. This can be done in $\leqslant 2N$ steps. After removing an orbit from the graph, only the points which eventually enter the orbit need to be modified. Even if we recompute all lengths after each removal of an orbit, the overall complexity is not more (and usually much less) than

$$2N + 2(N-1) + \cdots + 2 \approx N^2.$$

Since the procedure is done only once and offline, we do not try to optimize further.

Having defined the greedy orbit decomposition of f, we can proceed to define, with respect to it, the auxiliary sequence and the other definitions, as well as the coding, exactly as in Section 3.1.

Example 18. Notation as in Example 15, we have that the greedy orbit decomposition of f is $((N-1, N-2, \ldots, 1, 0))$, the auxiliary sequence is $(N-1)$, and the ordered orbit structure is (N). The fast forward function π is equal to g, and no descents at all are required to compute values $\pi^m(x)$.

The following theorem shows that, using the greedy orbit structure, the maximal possible number of descents cannot be greater than about $\sqrt{2N}$.

Theorem 19. *Assume that f is a function on $\{0, \ldots, N-1\}$. Then the maximal number of descents in the greedy orbit structure of f is not greater than $\lfloor(\sqrt{1+8N}-3)/2\rfloor$.*

Proof. Consider the greedy orbit structure $(C_0, \ldots, C_{\ell-1})$ and auxiliary sequence $(p_0, \ldots, p_{\ell-1})$ for f. Let d be the maximal number of descents in this structure. Then there is a sequence $i_0 < i_1 < \cdots < i_d$ such that for each $j = 1, \ldots, d$, the last member in C_{i_j} is mapped by f to some member of $C_{i_{j-1}}$. Since $(C_0, \ldots, C_{\ell-1})$ is a greedy orbit structure, we have that

$$|C_{i_0}| > |C_{i_1}| > \cdots > |C_{i_d}|.$$

Indeed, for each $j = 1, \ldots, d$, as C_{i_j} is not a ρ-orbit, the orbit in $\langle V \setminus (C_0 \cup \cdots \cup C_{i_{j-1}-1}), E\rangle$ starting with the first element of C_{i_j} is of size at least $|C_{i_j}| + 1$, and by the maximality of $|C_{i_{j-1}}|$, we have that $|C_{i_j}| + 1 \leqslant |C_{i_{j-1}}|$.

Consequently, for each $j = 0, \ldots, d$, $|C_{i_j}| \geqslant d - j + 1$, and therefore

$$N = |V| \geqslant \left|\bigcup_{j=0}^{d} C_{i_j}\right| = \sum_{j=0}^{d} |C_{i_j}| \geqslant \sum_{j=0}^{d} (j+1) = \frac{(d+1)(d+2)}{2}.$$

Thus, $d^2 + 3d + (2-2N) \leqslant 0$, that is,

$$d \leqslant \frac{-3 + \sqrt{9 - 4(2-2N)}}{2} = \frac{\sqrt{1+8N}-3}{2}. \qquad \square$$

The bound in Theorem 19 cannot be improved.

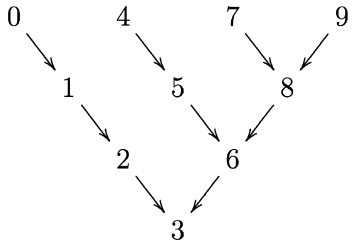
Example 20. Fix N. Let $d = \lfloor(\sqrt{1+8N}-3)/2\rfloor$. Then $M = (d+1)(d+2)/2 \leqslant N$. We will define a function on $\{0, \ldots, M-1\}$ whose greedy orbit decomposition has d descents starting at $M-1$. Clearly, such a function can be extended to a function on $\{0, \ldots, N-1\}$ with d descents in its greedy orbit decomposition by extending the first component.

Consider the function f whose greedy orbit decomposition is

$$((0, 1, \ldots, d), \ldots, (M-6, M-5, M-4), (M-3, M-2), (M-1))$$

with auxiliary sequence $(d, d, \ldots, M-4, M-2)$. There are $d+1$ components, and each component is descended into the previous component, so starting at the value $M-1$ we have d many descents.

E.g., for $d = 3$, $M = 10$ and the function is



its greedy orbit decomposition is ((0, 1, 2, 3), (4, 5, 6), (7, 8), (9)) and the auxiliary sequence is (3, 3, 6, 8). Computing $f^3(9)$ requires three descents.

Note that Example 20 has an orbit decomposition with at most one descent. E.g., in the case $d = 3$ we can take ((9, 8, 6, 3), (0, 1, 2), (4, 5), (7)) with auxiliary sequence (3, 3, 2, 1).

This suggests that in Definition 16, when we have more than one maximal orbit, we should try all possibilities. This way, the algorithm becomes exponential. We have tried a randomized approach which broke ties using coin flips. It did not give significantly better results. We would be glad but surprised if the answer to the following would turn out positive.

Problem 21. *Does there exist an efficient algorithm to find, for a given function f on* $\{0, \ldots, N-1\}$, *an orbit decomposition for which the maximal number of descents is as small as it can be for f*?

3.3. The random case

The random case, and presumably most of the cases encountered in practice, behaves much better than is provable for the worst case. For each $N = 2^2, 2^3, \ldots, 2^{20}$, we have sampled 100 random functions on $\{0, \ldots, N-1\}$. For these, we have computed the maximum and average number of descents. The results appear in Fig. 1 contains three increasing and one decreasing graphs. Among the increasing graphs, the uppermost is just $\log_2 N$, the intermediate graph is the maximum number of descents encountered for each N, and the lowest is the average number of descents. The decreasing graph is $\log_2 N$ divided by the average number of descents.

