Parameters of Dynamic Markov Compression
by Gary D Orser

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science
Montana State University

Abstract:
Dynamic Markov Coding (DMC), a lossless compression technique, can be used to compress graphics images better than either dictionary (LZ) or context modeling (PPM). By searching the parameter space of DMC, parameters can often be found which both increase compression performance and substantially reduce the memory required for compression and decompression.

PARAMETERS OF DYNAMIC MARKOV COMPRESSION

by

Gary D. Orser

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana
May 1996

APPROVAL

of a thesis submitted by

Gary D. Orser

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

4/25/96
Date

Chairperson, Graduate Committee

Approved for the Major Department

4/25/96
Date

Head, Major Department

Approved for the College of Graduate Studies

5/6/96
Date

Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under the rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signature Mary D. Dorn

Date 5/7/96

TABLE of CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Dynamic Markov Coding (DMC), a lossless compression technique, can be used to compress graphics images better than either dictionary (LZ) or context modeling (PPM). By searching the parameter space of DMC, parameters can often be found which both increase compression performance and substantially reduce the memory required for compression and decompression.

INTRODUCTION

The original impetus for this thesis topic, was the controversy that erupted in December of 1994 over the enforcement of a compression patent used in the graphics interchange format (GIF). At the time, GIF was the prevalent format used in the electronic distribution of images. This resulted in a storm of discussion in the online community about the patent, and a replacement for GIF. The discussion about replacements led to the question, "What was the best lossless compression method for graphics images?" This thesis focuses on the lossless compression of graphics images. In particular, the compression performance of Dynamic Markov Coding [1] applied to graphic images.

SURVEY of COMPRESSION METHODS

Lossless data compression is divided into two processes, coding and modeling. Coding is the process of encoding a given symbol with a given probability of occurrence in the fewest numbers of bits. Coding is approaching its theoretical limits, the entropy of the model. Modeling is the process of determining the probabilities for each symbol that the coding process is to encode. The theoretical limits of modeling are not well characterized. A model can range from perfect to completely imperfect. Every symbol is predicted with P(symbol) = 1.0 or every symbol is randomly distributed. The challenge is the efficient construction of good models.

Current effective lossless data compression modeling can be divided into two general categories: parsing compressors, such as dictionary based (LZ family); and statistical compressors, such as prediction by partial match (PPM), and state based (DMC).

It has been generally shown that in principle, these schemes are theoretically equivalent in their compression capabilities. This assumes an ergodic Markov source. Bell and Witten [3] showed that from any non-adaptive greedy parsing method (LZ) an equivalent symbolwise method(PPM) can be constructed. Bell and Moffat [2] demonstrated that DMC was a Finite Context

Automata (FCA), which is equivalent to variable order Markov models (PPM). However, more recently, Bunton [4]showed that while DMC was a FCA, an important distinction from all FSMX models was DMC's capacity for variable length extensions of contexts. FSMX models are only capable of single character minimal extensions, so DMC theoretically has more power than the PPM models. "DMC models reflect higher-order statistics of the conditioning contexts themselves. In contrast, the structure of FSMX models only reflects the zeroth order statistics of the conditioning contexts." [4]

Since the literature suggests that DMC is theoretically as good as or better than the other competitive lossless compression algorithms, DMC is an appropriate algorithm to investigate. Additionally, DMC's bit oriented approach may be suitable for a source (images) that is generated in a bit oriented fashion.

OVERVIEW of DMC

Dynamic Markov Coding (DMC) [1] is one particular compression method in a family of compressors that use an adaptive statistical model with an arithmetic coder. This family provides the best known lossless compression. It combines arithmetic coding with an adaptive state model. The adaptive state model "assumes only that the source data is a stream of bits generated by a discrete-parameter Markov chain model." These assumptions are also made in the LZ and the PPM techniques.

