COMBINATORIAL COMPRESSION AND PARTITIONING OF LARGE DICTIONARIES: THEORY AND EXPERIMENTS[1]

Aviezri S. Fraenkel[2,3] and Moshe Mor

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

Abstract 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 databases, in Hebrew and English, are presented in detail. 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.

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.

---

[1] 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.

[2] Partial affiliation with IRCOL.

[3] Supported in part by a grant of Bank Leumi Le'Israel.

Specifically, let $w = w_1w_2\cdots w_k$ be a word over a finite alphabet Σ, linearly ordered (under $<$). A lexicographic form (lexform for short) of w is a lexicographically ordered sequence $w_{q(1)}\cdots w_{q(\ell)}$ (for suitable $\ell \leq k$) 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 $w = w_1w_2\cdots w_k$ maps into a lexform $v = v_1v_2\cdots v_\ell$ ($\ell \leq k$), then the index of w is a sequence of length k consisting of the numbers $1,2,\cdots,\ell$, such that if $w_i = v_j$, then the i-th sequence number is j ($1 \leq j \leq \ell$, $1 \leq i \leq k$). 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 $\binom{n}{\ell}$. Since every combination can be represented by its serial number in some linear ordering of all combinations (see e.g. [LEH], [EVE]), 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 saving factor achieved by replacing dictionary words of length k by conumbers is at least $(2\pi k)^{-\frac{1}{2}}(ek^{-1})^k$ 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 (see [MOF2]). 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 $\underline{\underline{L}}$ of lexforms (or their corresponding conumbers) in fast memory, and the file $\underline{\underline{I}}$ 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 nonnumeric 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 $\underline{\underline{I}}$) reject the word. In both of these cases there are many accesses to $\underline{\underline{L}}$ in fast memory, and few accesses to $\underline{\underline{I}}$ 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 [FRA]. The subset contained some 114 million letters - excluding punctuation characters and blanks - 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 idio-

syncracies 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 $\lceil \ell g|\Sigma| \rceil$ 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 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 [KNU] 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 k-th word and I_k its index $(1 \leq k \leq d)$. The sort produces a sequence

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

In particular, $L_1 \leq \cdots \leq L_d$. Thereafter, all maximal blocks $(L_{k1},I_{k1}),\cdots,(L_{kt},I_{kt})$ 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 $(I_{k1},\cdots,I_{kt})$. The result is a sequence

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

where $r = |A|$ $(1 \leq r \leq d)$. 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 [KNU] may be used.

We now partition the sequence A into two sequences $\underline{\underline{L}} = \{L_1,\cdots,L_r\}$ of lexforms and $\underline{\underline{I}} = \{I_{11},\cdots,I_{1t(1)},\cdots,I_{r1},\cdots,I_{rt(r)}\}$ of indexes. The sequence $\underline{\underline{L}}$ can be stored in fast memory, $\underline{\underline{I}}$ on disk. No pointers from $\underline{\underline{L}}$ to $\underline{\underline{I}}$ are required if the lexforms are repeated in $\underline{\underline{I}}$, 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 $\binom{n}{\ell}$, where $n = |\Sigma|$. Instead or 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 $\binom{n}{\ell}$. 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 $\underline{\underline{L}}$ not in $\underline{\underline{I}}$. For the overall compression achieved, also $\underline{\underline{I}}$ must be considered. But since $\underline{\underline{I}}$ normally resides on disk, its storage is normally much cheaper than that of $\underline{\underline{L}}$.

Since $\binom{n}{\ell}$ grows rapidly with ℓ ($< n/2$), it is useful to consider only words of length $k \leq 8$, which holds for the majority of cases (see Table 7, Section 4). Longer words may be partitioned into segments of length ≤ 8.

Note that for fully utilizing 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 $\lceil \ell g\ n \rceil$ 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 saving factor gained by replacing dictionary words of length k by conumbers is at least $t = (2\pi k)^{-\frac{1}{2}}(ek^{-1})^k$ if $|\Sigma|$ is large.

