Average-Case Analysis Using Kolmogorov Complexity

Ming Li*
City University of Hong Kong and University of Waterloo

Paul Vitányi†
CWI and University of Amsterdam

Abstract

This expository paper demonstrates how to use Kolmogorov complexity to do the average-case analysis via four examples, and exhibits a surprising property of the celebrated associated universal distribution. The four examples are: average case analysis of Heapsort [17, 15], average nni-distance between two binary rooted leave-labeled trees [20], compact routing in computer networks [3], average-case analysis of an adder algorithm [4]. The property is that the average-case complexity of any algorithm whatsoever equals its worst-case complexity if the inputs are distributed according to the Universal Distribution [14]. We provide the proofs for the latter three items.

1 Introduction

Kolmogorov complexity has been very successfully applied to obtain lower bounds solving many long-standing open questions. See [15] for details. A much less well-known fact is that Kolmogorov complexity is also a powerful tool for average-case analysis of algorithms. The purpose of this expository paper is to explain such ideas via several elegant examples. We do not intend to comprehensively survey such results.

Often, it is very difficult to analyze the average-case complexity of an algorithm. This is because, unlike the worse-case analysis, the average-case analysis has to average over all instances of the input. In average-case analysis, the incompressibility method has an advantage over a probabilistic approach. In the latter approach, one deals with expectations or variances over some ensemble of objects. Using

*Supported in part by the NSERC Operating Grant OGP0046506, ITRC, a CGAT grant, and the Steacie Fellowship. Current address: Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. On sabbatical leave from: Department of Computer Science, University of Waterloo, Waterloo, Ont. N2L 3G1, Canada. E-mail: mli@cs.cityu.edu.hk

†Partially supported by the European Union through NeuroCOLT ESPRIT Working Group Nr. 8556, and by NWO through NFI Project ALADDIN under Contract number NF 62-376 and NSERC under International Scientific Exchange Award ISE0125663. Address: CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. Email: paulv@cwi.nl

Kolmogorov complexity, we can reason about an incompressible individual object. Because it is incompressible it has all statistical properties with certainty, rather than having them hold with some (high) probability as in a probabilistic analysis. This fact greatly simplifies the resulting analysis.

We briefly review the definition of Kolmogorov complexity. For a complete treatment of this subject, see [15]. Fix a universal Turing machine U with binary input alphabet. The machine U takes two inputs p and y. U interprets p as a program and simulates p on input y. The Kolmogorov complexity of a binary string x, given y, is defined as

$$C(x|y) = \min\{l(p) : U(p, y) = x\},$$

where $l(p)$ denotes the *length* (number of bits) of p. (If k is a number then $|k|$ denotes the *absolute value* of k. If A is a set then $d(A)$ denotes the *cardinality* of A, that is, the number of elements in it.) Thus $C(x|y)$ is the *minimum* number of bits in a description from which x can be effectively reconstructed, given y. Let $C(x) = C(x|\epsilon)$, where ϵ denotes the null string.

By a simple counting argument, the following claim can be easily proved.

Claim 1 *For each n and $c < n$, any y, there are at least $2^n - 2^{n-c}$ strings of length n with the property*

$$C(x|n, y) \geq n - c. \tag{1}$$

We call a string c-random if it satisfies $C(x|n, y) \geq n - c$. An undirected graph G on n nodes can be encoded by $n(n-1)/2$ bits, each bit indicating whether a certain edge is present. We say a graph G of n nodes is c-random if $C(G|n) \geq n(n-1)/2 - c$, here we use G to denote its own encoding, and c can be generalized to a function of n.

2 Heapsort and Tree Distance

We avoid the question of in which cases of average-case analysis one can apply Kolmogorov complexity. To this question, the authors would like to know the answer as well. We instead give a few successful applications in this section and two more together with proofs in the next two sections.

Heapsort. Heapsort is a widely used sorting algorithm. It is the first algorithm that sorts n numbers in-place with running time *guaranteed* to be of order $n \log n$. Here 'in-place' means it does not require extra nontrivial memory space. The method was first discovered by J.W.J. Williams [21] and subsequently improved by R.W. Floyd [7].

The heapsort algorithm works in two steps. First it converts the input into a heap. Then it sorts the input by repeatedly deleting the root (smallest element) and restoring the heap. It is well-known that we can build a heap from an array of n integers in $O(n)$ time. The second stage runs in n rounds to empty the heap. Each round takes between $O(1)$ and $2 \log n$ steps for restoring the heap, but the precise bound was unknown.