DMC compresses an input file as a stream of bits in network order. The symbols are {0,1}. Each symbol is arithmetically coded with the bit and a probability, P(bit|current state). The probability is provided by a finite state machine (FSM). After each bit is coded, the state machine is updated to reflect the last bit received. Each node in the FSM represents a state. A state can transition to one of two other states, depending on whether {0} or {1} was received. Associated with each transition is a probability of making that transition. The estimated probability is the ratio of previous transitions recorded in this state. This is an adaptive model. The model is changed after every bit. The decompression process is analogous to the compression process. The decompression process starts with the same initial model as the compression

process. It reverses the arithmetic code, generating a {0} or a {1} bit. It then updates the model with the same process as the compressor. The decompressor's model is reconstructed in step, just as the compressor's model was originally created.

The pseudo code for DMC is:

(1) initialize state machine

(2) for each bit:

    (3) arithmetically code current bit

    (4) update state machine

(5) end of for each bit

(6) final clean up

(1) Initialize state machine: Cormack and Horspool [1] discussed two starting models for their state machine. The first, and the simplest, consists of one state with transitions to itself. The second was a structure they called a braid. This is 256 trees. Each tree is 8 bits deep. Each leaf node, representing one value of an 8 bit byte, connects to the top of the corresponding tree that also represents the same value of that 8 bit byte. This structure improves compression performance when the data is from a known 8 bit source. All initial

transition probabilities are set to 0.5. The zero transition is as likely as the one transition.

(3) Arithmetically code current bit: Arithmetic coding encodes bits in the real interval [0-1). If a zero is to be encoded, the left edge of the interval is moved to the right. If a one is to be encoded, the right edge of the interval is moved to the left. The amount that the interval is reduced is dependent on the probability of the bit. If the probability of the bit is high, the interval is reduced a small amount. Conversely, if the probability of the bit is low, the interval is reduced a large amount. This implementation maintains the interval within a 24 bit window. Anytime the upper 8 bits are the same between upper and lower interval boundaries, those 8 bits are output, and the window is re-scaled to a 24 bit window.

(4) Update state machine: Each state consists of two counts and two state transitions, one each for the zero bit and the one bit. The count is incremented each time a given transition occurs. P(bit|current state) is defined as the ratio of the count of the transitions to the particular state to the count of the total transitions from this state, (count(bit) / (count(0) + count(1)). New nodes are created when the count of a transition in a particular state exceeds a threshold, and, when the transitions from the next state exceed another threshold. Cormack and Horspool [1] experimented with these thresholds. Higher thresholds decrease the number of nodes created, but decreased

compression efficiency. The lowest values for these thresholds, both equal to two, create the most nodes, but have the greatest compression efficiency. This result was not proved but demonstrated experimentally. When a new node is created the counts in the next node are apportioned between the new node and the next node based on the ratio of transitions to the next node and transitions out of the next node.

(6) Final cleanup: is simply outputting the 3 bytes that are left in the 24 bit window. As a result, the smallest possible compressed file is at least 3 bytes.

EXPERIMENTAL RESULTS

For graphics images a compression corpus exists at the home of the PNG (Portable Network Graphics) specification. This specification is intended to replace GIF for lossless graphics formats. The PNG specification, in addition to specifying file formats, currently specifies LZ77, dictionary modeling with Huffman coding as its compression method. LZ77 and Huffman coding are not patented. The corpus contains a wide variety of graphics images that can be used to test the efficiency of compression methods for graphic files. All files were converted to standard uncompressed pixel files as defined by the Pbmplus package. This is a byte oriented platform independent standard. There are red, green, and blue bytes for each pixel defined in the image. All images are rectangular.

