Combinatorial Compression and Partitioning of Large Dictionaries*

Aviezri S. Fraenkel† ‡ and Moshe Mor
Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel 76100

A method for compressing large dictionaries is proposed, based on transforming words into lexicographically ordered strings of distinct letters, together with permutation indexes. Algorithms to generate such strings are described. Results of applying the method to the dictionaries of two large databases, in Hebrew and English, are presented. The main message is a method of partitioning the dictionary such that the 'information bearing fraction' is stored in fast memory, and the bulk in auxiliary memory.

Downloaded from https://academic.oup.com/comjnl/article/26/4/336/377426 by guest on 20 August 2022

1. INTRODUCTION

A method for compressing very large dictionaries—the larger the better!—based on combinatorial transformations of words is proposed. The main idea is to replace each word w by a pair (L, I), where L is an ordered string of the distinct letters of w, and I is an index which permits transforming L back into w. The information contained in the L's is almost the same as that of the w's: the entropy increase in transforming the latter to the former is very small. The main variation investigated is when the L's reside in fast memory and the I's are relegated to disk. This results in very high savings of fast memory.

Specifically, let $w = w_1 w_2 \dots w_k$ be a word over a finite alphabet Σ, linearly ordered (under $<$). A *lexicographic form* (*lexform* for short) of w is a lexicographically ordered sequence $w_{q(1)} \dots w_{q(l)}$ (for suitable $l \le k$) of the *distinct* letters (also called characters) of w. Thus $w_{q(i)}$ precedes $w_{q(j)}$ if and only if $w_{q(i)} < w_{q(j)}$. Every word over Σ maps into a unique lexform, but any given lexform may be induced by several distinct words.

We define a few basic notions. If a word $w = w_1 w_2 \dots w_k$ maps into a lexform $v = v_1 v_2 \dots v_l$ ($l \le k$), then the *index* of w is a sequence of length k consisting of the numbers $1, 2, \dots, l$, such that if $w_i = v_j$, then the ith sequence number is j ($1 \le j \le l$, $1 \le i \le k$). Denoting by L the lexform of w and by I its index, we observe that the transformation $w \to (L, I)$ is a bijection. Thus the transformation $w \to (L, I)$ has a unique inverse. A *text* is a sequence of words, counting repetitions. The set of distinct words of a text is a *dictionary* of the text. (Of course a dictionary is a special case of a text, namely the case in which every word appears exactly once.) The *length* of a word is the number of its letters, counting multiplicities. For example, 'of the people, by the people, for the people' is a text of size 9, whose dictionary has size 5. The word 'people' has length 6, its lexform is 'elop', and its index is (4,1,3,4,2,1).

The proposed compression and partitioning method is based on replacing words by lexforms, storing only distinct lexforms and their corresponding indexes. The number of distinct lexforms of length l over an alphabet Σ of size $|\Sigma| = n$ is evidently $\binom{n}{l}$. Since every combination can be represented by its serial number in some linear ordering of all combinations,[1,2] a *serial combination number* (*conumber* for short) can be used to represent every lexform, thus achieving additional compression. In Proposition 1 it is proved that the fraction of storage needed when replacing dictionary words of length k by conumbers is at most $(2\pi k)^{-1/2} (ek^{-1})^k$ if $|\Sigma|$ is large. In Proposition 3 it is shown that the number of distinct indexes of words of length k is $\sum_{i=1}^{\infty} i^k 2^{-i-1}$, which is the number of *Cayley-permutations* (*C-permutations* for short) of length k.[3] Thus if we replace every index by its serial number (called *rank*) in some linear ordering of all indexes, a further compression is achieved.

The combinatorial compression method can thus be viewed as consisting of two phases:

A. Compression by transforming dictionary words into lexforms and indexes.
B. Further compression by transforming lexforms into conumbers and indexes into ranks.

A natural partition of the dictionary is obtained by storing the file $\mathfrak{L}$ of lexforms (or their corresponding conumbers) in fast memory, and the file $\mathfrak{I}$ of indexes (or their ranks) on disk. Such a partition may enable storage of a large dictionary in form of its lexforms in fast memory, which otherwise could not be kept in it because of lack of space. This is important in many applications such as data retrieval over legal material or other non-numeric material. Typical cases are: (1) most accesses to the dictionary are unsuccessful, that is, the word sought is not in the dictionary. (2) Many accesses are successful, but additional Boolean or metrical constraints (which can be verified without consulting $\mathfrak{I}$) reject the word. In both of these cases there are many accesses to $\mathfrak{L}$ in fast memory, and few accesses to $\mathfrak{I}$ on disk, whose access time is typically 10^4 times slower than that of fast memory.

The method was tested on the dictionaries of two large databases, one of which was in fact a database of legal material, namely a subset of the database of the Responsa retrieval project.[4] The subset contained some 114 million letters—excluding punctuation characters and blanks—

* This work was done within the Responsa Retrieval Project, developed initially at the Weizmann Institute of Science and Bar-Ilan University, now located at the Institute for Information Retrieval and Computational Linguistics (IRCOL), Bar-Ilan University, Ramat Gan, Israel. The work reported herein was done at the Weizmann Institute.
† Partial affiliation with IRCOL.
‡ Supported in part by a grant of Bank Leumi Le' Israel.