To restore a (min-)heap after the root key is deleted, the Williams' original algorithm takes the rightmost element from the bottom of the heap, puts it in the root, then it pushes this element down (swap it with the larger child) the heap, making two comparison each step, until this element is smaller than both of its children. This process takes $2 \log n$ steps in the worst case.

Floyd's algorithm compares the two children of the root, promotes the larger, and keeps on doing this until reaching the bottom, and then it fills the empty spot with the rightmost element in the bottom, and pushes this element back up the tree until it is greater than its father (precisely at the same position as in Williams' algorithm). The worst case of Floyd's algorithm is also $2 \log n$.

Despite Heapsort's prominence and serious efforts, the average case of Heapsort was open for 30 years. People tried to give probabilistic analysis of these two algorithms, but after 1 round of update, the probabilistic distribution changes. Only recently Schaffer and Sedgewick [19] succeeded in giving a precise analysis of its average case performance. I. Munro [17] suggested a remarkably simple solution using incompressibility. The idea is as follows. Fix a random heap H of Kolmogorov complexity approximately $n \log n$. For each of the n heap-restoring rounds, record the position where the last element finally resides in H. This position can be recorded by a 0-1 sequence encoding a path from the root to the position, with 0 indicating left branch and 1 indicating right. Each sequence is of length up to $\log n$. It is easy to see that one can reconstruct H from these n sequences. Thus, the average length of these sequences must be approximately $\log n$ (because Kolmogorov complexity of H is at least $n \log n$). Since most heaps are random, averaging, we conclude that Floyd's algorithm runs in $\log n$ steps on average, and Williams' algorithm uses $2 \log n$ steps on average.

Nni Distance. In computational biology, evolutionary trees are represented by unrooted unordered binary trees with uniquely labeled leaves and unlabeled internal nodes. Measuring the distance between such trees is useful in biology. A *nearest neighbor interchange* (*nni*) operation swaps two subtrees that are separated by an internal edge (u, v), as shown in Figure 1. See [13] for relevant references.

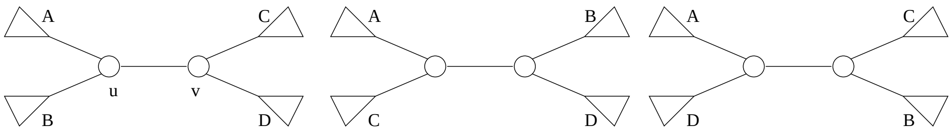
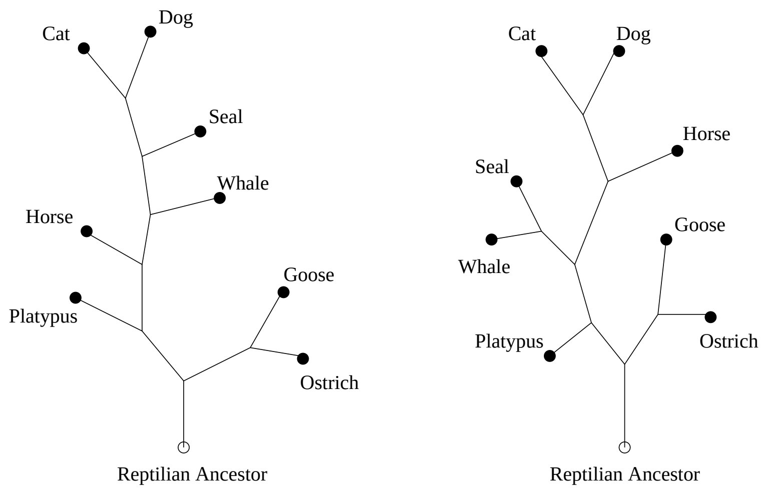


Figure 1: The two possible nni operations on an internal edge (u, v).

For example, in Figure 2 it takes 2 nni moves to convert (i) to (ii).

K. Culik II and D. Wood [6], improved by [13], proved that $n \log n + O(n)$ nni moves are sufficient to transform a tree of n leaves to any other tree with the same set of leaves. But the question is, is this the best upper bound? D. Sleator, R. Tarjan, and W. Thurston [20] proved an $\Omega(n \log n)$ lower bound for most pairs of trees, essentially using the incompressibility method. (Note, they proved their results for a more general graph transformation system.)