Table 1. Graphics compression corpus

anemone	722x471 true color sea anemone & some fish
carlsbad	16-bit gray scale converted from USGS
coral	15-bit color photo or live coral polyps
face1	24-bit color photo of woman's face
fall	15-bit color image of Autumn landscape.
figures	24-bit color photo of man and woman standing
haiti	16-bit gray scale, converted from USGS
house2	24-bit photo of side of house
madagas	16-bit gray scale, converted from USGS
neptune1	24-bit color photo Neptune with black sky background
neptune2	24-bit color photo of planet Neptune (closer up)
shark	15-bit color photo of shark, remoras, and diver
splash	24-bit color photo of milk drop in mid-splash
tree	24-bit color photo of Joshua tree

Table 2. DMC vs LZ and PPMc.

	anemone	carlsbad	coral	face1
Original Size	1,020,201	2,884,876	1,440,014	196,623
Original DMC(2,2)	961,493	514,756	343,666	134,972
DMC(12,12)	822,464	469,313	337,323	137,834
DMC(15,15)	824,513	480,607	340,547	139,029
DMC(3,3)	832,244	458,325	330,823	**134,311**
DMC(6,6)	821,108	**453,545**	**327,406**	134,856
DMC(9,9)	**820,614**	457,545	331,336	136,431
HA 0.999	852,438	491,133	310,753	137,825
ZIP -9	937,219	854,493	475,472	154,259

	fall	figures	haiti	house2
Original Size	1,166,414	196,623	1,270,820	196,623
Original DMC(2,2)	596,992	121,579	193,872	134,099
DMC(12,12)	571,143	120,256	149,865	133,715
DMC(15,15)	**570,336**	121,002	152,541	134,326
DMC(3,3)	591,818	119,744	**139,878**	132,474
DMC(6,6)	576,690	**118,776**	143,293	**132,297**
DMC(9,9)	572,385	119,341	146,735	133,003
HA 0.999	586,765	122,955	152,061	139,147
ZIP -9	649,513	137,513	195,093	154,420

	madagas	neptune1	neptune2	shark
Original Size	3,245,420	915,855	624,975	1,440,015
Original DMC(2,2)	1,249,199	226,722	284,429	147,750
DMC(12,12)	832,163	220,998	279,010	141,380
DMC(15,15)	842,585	222,577	281,329	142,340
DMC(3,3)	905,479	219,691	277,467	143,801
DMC(6,6)	876,013	**218,907**	**275,070**	141,046
DMC(9,9)	**821,726**	219,416	276,626	**140,856**
HA 0.999	980,012	220,460	289,408	143,619
ZIP -9	1,305,743	294,640	386,422	213,270

	splash	tree	Total	Compression	
Original Size	786,447	196,623	15,581,529		
Original DMC(2,2)	498,712	153,550	***5,561,791***	64.31%	2.86
DMC(12,12)	514,282	151,148	4,880,894	68.68%	2.51
DMC(15,15)	521,647	151,735	4,925,114	68.39%	2.53
DMC(3,3)	**490,964**	150,024	4,927,043	68.38%	2.53
DMC(6,6)	497,309	**149,705**	4,859,995	68.81%	2.50
DMC(9,9)	506,042	150,496	**4,823,028**	**69.05%**	**2.48**
HA 0.999	539,492	152,105	***5,118,173***	67.15%	2.63
ZIP -9	618,138	167,732	***6,449,654***	58.61%	3.31

Original DMC with the 8 bit braid initial model
LZ (maximum compression, ZIP 1.9 -9)
PPM (Prediction by Partial Match, method C, HA 0.999 -a2)

As can be seen in the Total column in Table 2, PPM had the best compression compared to LZ and the Original DMC (highlighted by italicized bold) This was reported in the previous literature.[5]

After some experimentation, noting DMC's sensitivity to initial conditions, the counting mechanism was changed, thinking that a slower growing function might improve compression performance of DMC. This did increase compression performance! After some analysis, it was discovered that this was no different than increasing the cloning parameters of the original DMC. Nonetheless, an increase in the cloning parameters caused compression to increase substantially. Original DMC to DMC(9,9) is a 13% improvement. PPM to DMC(9,9) is a 5.8% improvement. These are large numbers in the world of lossless compression. This is at variance to what had been reported in the original paper. [1]