PROOF. We use the following form of Stirling's formula [ABR 6.1.38]:

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

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

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} > \frac{1}{\sqrt{2\pi}} \frac{n^k e^{-((12k)^{-1} + (12(n-k))^{-1})}}{k^{k+\frac{1}{2}}(1-k/n)^{n-k+\frac{1}{2}}} \to \frac{1}{\sqrt{2\pi k}}\left(\frac{en}{k}\right)^k = tn^k \quad \text{as} \quad n \to \infty,$$

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

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 saving factor is at least t. ∎

Note that the saving factor is independent of $|\Sigma|$ as long as $|\Sigma|$ is large. Table 1 exhibits the savings projected by Proposition 1. The column headed by $-\ell g\ t$ gives the 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	$-\ell g\ t$
2	0.9
3	2.5
4	4.5
5	6.8
6	9.5
7	12.3
8	15.2

Table 2 displays several values of savings achievable for four values of n which are powers of 2. The table entries are also lower bounds of the savings, since the table assumes that lexforms have the same length as words. Comparing Table 1 with the penultimate column of Table 2, it is seen that the estimate of Proposition 1 is rather close to the actual lower bound for word-lengths 2-8. If the alphabet size

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

n	k	No. of bits for n^k	$\binom{n}{k}$	No. of bits for $\binom{n}{k}$	Possible savings (bits)	% of savings
32	2	10	496	9	1	1.0
32	3	15	4960	13	2	13.3
32	4	20	35960	16	4	20.0
32	5	25	201376	18	7	28.0
32	6	30	906192	20	10	33.3
32	7	35	3365856	22	13	37.1
32	8	40	10518300	24	16	40.0
64	2	12	2016	11	1	8.3
64	3	18	41664	16	2	11.1
64	4	24	635376	20	4	16.7
64	5	30	7624512	23	7	23.3
64	6	36	74794368	27	9	25.0
64	7	42	6.2122×10^{8}	30	12	28.6
64	8	48	4.4262×10^{9}	33	15	31.2
128	2	14	8128	13	1	7.1
128	3	21	341376	19	2	9.5
128	4	28	10668000	24	4	14.3
128	5	35	2.6457×10^{8}	28	7	20.0
128	6	42	5.4236×10^{9}	33	9	21.4
128	7	49	9.4526×10^{10}	37	12	24.5
128	8	56	1.4297×10^{12}	41	15	26.8
256	2	16	32640	15	1	6.2
256	3	24	2763520	22	2	8.3
256	4	32	1.7478×10^{8}	28	4	12.5
256	5	40	8.8095×10^{9}	34	6	15.0
256	6	48	3.6853×10^{11}	39	9	18.7
256	7	56	1.3162×10^{13}	44	12	21.4
256	8	64	4.0966×10^{14}	49	15	23.4

is not a power of 2, the savings by using conumbers are larger, because several possible characters are unutilized. This situation is shown in Table 3 for $n = 26$ and $n = 36$ (Latin alphabet supplemented by the digits 0-9, say).

Table 3 Same as Table 2, for two actual alphabet sizes

n	k	No. of bits for n^k	$\binom{n}{k}$	No. of bits for $\binom{n}{k}$	Possible savings (bits)	% of savings
26	2	10	325	9	1	10.0
26	3	15	2600	12	3	20.0
26	4	20	14950	14	6	30.0
26	5	25	65780	17	8	32.0
26	6	30	230230	18	12	40.0
26	7	35	657800	20	15	42.9
26	8	40	1562275	21	19	47.5
36	2	12	630	10	2	16.7
36	3	18	7140	13	5	27.8
36	4	24	58905	16	8	33.3
36	5	30	376992	19	11	36.7
36	6	36	1947792	21	15	41.7
36	7	42	8347680	23	19	45.2
36	8	48	30260340	25	23	47.9

For formulating transformations between a lexform and its conumber, define the <u>combinatorial representation</u> of any nonnegative integer N with respect to a fixed positive integer k, to be $(a_1,\cdots,a_k)$, where

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

subject to $0 \leq a_1 < a_2 < \cdots < a_k$ for uniqueness. See [LEH, p. 8].