The idea behind the proof is simple. Consider T_1 and T_2 such that $C(T_1|T_2) \geq$



(i) (ii)

Figure 2: The nni distance between (i) and (ii) is 2

$n \log n$. If we can encode each nni move with $O(1)$ bits, then there must be at least $\Omega(n \log n)$ nni moves since otherwise $C(T_1|T_2) < n \log n$. It is the encoding process that is hard and we refer the reader to [20].

There are also other applications. For example, in [8] and [10], we used Kolmogorov complexity to analyze the average-case complexity of some Longest Common Subsequence and Shortest Common Supersequence algorithms.

3 Compact Routing in Computer Networks

In very large networks like the global telephone network or the internet the mass of messages being routed creates major bottlenecks degrading performance. In this section, we are interested in determining the optimal space to represent routing schemes in communication networks on the average for all static networks. We follow [3].

A universal routing strategy for static communication networks will, for every network, generate a *routing scheme* for that particular network. Such a routing scheme comprises a *local routing function* for every node in this network. The routing function of node u returns for every destination $v \neq u$ an edge incident to u on a path from u to v. This way, a routing scheme describes a path, called a *route*, between every pair of nodes u, v in the network.

It is easy to see that we can do shortest path routing by entering a routing table in each node u which for each destination node v indicates to what adjacent node w a message to v should be routed first. If u has degree d, it requires a table of at most $n \log d$ bits, and the overall number of bits in all local routing tables never exceeds $n^2 \log n$.

Several factors may influence the cost of representing a routing scheme for a particular network. We use a basic model and refer the readers to [3] for other variations. Here, we consider point to point communication networks on n nodes described by an undirected labeled graph $G = (V, E)$, where $V = \{1, \ldots, n\}$. Assume that the nodes know the identities of their neighbors. This information is for free.

Theorem 2 *For shortest path routing in $O(\log n)$-random graphs local routing functions can be stored in $6n$ bits per node. Hence the complete routing scheme is represented by $6n^2$ bits.*

Proof: The next two lemmas can be proved easily by Kolmogorov complexity, we leave the proofs to the readers.

Lemma 3 *All $o(n)$-random labeled graphs have diameter 2.*

Lemma 4 *Let c be a fixed constant. If G is a $c \log n$-random labeled graph, then from each node i all other nodes are either directly connected to i or are directly connected to one of the least $(c+3)\log n$ nodes directly adjacent to i.*

Let G be an $O(\log n)$-random graph on n nodes. By Lemma 4 we know that from each node u we can shortest path route to each node v through the least $O(\log n)$ directly adjacent nodes of u. By Lemma 3, G has diameter 2. Once the message reached node v its destination is either node v or a direct neighbor of node v (which is known in node v by assumption). Therefore, routing functions of size $O(n \log \log n)$ can be used to do shortest path routing. We can do better than this.

Let $A_0 \subseteq V$ be the set of nodes in G which are not directly connected to u. Let $v_1, \ldots, v_m$ be the $O(\log n)$ least nodes directly adjacent to node u, Lemma 4, through which we can shortest path route to all nodes in A_0. For $t = 1, 2 \ldots, l$ define $A_t = \{w \in A_0 - \bigcup_{s=1}^{t-1} A_s : (v_t, w) \in E\}$. Let $m_0 = d(A_0)$ and define $m_{t+1} = m_t - d(A_{t+1})$. Let l be the first t such that $m_t < n / \log \log n$. Then we claim that v_t is connected by an edge in E to at least $1/3$ of the nodes not connected by edges in E to nodes $u, v_1, \ldots, v_{t-1}$.

Claim 5 *$d(A_t) > m_{t-1}/3$ for $1 \leq t \leq l$.*

Proof: Suppose, by way of contradiction, that there exists a least $t \leq l$ such that $|d(A_t) - m_{t-1}/2| \geq m_{t-1}/6$. Then we can describe G, given n, as follows.