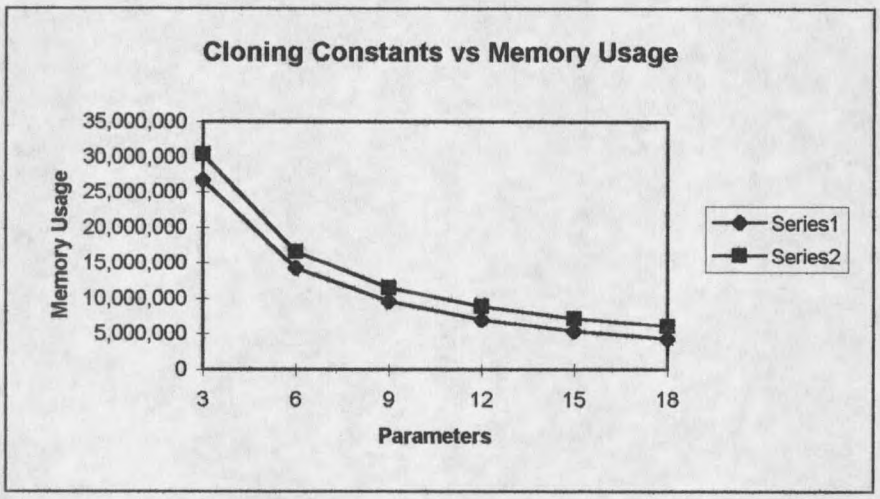
DMC run with different cloning parameters is also included in Table 2. In every case, with the appropriate choice of cloning parameters, DMC has the best compression. DMC (9,9) has the best total compression. (These are highlighted in bold text)

Just as noteworthy, DMC(9,9), uses less memory than DMC(2,2) as noted in Table 3 and Figure 1:

Table 3. Cloning constants vs memory usage

Cloning Constant	Memory Usage	
	anemone Series 1	shark Series2
3	26,753,448	30,351,096
6	14,308,536	16,597,344
9	9,565,872	11,513,544
12	6,983,016	8,857,872
15	5,381,544	7,183,920
18	4,320,096	6,097,272

Figure 1. Cloning constants vs memory usage



Shark (Series2) was selected as representative of the highly compressible images. Anemone (Series1) was selected as representative of the relatively incompressible images. A similar profile for memory usage was observed for all images. The constant shift of the two curves for Shark and Anemone is related to the difference of the original image sizes.

The space complexity of DMC is $\Theta(n)$. There can be at most one node for every bit in the input image. Unfortunately, the constants are large.

Let $K = \frac{C_1}{C_2}$ then

Implemented on a 32 bit operating system, node size is 16 bytes. There are two 4 byte pointers, and two 4 byte counters. Most images sizes are in bytes, so node size times 8 bits per byte results in

$\mathbf{C_1} = 128.$

However, by inspection of DMC,

$\mathbf{C_2} = (F_1 * \text{Parameter1} * F_2 * \text{Parameter2})$ where $F_1, F_2, < 1$

F_1 and F_2 are the average probability that the respective parameter is not sufficient to result in a cloning operation. Both Parameter1 and Parameter2 are counts which must be exceeded before cloning can occur. Clearly, higher values of Parameter1 and Parameter2 will reduce space requirements.

SEARCHING PARAMETER SPACE

The discovery of higher parameter values having positive impact on both memory and compression performance, suggested revisiting the standard benchmark for lossless compression, the Calgary corpus.

Table 4. DMC* with the Calgary Corpus.