CCC–0010–4620/83/0026–0336$04.00

© Wiley Heyden Ltd, 1983

comprising 28 million words (436 000 distinct (dictionary) words) mainly in Hebrew; and a subset of the database of seven biweekly updates of NTIS (U.S. National Technical Information Services), containing some 14 million letters of two million English words of length at least three (57 000 distinct words). Any word of length exceeding 13 was truncated to length 13.

The highlights of the results are that if phases A and B are used, then the above mentioned partitioning results in a fast memory space requirement of only 15% of the Responsa dictionary space; 55–60% of the NTIS dictionary. This rather large difference in compression is due not so much to language idiosyncracies as to dictionary size: the efficiency of the method increases with dictionary size! (We remark that the above saving is on top of an additional saving factor (not counted) obtained by replacing standard character representation by a minimal representation using only $\lceil \lg|\Sigma| \rceil$ bits per character (lg stands for log to the base 2, here and below). This is natural to do when working with conumbers and ranks, and is quite consistent with other compression methods. For example, if $|\Sigma| = 32$, a 5-bit code instead of the customary 8-bit code can be used, resulting in an additional 37.5% saving factor, not counted in the sequel.)

The details of the method—in form of phases A and B—are presented in Section 2. In Section 3 we briefly explore an extension and a variation of the main method. The extension is front compression applied to the file of lexforms; the variation is the use of performs instead of lexforms. A *perform* is a lexicographically ordered string of the letters of a word without deleting multiple letters. The final Section 4 contains highlights of the results of tests run on the two databases mentioned above. It ends with a short summary on decoding times, where decoding is the process of restoring the original word from its compressed version.

2. THE TWO PHASES OF COMBINATORIAL COMPRESSION

Phase A

This phase consists of two steps:
(i) Generation of lexforms and calculation of indexes.
(ii) Compression by sorted lexforms.

Step (i). This step transforms every word w in the dictionary D into a pair (L, I), where L is the lexform and I the index of w. The lexform is obtained by sorting the letters of w, deleting identical letters. Since the number of elements is small, any simple sorting algorithm such as insertion sort[5] will be more efficient than elaborate algorithms. If the lexform has length l, its characters are numbered consecutively from 1 to l. To get the index I of w, every letter of w is replaced by its corresponding number.

Step (ii). We start by sorting the pairs (L, I) lexicographically, where L is more significant than I. The input is a set of pairs $P = \{(L_k, I_k)\}_{k=1}^{d}$, where $d = |D|$ is the number of dictionary words, L_k is the lexform of the kth word and I_k its index $(1 \le k \le d)$. The sort produces a sequence

$$S = \{(L_k, I_k): (L_1, I_1) < \cdots < (L_d, I_d)\}.$$

In particular, $L_1 \le \cdots \le L_d$. Thereafter, all maximal blocks $(L_{k1}, I_{k1}), \ldots, (L_{kt}, I_{kt})$ for which $L_{k1-1} < L_{k1} = \cdots = L_{kt} < L_{kt+1}$ are collapsed into a single element consisting of a single lexform $L_k \equiv L_{k1}$, and a sequence of indexes $(I_{k1}, \ldots, I_{kt})$. The result is a sequence

$$A = \{(L_k; I_{k1}, \ldots, I_{kt}): I_{k1} < \cdots < I_{kt}, \quad 1 \le k \le r, L_1 < \cdots < L_r\}$$

where $r = |A|$ $(1 \le r \le d)$. Since d is normally large, it is advisable to use an efficient sorting method. For example, if D fits into fast memory at least temporarily, then quicksort, heapsort or radix exchange sort[5] may be used.

We now partition the sequence A into two sequences $\mathfrak{L} = \{L_1, \ldots, L_r\}$ of lexforms and $\mathfrak{I} = \{I_{11}, \ldots, I_{1t(1)}, \ldots, I_{r1}, \ldots, I_{rt(r)}\}$ of indexes. The sequence $\mathfrak{L}$ can be stored in fast memory, $\mathfrak{I}$ on disk. No pointers from $\mathfrak{L}$ to $\mathfrak{I}$ are required if the lexforms are repeated in $\mathfrak{I}$, serving there as key-fields.

Phase B

In phase B, lexforms and indexes produced in phase A are transformed into conumbers and ranks, respectively.

Step (i). Transformation of lexforms into conumbers. The number of distinct lexforms of length l over Σ is $\binom{n}{l}$, where $n = |\Sigma|$. Instead of representing a lexform v of length l by means of a string of l letters with a range of n^l, the same as a word of length l, we may represent it by its conumber, with a range of only $\binom{n}{l}$. This saving is on top of the saving achieved by using in the lexform only l out of k letters of the original word.