- This discussion in $O(1)$ bits.
- Nodes u, v_t in $2 \log n$ bits, padded with 0's if need be.
- The presence or absence of edges incident with nodes $u, v_1, \ldots, v_{t-1}$ in $r = n - 1 + \cdots + n - (t-1)$ bits. This gives us the characteristic sequences of $A_0, \ldots, A_{t-1}$ in V, where a *characteristic sequence* of A in V is a string of $d(V)$ bits with, for each $v \in V$, the vth bit equals 1 if $v \in A$ and the vth bit is 0 otherwise.
- A self-delimiting description of the characteristic sequence of A_t in $A_0 - \bigcup_{s=1}^{t-1} A_s$, using Chernoff's bound, in at most $m_{t-1} - (1/6)^2 m_{t-1} \log e + O(\log m_{t-1})$ bits.

- The description $E(G)$ with all bits corresponding to the presence or absence of edges between v_t and the nodes in $A_0 - \bigcup_{s=1}^{t-1} A_s$ deleted, saving m_{t-1} bits. Furthermore, we delete also all bits corresponding to presence or absence of edges incident with $u, v_1, \ldots, v_{t-1}$ saving a further r bits.

This description of G uses at most

$$n(n-1)/2 + O(\log n) + m_{t-1} - (1/6)^2 m_{t-1} \log e - m_{t-1}$$

bits, which contradicts the $O(\log n)$-randomness of G because $m_{t-1} > n/\log\log n$. ■

Recall that l is the least integer such that $m_l < n/\log\log n$. We construct the local routing function $F(u)$ as follows.

- A table of intermediate routing node entries for all the nodes in A_0 in increasing order. For each node w in $\bigcup_{s=1}^{l} A_s$ we enter in the wth position in the table the unary representation of the least intermediate node v, with $(u,v),(v,w) \in E$, followed by a 0. For the nodes that are not in $\bigcup_{s=1}^{l} A_s$ we enter a 0 in their position in the table indicating that an entry for this node can be found in the second table. By Claim 5, the size of this table is bounded by:
$$n + \sum_{s=1}^{l} (1/3)(2/3)^{s-1} sn \le n + \sum_{s=1}^{\infty} (1/3)(2/3)^{s-1} sn \le 4n$$
- A table with explicitly binary coded intermediate nodes on a shortest path for the ordered set of the remaining destination nodes. Those nodes had a 0 entry in the first table and there are at most $m_l < n/\log\log n$ of them, namely the nodes in $A_0 - \bigcup_{s=1}^{l} A_s$. Each entry consists of the code of length $\log\log n + O(1)$ for the position in increasing order of a node out of $v_1, \ldots, v_m$ with $m = O(\log n)$ by Lemma 4. Hence this second table requires at most $2n$ bits.

The routing algorithm is as follows. The direct neighbors of u are known in node u and are routed without routing table. If we route from start node u to target node w which is not directly adjacent to u, then we do the following. If node w has an entry in the first table then route over the edge coded in unary, otherwise find an entry for node w in the second table.

Altogether, we have $d(F(u)) \le 6n$. Slightly more precise counting and choosing l such that m_l is the first such quantity $< n/\log n$ shows $d(F(u)) \le 3n$. ■

A matching lower bound of $\Omega(n^2)$ can also be proved.

Theorem 6 *For shortest path routing in $o(n)$-random graphs each local routing function must be stored in at least $n/2 - o(n)$ bits per node. Hence the complete routing scheme requires at least $n^2/2 - o(n^2)$ bits to be stored.*

The results on Kolmogorov random graphs above have the following corollaries. Consider the subset of $(3\log n)$-random graphs within the class of $O(\log n)$-random graphs on n nodes. They constitute a fraction of at least $(1 - 1/n^3)$ of the class of all graphs on n nodes. The trivial upper bound on the minimal total number of bits for all routing functions together is $O(n^2 \log n)$ for shortest path routing on all graphs on n nodes. Simple computation of the average of the total number of bits used to store the routing scheme over all graphs on n nodes shows that both Theorem 2 and Theorem 6, hold for the *average case*.

The average case consists of the average cost, taken over all labeled graphs of n nodes, of representing a routing scheme for graphs over n nodes. For a graph G, let $T(G)$ be the number of bits used to store its routing scheme. The *average* total number of bits to store the routing scheme for routing over labeled graphs on n nodes is $\sum T(G)/2^{n(n-1)/2}$ with the sum taken over all graphs G on nodes $\{1, 2, \ldots, n\}$. That is, the uniform average over all the labeled graphs on n nodes.