A combination c out of a set of $\binom{n}{\ell}$ combinations is fixed by selecting ℓ positions $b_1,\cdots,b_\ell$ with $1 \leq b_1 < \cdots < b_\ell \leq n$ out of n positions. The conumber r $(0 \leq r < \binom{n}{\ell})$ of c is defined to be $\binom{n}{\ell} - \sum_{j=1}^{\ell}\binom{n-b_j}{\ell-j+1} - 1$ ([LEH, p. 28], [EVE, p. 33]). Conversely, the conumber of a combination c determines the positions $b_1,\cdots,b_\ell$: Given the conumber r of a combination out of $\binom{n}{\ell}$ combinations $(0 \leq r < \binom{n}{\ell})$, represent $R = \binom{n}{\ell} - r - 1$ in the combinatorial representation, that is, $R = \sum_{j=1}^{\ell}\binom{c_j}{\ell-j+1}$. Then $b_j = n - c_j$ $(1 \leq j \leq \ell)$ are the desired positions.

We now partition the set of lexforms into subsets, each containing lexforms of fixed length ℓ $(2 \leq \ell \leq 8)$. (Note that a subset containing lexforms of length ℓ is normally derived from words of various lengths $k \geq \ell$.) The lexforms in each subset are transformed into conumbers. The savings thus obtained are those estimated in Proposition 1 and Tables 1, 2 and 3.

Decoding involves computing the combinatorial representation. For computing the combinatorial representation of a nonnegative 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 inteter a_{k-2} satisfying $\binom{a_{k-2}}{k-2} \leq N - \binom{a_k}{k} - \binom{a_{k-1}}{k-1}$;$\cdots$. 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 $\binom{x}{r} \sim (2\pi r)^{-\frac{1}{2}}(exr^{-1})^r$ (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:

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

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

<u>PROOF</u>. 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 $r > 2$,

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

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,

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

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 4, which exhibits the values $x_0 - x_1$ and $x_2 - x_0$ for $1 \leq M \leq 3\times10^6$, $3 \leq r \leq 8$. It is seen that starting with x_1, at most r steps are required to get to x_0.

Table 4 The values $x_0 - x_1$ and $x_2 - x_0$ as a function of r for $1 \leq M \leq 3\times10^6$

x_0-x_1 \ r	3	4	5	6	7	8
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

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

PROOF. A C-permutation p of length k over $S = \{1,\cdots,k\}$ 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 $S = \{1,\cdots,k\}$. The result now follows since the number of C-permutations of length k over S is K_k [MOF2]. ■

The transformation between C-permutations and their ranks is effected by means of two algorithms given in [MOF2].

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 5 shows several savings achievable by replacing indexes by ranks. Note that this is a saving achieved in $\underline{I}$ rather than in $\underline{L}$.

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 stored 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) [GOT]. It is then natural to apply front compression also to all words of length exceeding 8.

Table 5 Savings achieved by using ranks instead of indexes

k	k^k	$\lceil \lg k^k \rceil$ No. of bits of k^k	K_k	$\lceil \lg K_k \rceil$ No. of bits of K_k	No. of bits saved	% of 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

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 behavior by using a method such as that of Schmidt and Shamir [SCS].

(ii) Performs. A permuted form (perform for short) of a word $w = w_1 \cdots w_k$ is a permutation $w_{p(1)} \cdots w_{p(k)}$ 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 $w = w_1 \cdots w_k$ maps into a perform $v = v_1 \cdots v_k$, then the index of w is a sequence of length k consisting of the numbers $1, \cdots, k$ such that if $w_i = v_j$, then the i-th 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 less 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, less 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 6.