An interesting observation is that none of the samples contained a point with more than $\log_2 N$ many descents. This should be contrasted with Example 20, and suggests that the cases in which the complexity of evaluating $f^m(x)$ can be larger than about $\log_2 N$ are indeed pathological.

Another observation, which is of great practical interest, is supplied by the decreasing graph: it shows that for the checked values of N, and presumably for all practical values of N, the average number of descents is about $(\log_2 N)/5$ or less. Recall from Corollary 12 and the remark after it, that the overall complexity is a small multiple of this number.

We conclude the paper by demonstrating that the mere consideration of average complexity rather than maximal complexity does not suffice to obtain the logarithmic phenomenon which we encountered in the random case.

Example 22. Consider the function f described in Example 20, and assume that $(\sqrt{1+8N}-3)/2$ is an integer (otherwise the following is only approximate). In this case, d is equal to this number, and $(d+1)(d+2) = 2N$.

The average number of descents for f is $1/N$ times

$$\begin{aligned}
&1 \cdot d + 2 \cdot (d-1) + \cdots + d \cdot 1 \\
&\quad = \sum_{i=1}^{d} i(d+1-i) = \sum_{i=1}^{d} i(d+1) - \sum_{i=1}^{d} i^2 \\
&\quad = \frac{(d+1)(1+d)d}{2} - \frac{d(d+1)(2d+1)}{6} \\
&\quad = \frac{d(d+1)(d+2)}{6} = N \cdot \frac{d}{3}.
\end{aligned}$$

Thus, the average number of descents in f is $d/3$ (which is roughly $\sqrt{2N}/3$).

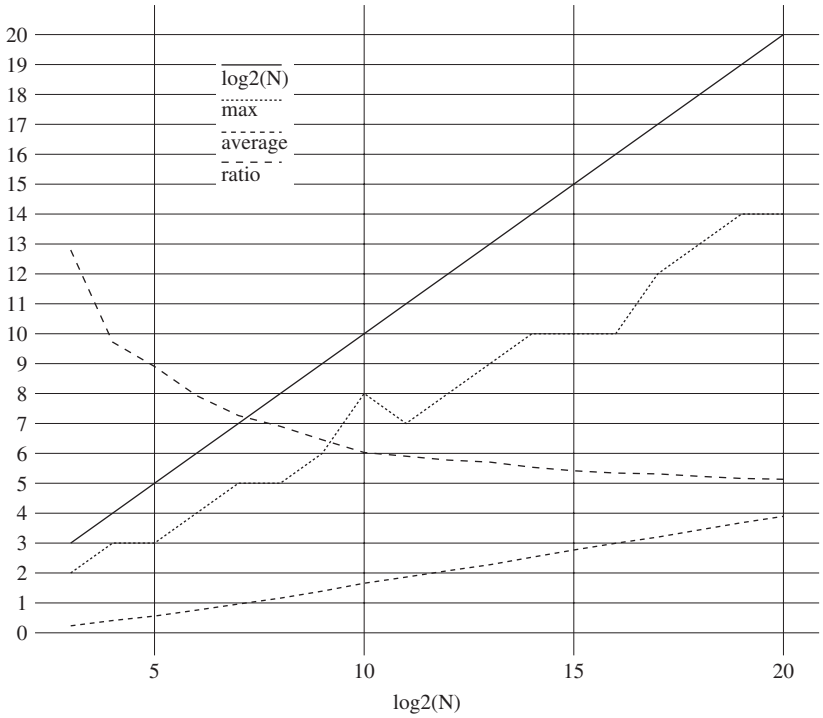


Fig. 1. Number of descents in the random case.

4. Conclusions, improvements, and open problems

We have shown that every lookup table T of size N can be coded by cN elements where c is a small constant, such that computations of the form $T^m(x)$—the mth iterate of T at x—can be done efficiently. The efficiency is measured in the number of recursions (descents) which our algorithm performs.

In the case that T is a permutation, no recursions are needed. When T is a general function, we can have up to $\sqrt{2N}$ recursions but not more than that, and if T is random, then the number of recursions reduces to about $(\log_2 N)/5$. The last assertion was only verified experimentally, and a rigorous explanation of this reduction from $O(\sqrt{N})$ to $O(\log N)$ in the random case would be interesting.

In a work in progress with Yossi Oren, we introduce another heuristic for the decomposition of graphs of functions. For this heuristic, the maximal number of descents reduces to $\log_2 N$ (which is optimal with respect to the worst-case behavior). There are still cases where the heuristic described in the current paper outperforms the newer heuristic, though.

The task of finding a heuristic approach which reduces the average number of recursions in the computations $T^m(x)$ to less than our $(\log_2 N)/5$ seems to be of great practical interest.

References

[1] O. Goldreich, S. Goldwasser, A. Nussboim, On the implementation of huge random objects, 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03), pp. 68–79. Full version: ⟨http://www.wisdom.weizmann.ac.il/~oded/p_toro.html⟩.

[2] M. Naor, O. Reingold, Constructing Pseudo-Random Permutations with a Prescribed Structure, J. Cryptology 15 (2002) 97–102.

[3] B. Tsaban, Permutation graphs, and sampling the cycle structure of a permutation, J. Algorithms 47 (2003) 104–121.