File	Size	DMC*	P1,P2	DMC*	[7] PPMC	[7] PPM*	[7] BW94	[4] GDMC	[4] DMC
BIB	111,261	30,537	2,2	2.20	2.11	1.91	2.07	2.05	2.28
BOOK1	768,771	232,924	2,11	2.42	2.48	2.40	2.49	2.32	2.51
BOOK2	610,856	167,176	2,3	2.19	2.26	2.02	2.13	2.02	2.25
GEO	102,400	60,173	5,11	4.70	4.78	4.83	4.45	5.16	4.77
NEWS	377,109	130,720	2,2	2.77	2.65	2.42	2.59	2.60	2.89
OBJ1	21,504	11,082	2,2	4.12	3.76	4.00	3.98	4.40	4.56
OBJ2	246,814	85,289	2,2	2.76	2.69	2.43	2.64	2.82	3.06
PAPER1	53,161	18,149	2,2	2.73	2.48	2.37	2.55	2.58	2.90
PAPER2	82,199	26,580	2,3	2.59	2.45	2.36	2.51	2.45	2.68
PIC	513,216	49,704	13,28	0.77	1.09	0.85	0.83	0.80	0.94
PROGC	39,611	13,648	2,2	2.76	2.49	2.40	2.58	2.67	2.98
PROGL	71,646	17,799	2,2	1.99	1.90	1.67	1.80	1.83	2.17
PROGP	49,379	12,318	2,2	2.00	1.84	1.62	1.79	1.90	2.22
TRANS	93,695	22,462	2,2	1.92	1.77	1.45	1.57	1.78	2.11
	3,141,622	878,561			892,441	821,405	856,217	848,956	928,379
	Weighted average			2.24	2.27	2.09	2.18	2.16	2.36
	Unweighted average			2.57	2.48	2.34	2.43	2.53	2.74

The optimum parameters, listed in column P1 P2, were obtained by exhaustive search. All parameters from 2,2 to 32,32 were tested. This shows that, with the optimum parameters selected for DMC there is a good improvement over the results reported by Teuhola and Raita [4]. It exceeded PPMc as reported by Cleary et al. [7] in 1995. Optimum parameter DMC, noted

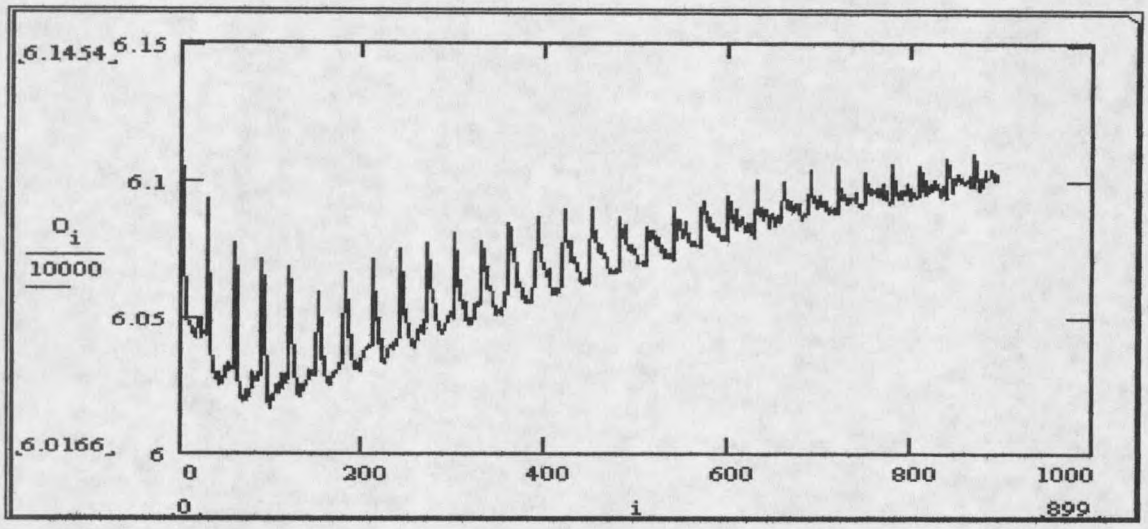
as DMC* is competitive (within 6.5%) with the best compression reported to date ,unbounded context length PPM* [7]. For the graphics image (pic), DMC* provides the best performance of any of the general compression methods. This also confirms previous noted results.