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. [LEH, p. 20]. Algorithms realizing the transformations between

Table 6 No. of bits needed for indexes of lexforms and performs

	Perform		Lexform		
Length of word (k)	No. of possible indexes (k!)	No. of bits needed	No. of possible indexes (C-permutations)	No. of bits needed	Difference in no. 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

permutations and their ranks are described by Pleszcynski [PLE].

(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 [ROY]. (Two of them are the well-known algorithms of Ord-Smith [ORD] for generation of permutations in lexicographic and pseudo-lexicographic order. The third is due to Wells [WEL].) An algorithm for permutation generation on vector processors with this property is given in [MOF1].

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 7 shows the distribution of the dictionary words by their lengths, Table 8 presents the same thing for lexforms and Table 9 summarizes the data. 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 $(1 \leq i \leq n = |\Sigma|)$. The "amount of information" in the dictionary using the entropy measure is $H = -\sum_{i=1}^{n} p_i \log p_i$. 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 10, which shows that the entropy increase does not exceed 1.3%.

Table 11 exhibits the distribution of the distinct lexforms by length and then gives some overall figures. The latter 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

Table 7 Distribution of wordlengths in dictionaries

Wordlength	Responsa Dictionary		NTIS Dictionary	
	No. of words	%	No. of words	%
1	27	.006	-	-
2	496	.114	-	-
3	5844	1.34	2767	4.86
4	37736	8.64	4313	7.57
5	105870	24.26	5698	10.00
6	135588	31.06	7295	12.80
7	92793	21.26	7762	13.62
8	38830	8.90	7341	12.88
9	12927	2.96	6442	11.30
10	4068	.93	5114	8.97
11	1455	.33	3533	6.20
12	503	.12	2596	4.56
≥ 13	353	.08	4128	7.24
Total	436490	100	56989	100

Table 8 Distribution of lengths of lexforms (with repetitions)

Word length	Responsa Dictionary Length of lexform								No. of repeated characters
	1	2	3	4	5	6	7	8	
3	3	539	5302						545
4		249	6606	30881					7104
5		67	3145	30421	72237				36912
6		8	829	12217	54704	67830			81657
7			137	2617	18098	42080	29861		86675
8			12	378	3628	12179	16241	6392	53055
Total Distrib. of lexforms	3	863	16031	76514	148667	122089	46102	6392	265948
Word length	NTIS Dictionary Length of lexform								No. of repeated characters
	1	2	3	4	5	6	7	8	
3	9	336	2422						354
4		61	999	3253					1121
5		1	191	1778	3728				2163
6		1	53	709	2982	3550			4563
7			4	213	1417	3579	2549		7068
8			2	67	525	2079	3124	1544	9135
Total Distrib. of lexforms	9	399	3671	6020	8652	9208	5673	1544	24404

Table 9 Database overview

	Responsa	NTIS
Total no. of words (all word lengths)	436490	56989
Total no. of characters (all word lengths)	2656217	443672
No. of words (word lengths 3-8)	416661	35176
No. of characters (lengths 3-8)	2471545	210875
No. of words without equal characters (3-8)	212503	17046
No. of words with equal characters (3-8)	204158	18130
No. of repeated characters (3-8)	265948	24404
Percentage of repeated characters (3-8)	10.76%	11.57%

Table 10 Entropy of original dictionaries and lexforms

Entropy	Responsa	NTIS
Original dictionary	4.274	4.271
Lexforms	4.330	4.314

Table 11 Distribution of different lexforms by length

Length	Responsa		NTIS	
	No. of lexforms	%	No. of lexforms	%
1	3	.004	9	.04
2	228	.276	160	.76
3	1849	2.23	1295	6.20
4	7940	9.59	2802	13.41
5	20367	24.61	4671	22.36
6	27884	33.69	5920	28.34
7	19446	23.49	4567	21.86
8	5053	6.11	1468	7.03
Total no. of lexforms	82770		20892	
Total no. of lexform characters	483451		118010	
$\frac{\text{No. of lexforms}}{\text{No. of words}}$	19.86%		59.39%	
$\frac{\text{No. of lexform characters}}{\text{No. of word characters}}$	19.56%		55.96%	

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.