4 Addition in $\log_2 n$ Steps on Average

Half a century ago, Burks, Goldstine, and von Neumann obtained a $\log_2 n$ expected upper bound on the 'longest carry sequence' for adding two n-bit binary numbers [2]. In computer architecture design, efficient design of adders directly affects the length of CPU clock cycle. The following algorithm (and its analysis using [2]) for adding two n-bit binary numbers x and y is known to the computer designers and can be found in standard computer arithmetic design books such as [9].

1. $S := x \oplus y$ (add bit-wise ignoring carries); $C :=$ carry sequence;
2. *while* $C \neq 0$ *do*
   $S := S \oplus C$;
   $C :=$ new carry sequence.

Let's call this 'no-carry adder' algorithm. The expected $\log_2 n$ carry sequence length upper bound of [2] implies that this algorithm runs in $1 + \log_2 n$ expected rounds (step 2). It turns out that this algorithm is the most efficient addition algorithm in the expected case currently known. Of course, it takes n steps in the worst case. This algorithm, in the average case, is exponentially faster than the trivial linear time 'ripple-carry adder' and it is two time faster than the well-known 'carry-lookahead adder'.

In the ripple-carry adder, the carry ripples from right to left, bit by bit, and hence it takes $\Omega(n)$ steps to compute the summation of two n-bit numbers.

The carry-lookahead adder is based on a divide and conquer algorithm which adds two n-bit numbers in $1 + 2\log_2 n$ steps. It is used in nearly all modern computers. For details about both adders, see any standard computer architecture textbook such as [9, 5].

The results in [2], [1], and [18] imply that the no-carry adder has expected time of at most $1 + \log_2 n$. But these proofs are all nontrivial probabilistic analysis.

[4] has given an almost trivial and elementary proof of the same fact using Kolmogorov complexity. We present their proof here.

Theorem 7 *The no-carry adder has the average running time of at most* $1+\log_2 n$.

Proof: For any binary string input x and y such that $l(x) = l(y) = n$, if the no-carry adder uses t rounds (i.e., excuting Step 2 for t times), then x and y can be written as

$$x = x'bu1x'', y = y'b\overline{u}1y'',$$

where $l(u) = t - 1$, $l(x') = l(y')$, b is 0 or 1, and $\overline{u}$ is the complement of u. Now we can describe x using y, n, q and the concatentation of the following binary strings:

- the position of u in y (in *exactly* $\log_2 n$ bits by padding),
- $x'x''$.

Here the program q contains information telling U how to compose x from the given information. Since the above two strings have total length $n - t - 1 + \log_2 n$, the value t can be deduced from n and input length. So $t+1$ bits of x are saved at the cost of extra $\log_2 n$ bits. See [4] for more careful discussion. Thus $C(x|n, y, q) \leq n - t - 1 + \log_2 n$. Therefore, for any string x of length n with $C(x|n, y, q) = n - i$, the algorithm must stop in at most $\log_2 n + i - 1$ steps on input x and y.

Since there are only 2^{n-i} programs of length $n-i$. there are at most 2^{n-i} strings x of length n with Kolmogorov complexity $C(x|n, y, q) = n - i$. Let p_i denote the probability that $C(x|n, y, q) = n - i$ for $l(x) = n$. Then $p_i \leq 2^i$ and $\sum p_i = 1$. Thus the average running time for each y is bounded above by

$$\sum_{i=2-\log n}^{n} p_i(i - 1 + \log n) \leq 1 + \log n$$

Since this holds for every y, this is also the average running time of the algorithm.
■

5 Average-Case Complexity Equals Worst-Case Complexity Under Universal Distributions

Consider a Turing machine such that the set of programs for which it halts is prefix-free, that is, no such program is the proper prefix of another such program. Such self-delimiting Turing machines compute all partial recursive functions and contain an appropriate universal machine U'. Similar to before we can define Kolmogorov complexity with respect to U' which is now induced by a set of prefix-free programs. The resulting *prefix complexity* $K(x)$ is slightly larger than $C(x)$, that is, $C(x) \leq K(x) \leq C(x) + 2 \log C(x)$.

The *universal distribution* $\mathbf{m}$ defined by $\mathbf{m}(x) = 2^{-K(x)}$ is one of the foremost notions in all of the the theory of Kolmogorov complexity. In [15] we give many remarkable properties an applications for this fundamental notion. It multiplicatively dominates all enumerable distributions (and therefore also all computable ones). Therefore, a priori it maximizes ignorance by assigning maximal probability to all objects. In [14, 15] we showed that the average-case computational complexity of *any algorithm whatsoever* under the universal distribution turns out to be of the same order of magnitude as the worst-case complexity. This holds both for time complexity and for space complexity.