Exhaustive search, scanning the entire file 900 times, to determine the best parameters for DMC is computationally intensive. Several methods were attempted to improve upon this.

One method is to sample a portion of the file, and then test different parameters on the sample. This did find better parameter settings than guessing or assuming a "best" average value, but did not often find the optimum parameter settings. Either, selecting different portions of the file or varying sample sizes, resulted in different parameter selections. It is difficult to determine what portion of the image will best represent the overall image.

A hill climbing search, increasing parameters until there was no further compression improvement was not successful. Parameter space had many areas that were local minimums. Figure 2 plots, in a 2 dimensional fashion, the total compression versus each parameter pairing for the file Geo. Each parameter pair is one point on the X axis. Parameter 2 varies most rapidly. Geo's optimum was at (5,11) or point 171 (5*32+11). It also shows the many local minima.

Figure 2. Geo, total compression vs parameter pairs



Setting the parameter values, even the optimal values, to some constant, is establishing an average that is used for the entire file. The question that follows is, "How much variation in a local region is there around the average?" Again, enumerating all parameters over small regions of an image is illuminating. The pic file was divided into regions of 4096 bytes. Figures 3 and 4 show the variation of compression performance in two representative regions. Figure 3 shows a region where 4096 bytes was being compressed to about 100 bytes. Figure 4 shows a region where 4096 bytes was being compressed to about 900 bytes.

Figure 3. Pic, region 1, compression vs parameter pairs.

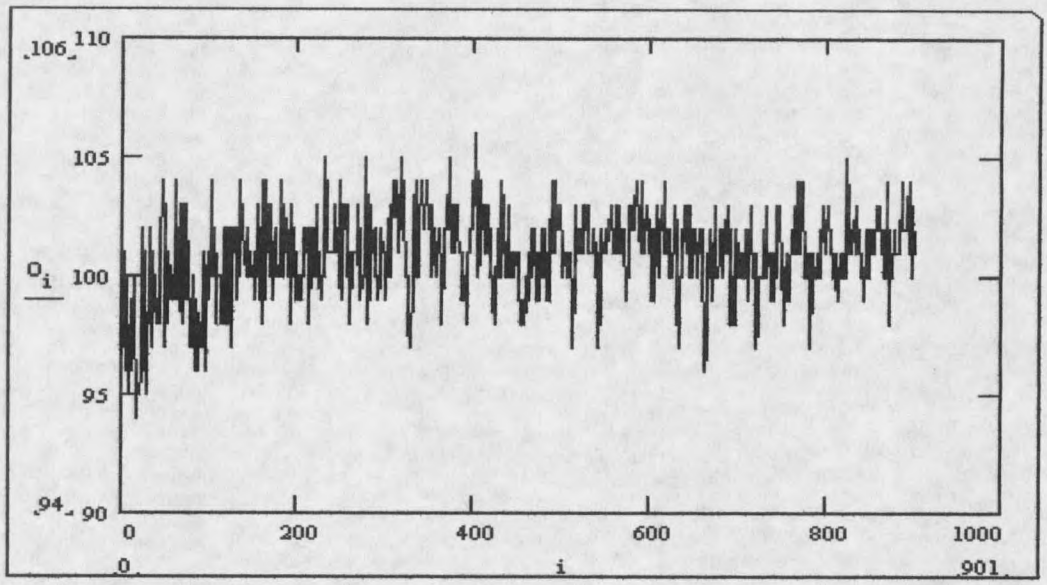
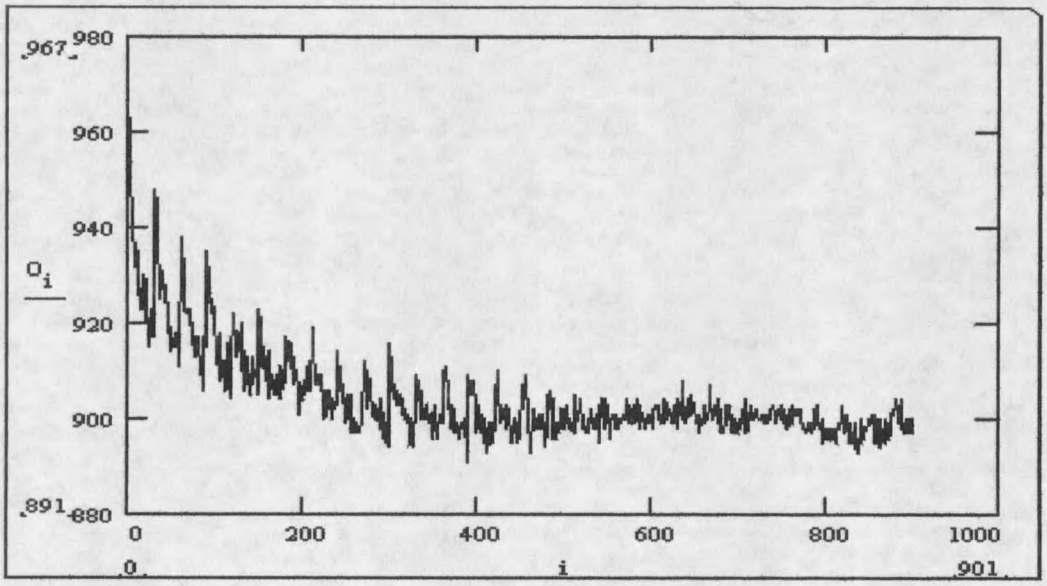


