Alphabet Permutation for Differentially Encoding Text

Gad M. Landau
Ofer Levi
University of Haifa

Steven Skiena*
SUNY Stony Brook

May 21, 2004

1 Introduction

One degree of freedom which is usually not exploited in developing high-performance text-processing algorithms is the encoding of the underlying atomic character set. Typically, standard character encodings such as ASCII or Unicode are assumed to be a fixed fact of nature, and indeed for most classical string algorithms the assignment of exactly which symbol maps to which k-length bit pattern appears to be an issue of no consequence.

In this paper, however, we consider a text compression method where the specific character set collating-sequence employed in encoding the text has a big impact on performance. We demonstrate that permuting the standard character collating-sequences yields a small win on Asian-language texts over *gzip*. We also show improved compression with our method for English texts, although not by enough to beat standard compression methods. However, we also design a class of artificial languages on which our method clearly beats *gzip*, often by an order of magnitude.

The significance of this work lies partially in evaluating an interesting approach to text compression. Even more, however, we seek to raise awareness of character encodings in the string-algorithms community and ask the question whether alphabet-permutation can lead to improvements in other string and text-processing algorithms.

2 Differential Encoding

Differential coding is a common preprocessing step for compressing numerical data associated with sampled signals and other time series streams. The temporal coherence of such signals

*Corresponding author: skiena@cs.sunysb.edu

implies that the value at time t_i likely differs little from that at t_{i+1}. Thus representing the signal as an initial value followed a stream of difference (i.e. $t_{i+1} - t_i$ for $0 \leq i < n$) should consist primarily of small differences. Such streams should be more compressible using standard techniques like run-length encoding, Huffman coding, and gzip than the original data stream.

Here we consider differential encoding of text by treating each character code as an integer. By taking the differences modulo the size of the alphabet, we can ensure that they can always be encoded using the same number of bits as the original character symbols.

Under what conditions might such a differentially encoded text T' be more compressible than the original straight text T? Let $w = s_1 s_2 \ldots s_k$ be a string of length k which occurs multiple times in T. We note that $w' = \delta_1 \delta_2 \ldots \delta_{k-1}$ occurs the same number of times in T', where $\delta_i = s_{i+1} - s_i$. Since w' is shorter by one character that w, differential encoding might seem inherently counter-productive to the goals of higher compression.

However, there are two potential benefits. First, the string w' in T' may arise from several different strings within T, whenever the strings have a common shift pattern. A well-known example is the collision of the suffixes of “IBM” and “HAL” in a differential encoding using the ASCII collating sequence [5]. Second, with the proper collating sequence we would expect to have a skewing in the distribution of symbols toward those representing smaller differences.

It is impossible to tell a priori whether differential encoding with alphabet permutation will lead to improved compression on any given language. For this reason, we report experimental results in the sections to follow.

The most relevant previous work is [3], where alphabet permutation was employed to improve the performance of compression algorithms based on the Burrows-Wheeler transform [2]. Previous work on differential coding for text compression includes [8, 9]. Our work goes farther in our efforts to optimize the alphabet permutation, extending the results to Asian language encoding, and building a theory of languages for which differential coding will be effective.

3 Experiments on English Texts

The key to successful differential encoding of English (or any other class of languages) lies in identifying the best collating sequence. However, designing the optimal character permutation is a non-trivial problem. In principle, we seek the order which most frequently collapses popular substrings into identical sequences of differences. However, this criteria is not well-defined and does not lend itself to local improvement-based optimization.

Instead, we seek an ordering which minimizes the expected size of the differences. Such an ordering would be expected to skew the distribution of symbols towards those representing small differences, which would clearly improve the performance of zeroth-order entropy compression algorithms such as Huffman codes. We would also expect that repeated difference sequences would occur more frequently in collating sequences which lead to a skewed symbol distribution.