For many algorithms the average-case running time under some distributions on the inputs is less than the worst-case running time. For instance, using (non-randomized) Quicksort on a list of n items to be sorted gives under the uniform distribution on the inputs an average running time of $O(n \log n)$ while the worst-case running time is $\Omega(n^2)$. The worst-case running time of Quicksort is typically reached if the list is already sorted or almost sorted, that is, exactly in cases where we actually should not have to do much work at all. Since in practice the lists to be sorted occurring in computer computations are often sorted or almost sorted, programmers often prefer other sorting algorithms which might run faster with almost sorted lists. Without loss of generality we identify inputs of length n with the natural numbers corresponding with binary strings of length n.

Definition 1 Consider a discrete sample space $\mathcal{N}$ with probability density function P. Let $t(x)$ be the running time of algorithm A on problem instance x. Define the *worst-case time complexity* of A as $T(n) = \max\{t(x) : l(x) = n\}$. Define the *$P$-average time complexity* of A

$$T(n|P) = \frac{\sum_{l(x)=n} P(x)t(x)}{\sum_{l(x)=n} P(x)}.$$

We compare the average time complexity for Quicksort under the Uniform Distribution $L(x)$ and under the Universal distribution $\mathbf{m}(x)$. Define $L(x) = 2^{-2l(x)-1}$, such that the conditional probability $L(x|l(x) = n) = 2^{-n}$. We encode the list of elements to be sorted as nonnegative integers in some standard way.

For Quicksort, $T(n|L) = \Theta(n \log n)$. We may expect the same complexity under $\mathbf{m}$, that is, $T(n|\mathbf{m}) = \Omega(n \log n)$. But Theorem 8 will tell us much more, namely, $T(n|\mathbf{m}) = \Omega(n^2)$. Let us give some insight why this is the case.

With the low average time complexity under the Uniform Distribution, there can only be $o((\log n)2^n/n)$ strings x of length n with $t(x) = \Omega(n^2)$. Therefore, given n, each such string can be described by its sequence number in this small set, and hence for each such x we find $K(x|n) \leq n - \log n + 3 \log \log n$. (Since n is known, we can find each $n - k$ by coding k self-delimiting in $2 \log k$ bits. The inequality follows by setting $k \geq \log n - \log \log n$.)

Therefore, no really random x's, with $K(x|n) \geq n$, can achieve the worst-case run time $\Omega(n^2)$. Only strings x which are nonrandom, with $K(x|n) < n$, among which are the sorted or almost sorted lists, and lists exhibiting other regularities, can have $\Omega(n^2)$ running time. Such lists x have relatively low Kolmogorov complexity $K(x)$ since they are regular (can be shortly described), and therefore $\mathbf{m}(x) = 2^{-K(x)}$ is very high. Therefore, the contribution of these strings to the average running time is weighted very heavily.

Theorem 8 (m-Average Complexity) *Let A be an algorithm with inputs in $\mathcal{N}$. Let the inputs to A be distributed according to the universal distribution $\mathbf{m}$. Then, the average case time complexity is of the same order of magnitude as the corresponding worst-case time complexity.*

Proof: We define a probability distribution $P(x)$ on the inputs that assigns high probability to the inputs for which the worst-case complexity is reached, and zero probability for other cases.

Let A be the algorithm involved. Let $T(n)$ be the worst-case time complexity of A. Clearly, $T(n)$ is recursive (for instance by running A on all x's of length n). Define the probability distribution $P(x)$ by

Step 1 **For** each $n = 0, 1, \ldots$, set $a_n := \sum_{l(x)=n} \mathbf{m}(x)$.

Step 2 **If** $l(x) = n$ and x is lexicographically least with $t(x) = T(n)$ **then** $P(x) := a_n$ **else** $P(x) := 0$.