'Saving' here means the compression achieved in $\mathfrak{L}$ not in $\mathfrak{I}$. For the overall compression achieved, $\mathfrak{I}$ must also be considered, but since $\mathfrak{I}$ normally resides on disk, its storage is normally much cheaper than that of $\mathfrak{L}$.

Since $\binom{n}{l}$ grows rapidly with l $(< n/2)$, it is useful to consider only words of length $k \le 8$, which holds for 96% of the Responsa database and 62% of the NTIS database. Longer words may be partitioned into segments of length ≤ 8.

Note that for full use of the compression of phase B, the internal representation of characters should be reduced to the minimum number of bits required, whence the saving is counted in bits rather than bytes. This is consistent with common data compression techniques, in which characters over Σ are normally represented by a minimal number of $\lceil \lg n \rceil$ bits which may be shorter than the standard internal computer representation of characters.

We now get an asymptotic lower bound on the saving gained up to this point.

Proposition 1

The fraction of storage needed when replacing dictionary words of length k by conumbers is at most $t = (2\pi k)^{-1/2}(ek^{-1})^k$ if $|\Sigma|$ is large.

Proof. We use the following form of Stirling's formula (Ref. 6: 6.1.38):

$$\sqrt{(2\pi r)}\left(\frac{r}{e}\right)^r < r! < \sqrt{(2\pi r)}\left(\frac{r}{e}\right)^r e^{1/12r}$$

Downloaded from https://academic.oup.com/comjnl/article/26/4/336/377426 by guest on 20 August 2022

for all $r > 0$. Letting $n = |\Sigma|$, we thus get

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!} < \frac{1}{\sqrt{(2\pi)}} \frac{n^k e^{1/12n}}{k^{k+1/2}(1-k/n)^{n-k+1/2}}$$

$$\rightarrow \frac{1}{\sqrt{(2\pi k)}}\left(\frac{en}{k}\right)^k = tn^k \quad \text{as } n \rightarrow \infty,$$

since $(1-(k/n))^n \rightarrow e^{-k}$ as $n \rightarrow \infty$.

Thus even if every lexform induced by words of length k has length k, the number of distinct lexforms is asymptotically bounded below by tn^k. Since the number of distinct words of length k over Σ is n^k, the fraction of storage needed is at most t. ∎

Note that the fraction of storage needed is independent of $|\Sigma|$ as long as $|\Sigma|$ is large. Table 1 exhibits the savings projected by Proposition 1. The column headed by $-\lg t$ gives the projected savings in terms of the difference of the number of bits between a representation by words and by conumbers.

Table 1. Asymptotic lower bounds on savings (in bits) obtained by replacing dictionary words by conumbers

k	$-\lg t$	k	$-\lg t$
2	0.9	6	9.5
3	2.5	7	12.3
4	4.5	8	15.2
5	6.8		

Let $N_1(k) = \lceil \lg n^k \rceil$ and $N_2(k) = \lceil \lg \binom{n}{k} \rceil$. Then the actual saving in bits is at least $S = N_1(k) - N_2(k)$. Several of these values are displayed in Table 2. The first four values of n are powers of 2. If n is not a power of 2, the savings are larger, because several possible characters are unused.

Table 2. Actual lower bounds on savings obtainable by replacing dictionary words by conumbers

n	k	Actual savings (bits)	n	k	Actual savings (bits)
32	2	1	256	2	1
32	3	2	256	3	2
32	4	4	256	4	4
32	5	7	256	5	6
32	6	10	256	6	9
32	7	13	256	7	12
32	8	16	256	8	15
64	2	1	26	2	1
64	3	2	26	3	3
64	4	4	26	4	6
64	5	7	26	5	8
64	6	9	26	6	12
64	7	12	26	7	15
64	8	15	26	8	19
128	2	1	36	2	2
128	3	2	36	3	5
128	4	4	36	4	8
128	5	7	36	5	11
128	6	9	36	6	15
128	7	12	36	7	19
128	8	15	36	8	23

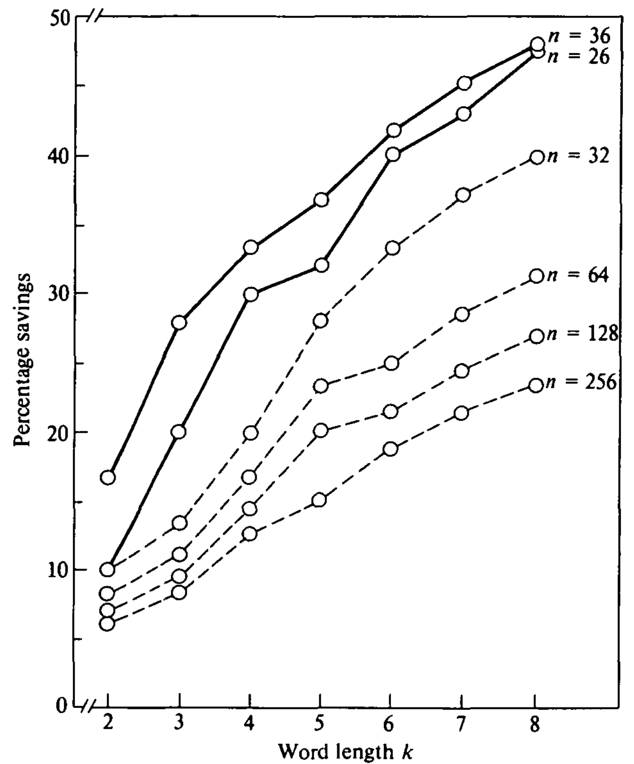

Figure 1. Percentage of actual savings achieved by replacing dictionary words by conumbers (lower bounds).