	Original			Differential		Permuted-Diff	
file	size	gzip	huffman	gzip	huffman	gzip	huffman
book1	767476	286727	434221	322455	489396	321347	472108
book2	598615	182574	339493	203269	381969	202812	369285
paper1	51093	15709	28896	17600	32500	17548	31447
paper2	81860	27375	46724	30440	52328	30350	50508
paper3	45997	16464	26395	18519	29575	18467	28565
paper4	13032	4826	7556	5435	8455	5397	8211
paper5	11730	4236	6660	4824	7508	4800	7304
paper6	36483	10942	20096	12219	22950	12169	22200
news	365318	118124	207541	133529	231233	133192	227751
gesture	44893	14910	25160	16756	28541	16717	27673
bib	104812	28833	59255	31834	66380	31694	65815
trans	73923	12734	39746	14227	45292	14207	44844
progc	38348	10760	20187	11948	23086	11920	22795
progp	48266	9290	24440	10414	27954	10400	27694
progl	71213	14620	36878	16230	42168	16209	41759
12sad10	424342	154122	237963	173374	270706	172830	260760

Table 1: Effect of Optimized Permutation on Differential Coding Methods.

Therefore, we seek the circular n-permutation π which minimizes the objective function

$$\min_{\pi \in \Pi} \sum_{i=1}^{n} \sum_{j=1}^{n} d(i,j) p(\sigma_i, \sigma_j)$$

where $p(i,j)$ is the probability that symbol j immediately follows symbol i, i.e. $p(i,j) = P(j|i)$, and $d(i,j)$ is the shortest "distance" from i to j around the circular permutation. Thus $d(i,j) = \min(|j-i|, n-|j-i|)$. This is an instance of the notorious *quadratic assignment problem*, an optimization problem significantly harder in practice than the traveling salesman problem [1]. If unweighted by probabilities, permutation optimization is related to the *linear assignment* problem [4], which although NP-complete under very restrictive conditions is managable in practice through heuristics [7]. Alterate optimization criteria are no doubt possible, but this is the one we used.

To estimate the conditional character-probabilities for the optimized collating sequence for English text, we used letter-pair (bigram) frequencies derived from a large corpus of text analyzed in [12], including the famous Brown corpus. The *Discropt* [10, 11] system was run for 10 hours optimizing the permutation over these frequencies, resulting in the following collating sequence:

```
. V G W C D I N H E T ' ' S A R O L F M P U Y B J Q Z X K
```

Table 1 compares differential compression using both the standard and optimized collating sequence, with both standard Huffman codes and gzip employed for encoding. The

	Original	Permuted		Permuted-Diff	
file	size	gzip	huffman	gzip	huffman
book1	768260	303884	394003	347211	466801
book2	599399	194201	313925	221163	362138
paper1	51877	16751	27181	19084	30970
paper2	82644	28902	42112	32848	49727
paper3	46781	17471	24037	19900	28040
paper4	13816	5086	6888	5764	8061
paper5	12514	4576	6402	5186	7093
paper6	37267	11652	19294	13251	21734
news	366102	127102	207326	144729	224090
gesture	45677	15884	23738	18241	26992
bib	105596	30372	58164	34151	65152
trans	74707	13762	42218	15465	45317
progc	39132	11641	21428	13242	22926
progp	49050	10205	26732	11589	28067
progl	71997	15796	39840	17839	42033
12sad10	425126	164272	220261	188444	257470

Table 2: Effect of Elias Predictive Coding on Differential Coding Methods.

permuted collating sequence typically reduces the size of the Huffman-encoded differential sequences by 3-4%, and gzip-encoded differential sequences by about 1% – however, both encoding algorithms work substantially better on the original text instead of the differential text.

The use of a single fixed collating sequence for all characters does not effectively capture the second-order entropy of the language, because the symbol distribution following each letter of the alphabet is distinct. In Table 2, we identify the best symbol permutation following each character, for each file separately; a method akin to a static version of Elias predictive coding [6]. Such a method produces better compression rates for differential encoding on sufficiently large files, even with the cost of storing the character permutation matrix. Interestingly, although Huffman codes work better on the Elias coded files, gzip does significantly worse than on the original file.

4 Experiments on Asian-Language Texts

We reasoned that differential encoding might perform better on Asian-language texts, because the larger size of the alphabet makes such texts more closely resemble quantized signals. However, accurately measuring the character bigram frequencies in Asian-language texts is made difficult by the enormous size of the alphabet. Vast amounts of training text would be needed to estimate the $(2^{16})^2$ bigram-pairs of 16-bit UNICODE. Further, quadratic assignment problems of such size are intractable to solve.