It is easy to see that a_n is enumerable since $\mathbf{m}(x)$ is enumerable. Therefore, $P(x)$ is enumerable. Below we use a fact from [15], Theorem 4.1 and the following Example 4.5, that $c_P \mathbf{m}(x) \geq P(x)$, where $c_P = K(P) + O(1)$ is a constant depending on P but not on x. We have defined $P(x)$ such that $\sum_{x \in \mathcal{N}} P(x) \geq \sum_{x \in \mathcal{N}} \mathbf{m}(x)$, and $P(x)$ is an enumerable probability distribution. The average case time complexity $T(n|\mathbf{m})$ with respect to the $\mathbf{m}$ distribution on the inputs, is now obtained by

$$\begin{aligned} T(n|\mathbf{m}) &= \sum_{l(x)=n} \frac{\mathbf{m}(x)t(x)}{\sum_{l(x)=n} \mathbf{m}(x)} \\ &\geq \frac{1}{c_P} \sum_{l(x)=n} \frac{P(x)}{\sum_{l(x)=n} \mathbf{m}(x)} T(n) \\ &= \frac{1}{c_P} \sum_{l(x)=n} \frac{P(x)}{\sum_{l(x)=n} P(x)} T(n) = \frac{1}{c_P} T(n). \end{aligned}$$

The inequality $T(n) \geq T(n|\mathbf{m})$ holds vacuously. ■

The analogue of the theorem holds for other complexity measures (like *space* complexity), by about the same proof. Further research has been done on related measures that exhibit similar behaviour. See for example [16, 12, 11].

6 Acknowledgement

We thank our coauthors for papers [3, 4]: R. Beigel, H. Buhrman, W. Gasarch, J.H. Hoepman, L. Zhang. We have used the proofs from these two papers to demonstrate our points.

References

[1] B.E. Briley, Some new results on average worst case carry. *IEEE Trans. Computers*, C-22:5(1973).

[2] A.W.Burks, H.H. Goldstine, J. von Neumann, Preliminary discussion of the logical design of an electronic computing instrument. Institute for Advanced Studies, Report (1946). Reprinted in *John von Neumann Collected Works*, vol 5 (1961).

[3] H. Buhrman, J.H. Hoepman, P.M.B. Vitányi, Optimal routing tables, *Proc. 15th ACM Symp. Principles Distribut. Comput.*, ACM Press, 1996, 134-142.

[4] R. Beigel, W. Gasarch, M. Li, and L. Zhang, Addition in $\log_2 n$ steps on average, *manuscript*.

[5] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to algorithms.* MIT Press, 1990.

[6] K. Culik II, D. Wood, A note on some tree similarity measures, *Inform. Process. Lett.*, 15(1982), 39-42.

[7] R.W. Floyd, Treesort 3: Algorithm 245, *Comm. ACM*, 7(1964), 701.

[8] D. Foulser, M. Li, and Q. Yang, Theory and algorithms for plan merging. *Artificial Intelligence*, 57(1992), 143-181.

[9] K. Hwang, *Computer arithmetic: principles, architecture, and design.* Wiley, New York, 1979.

[10] T. Jiang and M. Li, On the approximation of shortest common supersequences and longest common subsequences. *SIAM J. Comput.*, 24:5(1995), 1122-1139.

[11] A.K. Jagota and K.W. Regan, Testing Neural Net Algorithms on General Compressible Data, In *Proceedings of the International Conference on Neural Information Processing*, Hong Kong, 1996, Springer-Verlag.

[12] K. Kobayashi, On malign input distributions for algorithms, *IEICE Trans. Inform. and Syst.*, E76-D:6(1993), 634-640.

[13] M. Li, J. Tromp, and L. Zhang, On the nearest neighbor interchange distance, COCOON'96, Hong Kong, 1996. Final version to appear in *J. Theoret. Biology*, 1996.

[14] M. Li and P. Vitányi, Average case complexity equals worst-case complexity under the Universal Distribution. *Inform. Process. Lett.*, 42(1992), 145-149.

[15] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications.* Springer-Verlag, New York, 1993.

[16] P.B. Miltersen, The complexity of malign ensembles, *SIAM J. Comput.*, 22:1(1993), 147-156.

[17] I. Munro, *Personal communication*, 1993.

[18] G. Schay, How to add fast–on average. *American Mathematical Monthly*, 102:8 (1995), 725-730.

[19] R. Schaffer and R. Sedgewick, *J. Algorithms*, 15(1993), 76-100.

[20] D. Sleator, R. Tarjan, and W. Thurston, Short encodings of evolving structures, *SIAM J. Discr. Math.*, 5(1992), 428-450.

[21] J.W.J. Williams, Algorithm 232: HEAPSORT, *Comm. ACM*, 7(1964), 347-348.