This situation is reflected in the last two n-values of Table 2, for $n = 26$ and $n = 36$ (Latin alphabet supplemented by the digits 0–9, say). Comparing the last columns of Tables 1 and 2, it is seen that the estimate of Proposition 1 is rather close to the actual lower bound for word lengths 2–8. Figure 1 displays graphically the percentage $S/N_1(k)$ of the actual savings. Note that Table 2 and Figure 1 reflect lower bounds on the actual savings, since we assumed that lexforms have the same length as words.

For formulating transformations between a lexform and its conumber, define the *combinatorial representation* of any non-negative integer N with respect to a fixed positive integer k, to be $(a_1, \ldots, a_k)$, where

$$N = \binom{a_k}{k} + \binom{a_{k-1}}{k-1} + \cdots + \binom{a_1}{1}$$

subject to $0 \le a_1 < a_2 < \cdots < a_k$ for uniqueness (see Ref. 1, p. 8).

A combination c out of a set of $\binom{n}{l}$ combinations is fixed by selecting l positions $b_1, \ldots, b_l$ with $1 \le b_1 < \cdots < b_l \le n$ out of n positions. The conumber r $(0 \le r < \binom{n}{l})$ of c is defined to be

$$\binom{n}{l} - \sum_{j=1}^{l} \binom{n-b_j}{l-j+1} - 1 \quad \text{(Ref. 1, p. 28; Ref. 2, p. 33).}$$

Conversely, the conumber of a combination c determines the positions $b_1, \ldots, b_l$: Given the conumber r of a combination out of $\binom{n}{l}$ combinations $(0 \le r < \binom{n}{l})$, represent $R = \binom{n}{l} - r - 1$ in the combinatorial representation, that is,

$$R = \sum_{j=1}^{l} \binom{c_j}{l-j+1}$$

Then $b_j = n - c_j$ $(1 \le j \le l)$ are the desired positions.

Downloaded from https://academic.oup.com/comjnl/article/26/4/336/377426 by guest on 20 August 2022

We now partition the set of lexforms into subsets, each containing lexforms of fixed length l ($2 \leq l \leq 8$). (Note that a subset containing lexforms of length l is normally derived from words of various lengths $k \geq l$.) The lexforms in each subset are transformed into conumbers. The savings thus obtained are those estimated in Proposition 1 and Tables 1, 2 and Fig. 1.

Decoding involves computing the combinatorial representation. For computing the combinatorial representation of a non-negative integer N with respect to k, we have to calculate the largest integer a_k satisfying

$$\binom{a_k}{k} \leq N;$$

the largest integer a_{k-1} satisfying

$$\binom{a_{k-1}}{k-1} \leq N - \binom{a_k}{k};$$

the largest integer a_{k-2} satisfying

$$\binom{a_{k-2}}{k-2} \leq N - \binom{a_k}{k} - \binom{a_{k-1}}{k-1}; \ldots$$

It is thus of importance to give an efficient method for computing the combinatorial representation. Here is one.

Let M be a positive integer. For computing efficiently the largest integer $x = x_0$ satisfying $\binom{x}{r} \leq M$, recall that the proof of Proposition 1 shows that $\binom{x}{r} \sim (2\pi r)^{-1/2}(exr^{-1})^r$ (where $\sim$ denotes 'asymptotic to'). Hence it makes sense to start with

$$x_1 = \left\lceil \frac{r}{e}(\sqrt{(2\pi r)}\, M)^{1/r} \right\rceil$$

Indeed, the following holds:

Proposition 2

For $r = 2$, $x_0 = \lfloor (1 + \sqrt{(1 + 8M)})/2 \rfloor$. For $r > 2$, $x_1 \leq x_0 < x_2$, where

$$x_2 = \left\lceil \frac{r}{e}(\sqrt{(2\pi r)}\, M\, e^{1/12r})^{1/r} \right\rceil + r - 1$$

Proof. For $r = 2$, the requirement of determining the largest solution of the quadratic inequality $\binom{x}{2} \leq M$ is directly seen to be $x_0 = \lfloor (1 + \sqrt{(1 + 8M)})/2 \rfloor$.