For this reason, we chose a different method to construct the alphabet permutation for 16-bit UNICODE. We assume that symbol usage in any fixed-length alphabet obeys a Zipf's law-type distribution, so frequently-used symbols are much more popular than expectation. Our permutation was derived by determining the symbol frequencies for all characters over all documents, and ordering the symbols in order of decreasing frequency. Since the most popular symbols are located near each other in the code space, we would anticipate that such an ordering would lead to smaller average differences over arbitrary encodings.

Table 3 details the results of our experiments on Chinese, Japanese, and Korean UNICODE texts. We experimented with both 8-bit and 16-bit recoded alphabets. The 8-bit alphabet permutation produced worse results than the original alphabet encoding for both gzip and Huffman codes, but permuting the full 16-bit alphabet encoding did permit the differential gzip encoding to beat the conventional gzip encodings by 1-2% on almost all files.

5 Experiments on Martian-Language Texts

To demonstrate that gzip can be significantly beaten via differential encoding for certain languages, we define a class of artificial languages which we will call *Martian*.

Martian words evolve in *families*. Each family is defined by a length-$(l-1)$ sequence of differences from 0 to $\alpha - 1$, where $\alpha = |\Sigma|$ and Σ is the length of the alphabet. There are α distinct length-l words in each family, formed by prepending each $\sigma \in \Sigma$ to the difference sequence. For example, for $\Sigma = \{a, \ldots, z\}$ the family $(+2, +3, -6)$ defines the words *acfz*, *bdga*, *cehb*, and so forth.

In the experiments below, we compress randomly generated Martian texts constructed with the following parameters:

- The alphabet size is α, ranging from 2 to 256.
- The number of word families is f. Word families were generated by sampling uniformly at random from the α possible differences. No effort was made to ensure that words would be part of at most one family.
- The length of each each word is l.
- The number of words in the text is n. The random text was generated by sampling uniformly with replacement from the αf words in the language.

We achieve our greatest improvement in differentially encoding Martian texts with relatively short texts drawn from large families of long words. Table 4 demonstrates that differential encoded gzip results in 5.8 times better compression than plaintext gzip on files from 2500 to 50,000 words for 20 families of 20-character words. Even more extreme performance is obtainable by further lengthening the words.

	Original			8-bit encoding		16-bit encoding		
file	size	gzip	huff	diff-gzip	diff-huff	huff	diff-huff	perm-gzip
ChuanXiLu1	100904	30545	57307	32774	73389	33828	54734	29953
ChuanXiLu2	129750	39753	73804	42908	94946	43794	70785	39054
ChuanXiLu3	96894	30314	55210	32782	70839	33577	53859	29779
dai	83100	40318	64540	51850	70611	41203	73290	38544
RenXue	229590	76878	130902	83277	167918	78475	126092	75810
XinMinShuo	357138	115668	202585	125114	259954	118312	190772	114354
Xunzi	88094	46182	69361	58050	76014	46089	82352	44296
ZhengMeng	66212	35033	51465	44044	56086	34586	62659	33165
ZhouDunyiJi	85646	26487	48782	28778	62153	29773	48315	25968
ZhuziYulei1-6	297888	87767	167904	93602	216643	95531	156285	86801
ZhuziYulei14-18	446066	129448	252811	137478	325836	140393	231406	128122
ZhuziYulei7-13	273836	85480	155937	91431	199607	89439	145787	84477
Gan	156494	68335	106217	87897	117985	78332	114737	67514
Goju-no-to	114108	60709	83634	77433	91324	64086	98553	59781
Hojoki	22936	10779	15127	13595	16861	12357	17786	10626
horoki	420774	166957	277984	211118	294482	193615	264312	165275
Jigokuhen	58226	24460	39741	31198	43952	30591	45156	24083
Kageronikki	202632	76612	112690	100539	129172	80347	106842	76135
kaidoki	68158	31979	46631	39928	50387	36185	55050	31270
KanadehonChushingura	164122	63883	110045	79724	115361	74324	111094	62601
Kappa	84092	32524	57883	41222	62053	43587	61638	32031
Kokoro	339050	137499	233746	178618	256945	163798	232484	135906
KoshokuGoninOnna	82240	41057	57557	52177	63581	43006	67447	40119
KoshokuIchidaiOnna	125278	56263	86343	71393	95161	63396	96308	55020
Makura-no-soshi	276342	117992	178011	154409	199284	126817	179134	116691
Midaregami	45768	15051	26696	19039	27827	18506	27133	14719
Monogatari	63878	21782	36172	28288	40890	24498	33777	21615
MurasakiShikibu-nikki	77972	33616	51395	43627	57087	37939	55137	33280
OkuNoHosomichi	31430	15878	22434	19329	24147	18299	28551	15453
SankaWakashu	178138	60699	107901	78642	114954	70872	101243	59562
Shayo	207568	78946	133906	101544	147648	96732	133419	78025
SonezakiShinju	41170	17357	28388	21461	29981	21693	32848	17008
Taketori	40762	17643	27875	22832	30886	21292	32271	17444
Tsurezuregusa	145156	62583	95940	80163	106215	69703	101209	61560
UgetsuMonogatari	102562	52121	70229	65919	78896	55063	82895	51204
Ukigumo	259698	114305	182091	145127	196343	132205	186899	112534
cjk	51172	25178	41410	31152	44671	28878	51207	24377
Ijangui	22240	13667	18431	16459	19854	14902	27781	13097
zuochuan-sjis	95666	34710	54191	45093	60316	38822	48472	34872