The result of applying front compression to lexforms is shown in Table 12. 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 relatively large saving. As stated earlier, however, it disables use of hashing, thus slowing down decoding. Table 13 is the analog of Table 11 for performs. Note that the savings are considerably smaller than for lexforms.

Table 12 Compression of lexforms by front compression

	Responsa				NTIS			
	$\|\Sigma\| = 32$		$\|\Sigma\| = 256$		$\|\Sigma\| = 32$		$\|\Sigma\| = 256$	
	No. of bits	Saving	No. of bits	Saving	No. of bits	Saving	No. 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%

Table 13 Distribution of performs by length

Length	Responsa		NTIS	
	No. of performs	%	No. of performs	%
3	2151	1.16	1435	4.83
4	10596	5.68	3010	10.12
5	32338	17.34	4520	15.21
6	55790	29.92	6393	21.51
7	54822	29.40	7271	24.46
8	30765	16.50	7095	23.87
Total no. of performs	186462		29724	
Total no. of characters	1175141		184960	
$\frac{\text{No. of performs}}{\text{No. of words}}$	44.7%		84.5%	
$\frac{\text{No. of perform characters}}{\text{No. of word characters}}$	47.5%		87.7%	

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 $\lceil \ell g|\Sigma| \rceil$ bits only. Table 14 gives the additional savings achieved when lexforms are transformed into conumbers. In this table, $N_2 = kN_1 \ell g\, n$, and $N_3 = N_1\lceil \ell g\binom{n}{k} \rceil$ (where $\binom{n}{k}$ and $\lceil \ell g\binom{n}{k} \rceil$ are listed in Table 2).

The result of replacing indexes by ranks is shown in Table 15: The entries in column N_1 are taken from Table 7. If word length is bounded by 8, each index digit can be represented by 3 bits, hence $N_2 = 3kN_1$. Also $N_3 = N_1\lceil \ell g\, K_k \rceil$, where $\lceil \ell g\, K_k \rceil$ is given in Table 5.

Overall. The overall savings gained by transforming dictionary words into conumbers and ranks are exhibited in Table 16. It shows, in particular, that transforming words into conumbers produces a file L which occupies only about 15% of the space required for the Responsa dictionary; 40-45% of the NTIS dic-

Table 14 Additional savings gained by replacing lexforms by conumbers

$n = |\Sigma| = 32$

k Length of lexform	Responsa N_1 No. of lexforms	Responsa N_2 No. of bits for lexforms	Responsa N_3 No. of bits for conumbers	NTIS N_1 No. of lexforms	NTIS N_2 No. of bits for lexforms	NTIS N_3 No. of bits for conumbers
1	3	15	15	9	45	45
2	228	2280	2052	160	1600	1440
3	1849	27735	24037	1295	19425	16835
4	7940	158800	127040	2802	56040	44832
5	20367	509175	366606	4671	116775	84078
6	27884	836520	557680	5920	177600	118400
7	19446	680610	427812	4567	159845	100474
8	5053	202120	121272	1468	58720	35232
Total	82770	2417255	1626514	20892	590050	401336
Additional savings		32.7%			32.0%	

$n = |\Sigma| = 256$

k Length of lexform	Responsa N_1 No. of lexforms	Responsa N_2 No. of bits for lexforms	Responsa N_3 No. of bits for conumbers	NTIS N_1 No. of lexforms	NTIS N_2 No. of bits for lexforms	NTIS N_3 No. of bits for conumbers
1	3	24	24	9	72	72
2	228	3648	3420	160	2560	2400
3	1849	44376	40678	1295	31080	28490
4	7940	254080	222320	2802	89664	78456
5	20367	814680	692478	4671	186840	158814
6	27884	1338432	1087476	5920	284160	230880
7	19446	1088976	855624	4567	255752	200948
8	5053	323392	247597	1468	93952	71932
Total	82770	3867608	3149617	20892	944080	771992
Additional savings		18.6%			18.2%	