Figure 4. Pic, region 2, compression vs parameter pairs.



As can be seen at parameter pairing 17 (2,17) of Figure 3 , where you are achieving the best compression for one region, you are achieving sub-optimal ,compression at another region, paring 17 (2,17) of Figure 4, compression = 927. At parameter pairing 392 (14,31) optimal compression for the region is 892 bytes. This would be about a 4% improvement.

Also note in Figure 4, at pairing 862 (30,21), compression is 894. This is nearly optimum, and the higher parameter values would result in less memory being required.

DMC SIMULATING RUN LENGTH ENCODING

Run length coding (RLE) has been popular as a simple and modestly effective method for compressing graphics images. It is a part of the TIFF 6.0 specification, known as PackBits; defined as a compression method for the BMP image format; as well as many others. As an aside, but to reassure ourselves that DMC is an appropriate compression method, consider this test case. Assume a graphics image, 640x960 completely white. This is a highly compressible image, particularly suited for RLE compression. Table 4 shows the results of the three major types of compression.

Table 5. White image compression.

Original Size	78,811
DMC1	37
ZIP	106
HA (PPMc)	22

This small example shows the efficacy of DMC and PPMc, both context modelers, with arithmetic coding. The markov models perform 3 to 5 times better than LZ.

If we modify DMC1 so that the first node is preloaded with the apriori probability (76,811/76,811 + 0.2), DMC reduces the white image to only 3 bytes!. This example shows the advantage of correctly predicting the next bit with a high probability. Since we preloaded the counts with our apriori

knowledge, the probability starts at very high values (very close to 1.00), as opposed to converging more slowly in the non preloaded count case. The high probability causes the interval in the arithmetic coding step to be reduced by a very small amount. So small that the arithmetic code never exceeds the 24 bit window. Both, a 38,400 byte (38,400 = 640*480 /8) white 640x480 image and a 76,800 byte white 640x960 image, are reduced to 3 bytes (only the final cleanup bytes)! The final 3 bytes are different in each case and control the decompression process so that the correct number of bytes is created when it is uncompressed. The combination of an arithmetic coder, which can code into fractional bits, and a high probability modeler simulates run length encoding. To represent a number of bytes larger than 65536 a 3 byte count value would be needed.