For any real x, $x(x-2) < x^2 - 2x + 1 = (x-1)^2$. Hence for any $x > 1$, $x(x-1)(x-2) < (x-1)^3$. Therefore for $r > 2$,

$$\binom{x_1}{r} = \frac{x_1(x_1-1)\ldots(x_1-r+1)}{r!} < \frac{(x_1-1)^r}{r!} < \frac{\sqrt{(2\pi r)}}{r!}\left(\frac{r}{e}\right)^r M$$

Thus Stirling's formula (see proof of Proposition 1), implies

$$\binom{x_1}{r} < M;$$

hence $x_1 \leq x_0$.

On the other hand,

$$\binom{x_2}{r} = \frac{x_2(x_2-1)\ldots(x_2-r+1)}{r!} \geq \frac{(x_2-r+1)^r}{r!} \geq \frac{\sqrt{(2\pi r)}}{r!}\left(\frac{r}{e}\right)^r M\, e^{1/12r} > M$$

∎

Note that for fixed M, even very large M, we have

$$\left\lceil \frac{r}{e}(\sqrt{(2\pi r)}\, M\, e^{1/12r})^{1/r} \right\rceil - \left\lceil \frac{r}{e}(\sqrt{(2\pi r)}\, M)^{1/r} \right\rceil \to 0$$

as r increases, and the convergence is very fast. Hence $x_2 - x_1 \leq r$ even for r not very large. Thus the computation of x_0 involves relatively few steps. This is illustrated in Table 3, which exhibits the values $x_0 - x_1$ and $x_2 - x_0$ for $1 \leq M \leq 3 \times 10^6$, $3 \leq r \leq 8$. It is seen that starting with x_1, at most r steps are required to get to x_0.

Step (ii). Transformation of indexes into ranks. Recall that a rank of an index is the serial number of the index in some linear ordering of all the indexes.

Proposition 3

The number of indexes of words of length k is

$$K_k = \sum_{i=1}^{\infty} i^k 2^{-i-1}$$

Table 3. The values $x_0 - x_1$ and $x_2 - x_0$ as a function of r for $1 \leq M \leq 3 \times 10^6$

	r: 3	4	5	6	7	8
$x_0 - x_1$						
0	52744	326426	0	0	0	0
1	750965	2673574	2491963	1105362	0	0
2	1920838	0	508037	1894637	2625013	1094734
3	275453	0	0	1	374987	1905204
4	0	0	0	0	0	2
5 and above	0	0	0	0	0	0
$x_2 - x_0$						
0	0	0	0	0	0	0
1	482	0	0	0	0	0
2	2985775	1533445	72773	1	0	0
3	13743	1466555	2927227	1697737	246082	2
4	0	0	0	1302262	2753918	1813950
5	0	0	0	0	0	1186048
6 and above	0	0	0	0	0	0

Downloaded from https://academic.oup.com/comjnl/article/26/4/336/377426 by guest on 20 August 2022

Proof. A C-permutation p of length k over $S = \{1,\ldots,k\}$ is a permutation of n elements from S with possible repetitions, such that if j appears in p, then also every $i < j$ appears in it. Note that an index of a word of length k is precisely a C-permutation of length k on the set $S = \{1, \ldots, k\}$. The result now follows since the number of C-permutations of length k over S is K_k.[3] ∎

The transformation between C-permutations and their ranks is effected by means of two algorithms given in Ref. 3.

Assuming words of length k with distinct letters, the saving gained by transforming indexes into ranks is $k^{-k}K_k$, since k^k is the number of k-digit numbers of length k. Table 4 shows several savings achievable by replacing indexes by ranks. Note that this is a saving achieved in $\mathcal{I}$ rather than in $\mathcal{L}$.

Table 4. Savings achieved by using ranks instead of indexes

k	k^k	$\lceil \lg k^k \rceil$ Number of bits of k^k	K_k	$\lceil \lg K_k \rceil$ Number of bits of K_k	Number of bits saved	Percentage savings
2	4	2	3	2	0	0
3	27	5	13	4	1	20.0
4	256	8	75	7	1	12.50
5	3125	12	541	10	2	16.67
6	46656	16	4683	13	3	18.75
7	823543	20	47293	16	4	20.0
8	16777216	24	545835	20	4	16.67

3. EXTENSIONS AND VARIATIONS

Among the various possibilities for extensions and variations of the method, we point out briefly one extension and one variation.

(i) Front compression

Instead of transforming lexforms into conumbers, the sorted file of lexforms can be compressed by front compression. That is, identical leading characters of consecutive lexforms are replaced by their count of identical characters (except for the first lexform in the sequence).[7] It is then natural to apply front compression also to all words of length exceeding 8.