tionary. 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 17. Note in particular, that the $\underline{L}$-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 15 Additional savings achieved by replacing indexes by their ranks

k Length of word	Responsa			NTIS		
	N_1 No. of indexes	N_2 No. of bits for indexes	N_3 No. of bits for ranks	N_1 No. of indexes	N_2 No. of bits for indexes	N_3 No. of bits for ranks
3	5844	52596	23376	2767	24903	11068
4	37736	452832	264152	4313	51756	30191
5	105870	1588050	1058700	5698	85470	56980
6	135588	2440584	1762644	7295	131310	94835
7	92793	1948653	1484688	7762	163002	124192
8	38830	931920	776600	7341	176184	146820
Total	416661	7414635	5370160	35176	632625	464086
Additional savings		27.6%			26.6%	

Table 16 Savings achieved by phases A and B (word lengths 3-8)

	Responsa				NTIS			
	$\|\Sigma\| = 32$		$\|\Sigma\| = 256$		$\|\Sigma\| = 32$		$\|\Sigma\| = 256$	
	No. of bits	Saving	No. of bits	Saving	No. of bits	Saving	No. 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 17 Overall compression by transforming dictionary words into lexforms with front compression, and indexes into ranks

	Responsa				NTIS			
	$\|\Sigma\| = 32$		$\|\Sigma\| = 256$		$\|\Sigma\| = 32$		$\|\Sigma\| = 256$	
	No. of characters	Saving	No. of characters	Saving	No. of characters	Saving	No. 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%

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 $\lceil \frac{r}{e} (\sqrt{2\pi r}\, M)^{1/r} \rceil$, $\binom{m}{r}$ were run for timing purposes. Each program was run 10^6 times. The times given in Table 18 are the result of dividing the total time by 10^6. The table indicates that decoding is a fast process.

Table 18 Timing results

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

ACKNOWLEDGEMENT

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 cooperated with us on the NTIS database experiments; to Messers 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

[ABR] M. Abramowitz and I.A. Stegun, Handbook of Mathematical Functions, National Bureau of Standards, June, 1964 (Ninth printing, 1970).

[EVE] S. Even, Algorithmic Combinatorics, MacMillan, New York, N.Y., 1973.

[FRA] 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, 1975, 131-141. (Reprinted in Jurimetrics J. 16 (1976), 149-156 and in Informatica e Diritto II, No. 3 (1976), 362-370.)

[GOT] 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 (1975), 453-458.

[KNU] D.E. Knuth, The Art of Computer Programming, Vol. 3 - Sorting and Searching, Addison-Wesley, Reading, MA, 1973.

[LEH] D.E. Lehmer, The machine tools of combinatorics, in: Applied Combinatorial Mathematics (E.F. Beckenbach, Ed.), J. Wiley, New York, N.Y., 1964, 5-31.

[MOF1] M. Mor and A.S. Fraenkel, Permutation generation on vector processors, The Computer Journal 25, 4 (November, 1982), 423-428.

[MOF2] M. Mor and A.S. Fraenkel, Cayley-Permutations, Discrete Math., in press.

[ORD] R.J. Ord-Smith, Generation of permutation sequences: Part 2, The Computer Journal 14 (1971), 136-139.

[PLE] S. Pleszcynski, On the generation of permutations, Information Processing Letters 3, 6 (July, 1975), 180-183.

[ROY] M.K. Roy, Evaluation of permutation algorithms, The Computer Journal 21, 4 (November, 1978), 296-301.

[SCS] J. Schmidt and E. Shamir, An improved program for constructing open hash tables, in: 7th Colloquium on Automata, Languages and Programming (J.W. de Bakker and J. van Leeuwen, Eds.), July 14-18, 1980, Springer Verlag, Berlin, 569-581.

[WEL] M.B. Wells, Elements of Combinatorial Computing, Pergamon Press, Oxford, 1971.