Table 3: Effect of Optimized Permutation on Differential Coding Methods, for Chinese, Japanese, and Korean Texts

			$\alpha = 32$		$\alpha = 64$		$\alpha = 128$		$\alpha = 256$	
f	l	*size*	gzip	diff-gzip	gzip	diff-gzip	gzip	diff-gzip	gzip	diff-gzip
5	5	2500	4584	3687	5174	4169	6445	4607	8606	5132
5	5	5000	9059	7229	9680	8162	11080	8948	14061	9958
5	5	10000	18000	14259	18879	16144	20371	17604	23782	19391
5	5	25000	44110	35131	45993	39929	47751	43458	52756	47395
5	5	50000	87358	69896	90828	79579	92981	86511	100849	93925
5	10	2500	5607	4171	6745	4601	9672	5021	14498	5500
5	10	5000	10435	8155	11380	8983	14691	9776	21855	10647
5	10	10000	19860	16030	20645	17691	24505	19193	35454	20823
5	10	25000	47754	39603	48407	43767	53932	47344	76321	51168
5	10	50000	94066	78884	94701	87275	103071	94267	144064	101737
5	20	2500	7184	4814	10184	5351	16414	5761	27385	6154
5	20	5000	12253	9358	15792	10492	25493	11308	46488	12032
5	20	10000	22375	18447	26880	20742	43485	22351	84336	23768
5	20	25000	52726	45644	60333	51458	97928	55429	198693	58950
5	20	50000	103345	90982	115986	102755	188780	110577	389213	117645
10	5	2500	5132	4244	6114	4602	7875	5115	10165	5537
10	5	5000	9851	8292	10859	8940	13329	9893	17403	10849
10	5	10000	19630	16402	20437	17574	23174	19340	29305	21153
10	5	25000	48429	40476	48646	43384	52418	47383	63399	51613
10	5	50000	96041	80643	95232	86327	100784	94021	120058	102085
10	10	2500	6529	4616	8917	5024	13038	5497	18429	5931
10	10	5000	11344	8987	13972	9745	20137	10610	30120	11525
10	10	10000	20992	17638	23915	19099	33315	20734	52755	22497
10	10	25000	50018	43489	53776	47051	72654	50927	120689	55366
10	10	50000	98363	86607	103531	93611	138149	101239	233663	110140
10	20	2500	9534	5347	14726	5791	24297	6194	36224	6637
10	20	5000	15188	10352	23255	11236	41580	11991	65932	12830
10	20	10000	26404	20328	40508	22077	76069	23561	126013	25224
10	20	25000	59834	50240	92024	54594	180190	58263	305697	62485
10	20	50000	115555	100143	177859	108741	353604	116089	605099	124485
20	5	2500	5836	4741	7204	5113	9137	5542	11250	5948
20	5	5000	10755	9214	12646	9885	16049	10771	20421	11656
20	5	10000	20740	18123	22688	19295	27742	20972	36038	22835
20	5	25000	50222	44628	52318	47263	61265	51167	81557	55802
20	5	50000	98754	88708	101124	93778	117056	101301	157107	110648
20	10	2500	8352	5134	11695	5536	16410	5965	21301	6376
20	10	5000	13582	9899	18602	10619	27224	11486	37950	12365
20	10	10000	23701	19293	31375	20613	48249	22294	70462	24247
20	10	25000	54079	47374	69759	50462	110791	54630	168058	59676
20	10	50000	104742	93830	133662	100145	215310	108519	330452	118582
20	20	2500	13564	5888	21528	6312	31968	6745	42379	7198
20	20	5000	21703	11312	37090	12108	58743	12929	81159	13791
20	20	10000	38051	22116	68420	23649	111898	25271	158012	26970
20	20	25000	87213	54422	161892	58257	271528	62320	389073	66646
20	20	50000	168542	108284	318406	115932	538072	124003	775298	132668