Front compression can be applied to the file of conumbers instead of to the file of lexforms. In fact, the transformation of lexforms into conumbers preserves order, and so it can be applied without additional sorting. Experimental results indicate, however, that front compression of lexforms gives better results overall. If decoding and retrieval times are critical (as in real-time applications), then a hash-table method is advantageous. In this case front compression cannot be used and then the replacement of lexforms by conumbers (but without front compression) is preferable. The dictionary can be stored in an almost full hash table with a good average and worst case behaviour by using a method such as that of Schmidt and Shamir.[8]

(ii) Performs

A *permuted form* (*perform* for short) of a word $w = w_1 \ldots w_k$ is a permutation $w_{p(1)} \ldots w_{p(k)}$ of all the—not necessarily distinct—letters of w such that $w_{p(i)}$ precedes $w_{p(j)}$ if $w_{p(i)} \leq w_{p(j)}$. Informally, whereas a lexform is an ordered string of the distinct letters of w, a perform is an ordered string of all its letters. If a word $w = w_1 \ldots w_k$ maps into a perform $v = v_1 \ldots v_k$, then the *index* of w is a sequence of length k consisting of the numbers $1, \ldots, k$ such that if $w_i = v_j$, then the ith sequence number is j ($1 \leq i, j \leq k$).

The perform of any word w is at least as long as the lexform of w, and the numbers constituting the index of the perform of a word w are at least as large as the numbers constituting the index of the lexform of w. Moreover, normally fewer words map into the same perform than into the same lexform. Thus transforming dictionary words into performs and indexes will normally yield less compression than transforming words into lexforms. However, fewer indexes have to be checked per perform than per lexform, so decoding time for performs is somewhat shorter than for lexforms.

Analogously to phase B above, we may transform performs into conumbers (serial numbers of linearly ordered performs) and indexes into ranks. For a word of length k over an alphabet Σ with $|\Sigma| = n$, the number of distinct performs is evidently $\binom{n+k-1}{k}$, which is the number of k-combinations with repetitions. Thus the number of conumbers of performs is larger than the number of conumbers of lexforms. The number of indexes of words of length k with respect to performs, however, is at most $k!$. This is less than the number of indexes of lexforms, which was shown to be the number K_k of C-permutations. In fact, it is easy to verify that $(e/2)^k > 2\sqrt{(2\pi k)}\, e^{1/12k}$ for all $k \geq 9$. Hence by Stirling's formula,

$$k! < \sqrt{(2\pi k)} \left(\frac{k}{e}\right)^k e^{1/12k} < \frac{1}{2}\left(\frac{k}{2}\right)^k < \sum_{i=1}^{\infty} i^k 2^{-i-1} = K_k.$$

The fact that $k! < K_k$ also for $2 \leq k \leq 8$ is seen from Table 5.

The rank of an index with respect to a perform can be computed in one of the following ways:

(1) There is a one-to-one correspondence between permutations and their ranks based on the factorial representation of integers (see e.g. Ref. 1, p. 20). Algorithms realizing the transformations between permutations and their ranks are described by Pleszczynski.[9]

Table 5. No. of bits needed for indexes of lexforms and performs

	Perform		Lexform		
Length of word (k)	Number of possible indexes ($k!$)	Number of bits needed	Number of possible indexes (C-permutations)	Number of bits needed	Difference in number of bits needed
1	1	1	1	1	0
2	2	1	3	2	1
3	6	3	13	4	1
4	24	5	75	7	2
5	120	7	541	10	3
6	720	10	4683	13	3
7	5040	13	47293	16	3
8	40320	16	545835	20	4

Downloaded from https://academic.oup.com/comjnl/article/26/4/336/377426 by guest on 20 August 2022

(2) An ordered table of permutations can be consulted (up to size $k = 8$, say). The order of the table should be such that the $j!$ permutations of the first j symbols are generated before the $(j+1)$th symbol is moved, so that indexes of different lengths can use the same permutation table. Three algorithms with this property are compared by Roy.[10] (Two of them are the well-known algorithms of Ord-Smith[11] for generation of permutations in lexicographic and pseudo-lexicographic order. The third is due to Wells.[12]) An algorithm for permutation generation on vector processors with this property is given in Ref. 13.

To summarize, the use of performs yields less compression but gives slightly better decoding times than the use of lexforms.

4. EXPERIMENTS

In this section we give some results obtained by applying the method to the Responsa and NTIS dictionaries. We end with brief remarks on the decoding speed.

Phase A

Recall that in phase A every dictionary word is transformed into a lexform and a corresponding index. During this process, identical characters are deleted. Table 6 summarizes the data of the dictionaries used for the experiments. Note that about half the words contain equal characters, and the number of equal characters is about 11% of the total number of characters.

Let p_i be the probability of appearance of letter i in the dictionary $(1 \le i \le n = |\Sigma|)$. The 'amount of information' in the dictionary using the entropy measure is $H = -\sum_{i=1}^{n} p_i \lg p_i$. Since only about 11% of the characters are repeated, it seemed likely that the transformation from dictionary words to lexforms would not increase the entropy by much. This assumption was tested for the Responsa and NTIS dictionaries by computing the frequency of the different letters. The results are summarized in Table 7, which shows the entropy increase does not exceed 1.3%.