ANALYSIS

Why does DMC perform better on graphics images? One reason is the nature of graphics images. These images, although stored byte by byte, use a sampling process to generate the RGB values. If there are sampling errors, as there inevitably are with a analog to digital conversion, they tend to be in the least significant portion of the RGB bytes. DMC operates a bit at a time and can use the reliable most significant bits without having the noisy least significant bits contaminate the entire byte. For illustration, consider this special case. Assume that each of the separate RGB values is of the form {1111xxxx}, where xxxx varies from {0000-1111}. Randomly create one pixel for every possible combination, 4096 (16x16x16) pixels or 12,288 bytes. Byte oriented compression methods will find no compression, assuming it is a random distribution. Every new byte will create an escape. Bit oriented DMC will highly compress the first 4 bits of every byte. PPM(a) through PPM(d), PPM*, and the byte oriented GDMC all suffer the same problem. The input could be transformed into a character form of the bit values. While this would help with this special case, with actual graphics images this will not help PPM(a) through PPM(d), as these methods normally restrict contexts to lengths of 6 or less. These methods do compress graphics images. Therefore, they are discovering combinations of RGB pixels that generate reliable predictions. They would need

to use context lengths of up to 48 (bits) to achieve the same effect. GDMC and PPM* would probably perform as well as DMC, although their memory and time requirements would probably be large.

CONCLUSIONS

The original DMC [1] compresses graphics images better than the best other lossless methods in the literature GDMC[7], PPM*[4] and BW94[4]. Two identified reasons for the better performance are: the bit oriented nature of DMC, and the ability of DMC to represent higher order conditioning contexts.

Detailed analysis of the effect of different parameter settings demonstrate that higher parameter settings, cloning states more infrequently, result in better compression performance of graphic images and substantially reduce memory requirements for DMC.

Further analysis shows that different regions of an image compress to differing degrees with different parameter settings. This offers a possibility of achieving even better compression performance, although it appears that the improvement would only be in the 5 percent range, with the current implementation of DMC.

There are obvious weaknesses to the current approach. Enumeration of all possibilities to identify the best parameters is hardly efficient. Memory usage is still high, although certainly within reach of current hardware configurations. On the positive side, the search for the best parameters only applies to

compression. Decompression simply uses the parameter settings provided by the compression side and runs in $\Theta(n)$ time, even if n is bits.

Perhaps even more interesting are the research possibilities that are now illuminated. If different regions compress differently, how do you define regions? What are the characteristics of a region? Can tradition image segmentation techniques identify regions that compress with a certain parameter settings? If regions can be characterized by their compression performance, can this profile be saved to a temporary store to be used later if that particular characterization reoccurs? This could reduce main memory requirements for DMC even further. Further work could be done to:

Find better methods for searching parameter space.

Reduce node size by making pointers smaller with local references.

Reduce node size with smaller counts by scaling counts periodically.

BIBLIOGRAPHY

[1] Data Compression Using Dynamic Markov Modelling; G.V. Cormack and R.N.S. Horspool; The computer Journal, Vol. 30, No. 6, 1987, pp. 541.

[2] A note on the DMC data compression scheme; T. Bell and A. Moffat; The Computer Journal, Vol. 32, No 1, 1989, pp. 16.

[3] The Relationship between Greedy Parsing and Symbolwise Text Compression; Timothy C. Bell and Ian H. Witten; Journal of the Association for Computing Machinery, Vol. 41, No. 4, July 1994, pp. 708-724.

[4] Application of a Finite-State Model to Text Compression; Jukka Teuhola and Timo Raita; The Computer Journal, Vol. 36, No. 7, 1993, pp 607.

[5] The Structure of DMC; Suzanne Bunton; Proceedings of Data Compression Conference, March 1995.

[6] Text compression; Timothy C. Bell, John G. Cleary and Ian H Witten; Prentice Hall 1990, ISBN 0-13-911991-4.

[7] Unbounded Context Length for PPM, John G. Cleary, W.J. Teahan, Ian H. Witten, Proceedings of Data Compression Conference, March 1995.