Table 4: Comparing gzip and Differential-gzip Compression of Martian Language Texts, as a Function of Family Size, Word Length, and Text Length.

Differential gzip outperforms normal gzip because the coherence of words in a family is unrecognizable by gzip until each word has been seen enough times to be encoded by gzip as a single symbol. The advantage of differential gzip on Martian texts will disappear once all words in each family are defined by code strings, but this takes much longer with the plaintext gzip.

Experiments with several random permutations of the Martian alphabet confirms that there is no benefit from differential encoding unless the correct character ordering is used. Determining the optimal ordering (or even a good one) from a text corpus is presumably an intractable problem. This leaves the theoretical (but highly unlikely) possibility that English text can be efficiently differentially compressible were only the correct alphabet permutation were found.

Acknowledgments

We thank Danny Keren for his insight that coding techniques for quantized signals can be applied to strings, Dafna Sheinwald for helpful discussions, and Vihnthuy Phan and Pavel Sumazin for optimizing the English-alphabet permutation using Discropt.

References

[1] R. Burkard, E. Cela, P. Pardalos, and L. Pitsoulis. *The Quadratic Assignment Problems*, pages 241–238. Kluwer, 1998.

[2] M. Burrows and D. Wheeler. A block-sorting lossless data compression algorithm. Digital Systems Research Center Research Report 124, 1994.

[3] B. Chapin and S. Tate. Higher compression from the burrows-wheeler transform by modified sorting. In *IEEE Data Compression Conference*, 1998.

[4] P. Chinn, J. Chvátolvá, A. K. Dewdney, and N. E. Gibbs. The bandwidth problem for graphs and matrices – a survey. *J. Graph Theory*, 6:223–254, 1982.

[5] A. C. Clarke. *2001: A Space Odyssey*. Roc, reissue edition, 2001.

[6] P. Elias. Interval and recency rank source coding: two on-line adaptive variable-length schemes. *IEEE Trans. on Information Theory*, 33:3–10, 1987.

[7] N. Gibbs, W. Poole, and P. Stockmeyer. A comparison of several bandwidth and profile reduction algorithms. *ACM Trans. Math. Software*, 2:322–330, 1976.

[8] D. Gottlieb, S. Agerth, P. Lehot, and H. Rabinowitz. A classification of compression methods and their usefulness for a large data processing center. *National Computer Conference*, 44:453–458, 1975.

[9] M. F. Lynch. Compression of bibliographic files using an adaptation of run-length coding. *Information Storage and Retrieval*, 9:207–214, 1973.

[10] V. Phan and S. Skiena. An improved time-sensitive metaheuristic optimizer. In *3rd Int. Workshop on Experimental and Efficient Algorithms (WEA)*, Buzios, Brazil, 2004.

[11] V. Phan, S. Skiena, and P. Sumazin. A time-sensitive system for black-box combinatorial optimization. In *Workshop on Algorithm Engineering and Experimenation (ALENEX '02)*, 2002.

[12] H. Rau and S. Skiena. Dialing for documents: an experiment in information theory. *Journal of Visual Languages and Computing*, pages 79–95, 1996.