Table 6. Database overview

	Responsa	NTIS
Total number of words (all word lengths)	436490	56989
Total number of characters (all word lengths)	2656217	443672
Number of words (word lengths 3–8)	416661	35176
Number of characters (lengths 3–8)	2471545	210875
Number of words without equal characters (3–8)	212503	17046
Number of words with equal characters (3–8)	204158	18130
Number of repeated characters (3–8)	265948	24404
Percentage of repeated characters (3–8)	10.76	11.57

Table 7. Entropy of original dictionaries and lexforms

Entropy	Responsa	NTIS
Original dictionary	4.274	4.271
Lexforms	4.330	4.314

Table 8 exhibits some statistics on the lexforms. They show that the file of lexforms occupies only about 20% of the dictionary file of the Responsa; 56% for the NTIS dictionary. Further, the number of distinct lexforms is only about 20% of the number of distinct Responsa dictionary words; 60% for the NTIS dictionary. In order to find out whether these large differences are due to language idiosyncracies or to dictionary sizes, phase A was also run on a Hebrew dictionary of one of the Responsa books containing $d = 60\,636$ distinct words—only just larger than the NTIS dictionary. It turned out that the number of distinct lexforms was about 49% of d. This result indicates that the efficiency is primarily a function of the size of the dictionary, though the language does have an effect. In particular, the compression efficiency of the method increases markedly with dictionary size.

Table 8. Some statistics on lexforms

	Responsa	NTIS
Total number of lexforms	82770	20892
Total number of lexform characters	483451	118010
Number of lexforms / Number of words	19.86%	59.39%
Number of lexform characters / Number of word characters	19.56%	55.96%

The result of applying front compression to lexforms is shown in Table 9. It is assumed that a 4-bit string is adjoined to every lexform of length 3–5 to denote the length of the identical prefix; a 5-bit string for words of length 6–8. It is seen that front compression yields a

Table 9. Compression of lexforms by front compression

	Responsa				NTIS			
	$\|\Sigma\| = 32$		$\|\Sigma\| = 256$		$\|\Sigma\| = 32$		$\|\Sigma\| = 256$	
	Number of bits	Saving	Number of bits	Saving	Number of bits	Saving	Number of bits	Saving
Size of lexforms	2417255		3867608		590050		944080	
Lexforms after front compression	870925	64.0%	1393480	64.0%	242085	59.0%	387336	59.0%

Downloaded from https://academic.oup.com/comjnl/article/26/4/336/377426 by guest on 20 August 2022

Table 10. Some statistics on performs

	Responsa	NTIS
Total number of performs	186462	29724
Total number of characters	1175141	184960
Number of performs / Number of words	44.7%	84.5%
Number of perform characters / Number of word characters	47.5%	87.7%

relatively large saving. As stated earlier, however, it disables use of hashing, thus slowing down decoding. Table 10 is the analogue of Table 8 for performs. Note that the savings are considerably smaller than for lexforms.

Phase B

In phase B, lexforms are transformed into conumbers, and indexes into ranks. The amount of additional savings gained by this transformation depends on the size of the alphabet Σ: recall that each letter is represented by $\lceil \lg |\Sigma| \rceil$ bits only. The results are shown in Table 11.

Overall

The overall savings gained by transforming dictionary words into conumbers and ranks are exhibited in Table 12. It shows, in particular, that transforming words into conumbers produces a file $\mathfrak{C}$ which occupies only about 15% of the space required for the Responsa dictionary; 40–45% of the NTIS dictionary. If the ranks are kept in fast memory, only about 50% of the original Responsa dictionary space is needed; about 80% of the NTIS dictionary. More generally, the latter compression figures hold if both the lexforms and the ranks are stored on the same medium; either both in fast memory or both on disk. If the lexforms are in fast memory and the ranks on disk, we have to augment the ranks with another copy of the lexforms. A similar remark applies to the next and last compression results.

The results of applying phase A, replacing indexes by ranks and using front compression on the lexforms and on all words of length exceeding 8, are shown in Table 13. Note in particular that the $\mathfrak{C}$-file in fast memory occupies only 11% of the Responsa dictionary; 39% of the NTIS dictionary. If the ranks are also stored in fast memory, there is a saving of 48–63% for the Responsa dictionary; 40–48% for the NTIS dictionary.

Table 11. Additional savings gained by applying phase B

		Responsa	NTIS
Replacing lexforms by conumbers	$\lvert\Sigma\rvert = 32$	32.7%	32.0%
	$\lvert\Sigma\rvert = 256$	18.6%	18.2%
Replacing indexes by their ranks		27.6%	26.6%

Table 14. Timing results

	Time (s)
$\left\lceil \frac{r}{e} (\sqrt{(2\pi r)}\, M)^{1/r} \right\rceil$ for $M = 1000$, $M = 10^9$; and $r = 8$, $r = 13$ (all four combinations require about same time)	1.2×10^{-4} s
$\binom{m}{r}$	
$r = 3$; $m = 10^3$ or $m = 10^9$	4.3×10^{-5} s
$r = 8$; $m = 10^3$ or $m = 10^9$	1.2×10^{-4} s
Computing $\binom{m+1}{r}$ from $\binom{m}{r}$ by $\binom{m+1}{r} = \frac{m+1}{m-r+1}\binom{m}{r}$	3.8×10^{-6} s

Table 12. Savings achieved by phases A and B (word lengths 3–8)

	Responsa				NTIS			
	$\lvert\Sigma\rvert = 32$		$\lvert\Sigma\rvert = 256$		$\lvert\Sigma\rvert = 32$		$\lvert\Sigma\rvert = 256$	
	Number of bits	Saving	Number of bits	Saving	Number of bits	Saving	Number of bits	Saving
Original dictionary	12357725		19772360		1054375		1687000	
Conumbers	1626514	86.8%	3149617	84.1%	401336	61.9%	771992	54.2%
Ranks	5370160	56.5%	5370160	72.8%	464086	56.0%	464086	72.5%
Total	6996674	43.4%	8519777	56.9%	865422	17.9%	1236078	26.7%

Table 13. Overall compression by transforming dictionary words into lexforms with front compression, and indexes into ranks

	Responsa				NTIS			
	$\lvert\Sigma\rvert = 32$		$\lvert\Sigma\rvert = 256$		$\lvert\Sigma\rvert = 32$		$\lvert\Sigma\rvert = 256$	
	Number of characters	Saving	Number of characters	Saving	Number of characters	Saving	Number of characters	Saving
Lexforms and front compression on lexforms	291648	89.0%	291648	89.0%	171961	61.2%	171961	61.2%
Ranks	1074032	59.6%	671270	74.7%	92818	79.1%	58011	86.9%
Total	1365680	48.6%	962918	63.7%	264779	40.3%	229972	48.2%

Downloaded from https://academic.oup.com/comjnl/article/26/4/336/377426 by guest on 20 August 2022

We close with some timing data relevant to decoding. The algorithms were written in PL/1 and run on an IBM 370/165 computer. Some programs to compute the basic functions used in decoding such as $\lceil (r/e)(\sqrt{(2\pi r)}\,M)^{1/r} \rceil$, $\binom{m}{r}$ were run for timing purposes. Each program was run 10^6 times. The times given in Table 14 are the result of dividing the total time by 10^6. The table indicates that decoding is a fast process.

Acknowledgements

We wish to express our gratitude to Professor Y. Choueka, Head of the Institute for Information Retrieval and Computational Linguistics (IRCOL) at Bar Ilan University, for kindly placing at our disposal the Responsa database; to Mr. K. Keren, Head of the Israel National Center for Scientific and Technological Information (COSTI) who co-operated with us on the NTIS database experiments; to Messrs. A. Fullop, Y. Pechenik and E. Niovits at IRCOL; Mrs. I. Sered at COSTI and all other members of IRCOL and COSTI who helped us in various ways.

REFERENCES

1. D. E. Lehmer, The machine tools of combinatorics. In *Applied Combinatorial Mathematics* edited by E. F. Beckenbach, pp. 5–31, Wiley, New York (1964).
2. S. Even, *Algorithmic Combinatorics*, MacMillan, New York (1973).
3. M. Mor and A. S. Fraenkel, Cayley-permutations, *Discrete Math.*, in press.
4. A. S. Fraenkel, All about the Responsa retrieval project you always wanted to know but were afraid to ask. Expanded Summary, *Proc. Third Symp. on Legal Data Processing in Europe*, Oslo, 131–141 (1975). (Reprinted in *Jurimetrics J.* **16**, 149–156 (1976) and in *Informatica e Diritto* **II**, (3), 362–370 (1976).)
5. D. E. Knuth, *The Art of Computer Programming, Vol. 3—Sorting and Searching*, Addison-Wesley, Reading, MA (1973).
6. M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, National Bureau of Standards, June (1964). Ninth printing (1970).
7. D. Gotlieb, S. A. Hagerth, P. G. H. Lehot and H. S. Rabinowitz, A classification of compression methods and their usefulness for a large data processing center. *National Comp. Conference* **44**, 453–458 (1975).
8. J. Schmidt and E. Shamir, An improved program for constructing open hash tables. In *7th Colloquium on Automata, Languages and Programming* edited by J. W. de Bakker and J. van Leeuwen, pp. 569–581, July 14–18, Springer Verlag, Berlin (1980).
9. S. Pleszcynski, On the generation of permutations. *Information Processing Letters*, **3**, (6), 180–183 (1975).
10. M. K. Roy, Evaluation of permutation algorithms. *The Computer Journal* **21**, 296–301 (1978).
11. R. J. Ord-Smith, Generation of permutation sequences: Part 2. *The Computer Journal* **14**, 136–139 (1971).
12. M. B. Wells, *Elements of Combinatorial Computing*, Pergamon Press, Oxford (1971).
13. M. Mor and A. S. Fraenkel, Permutation generation on vector processors. *The Computer Journal* **25**, 423–428 (1982).

Received January 1983

Downloaded from https://academic.oup.com/comjnl/article/26/4/336/377426 by guest on 20 August 2022