Towards a Generic Data Compression Advisor

Jennie Rogers

Brown University
Providence, RI, USA
jennie@cs.brown.edu

***Abstract*— Compression allows scalable storage of large amounts of data and alleviates the I/O bottleneck for data-intensive applications. Over the years, a large number of compression algorithms have been developed to support various data types. Our goal is to develop a generic tool that helps quickly and accurately decide what compression scheme(s) best match the constraints and expectations of users with respect to factors such as compression time (encoding/decoding), space (data/meta-data), and quality (lossiness).**

As an initial step, we investigate techniques for generating and expanding time-space-quality Pareto frontiers for multidimensional array data. To this end, we studied several enhancements to traditional algorithms; these include 1) partitioning the arrays into multidimensional tiles and applying heterogeneous compression schemes across the tiles; (2) introducing lossiness to trade accuracy for space; (3) generalizing uni-dimensional compression schemes to recognize and leverage multidimensional patterns; and (4) adaptively adjusting compression parameters on the basis of data characteristics and memory limitations. Our measurements based on two real-world data sets reveal that these techniques are effective in generating new Pareto optimal points.

I. Introduction

Data compression is critical to making large datasets scalable. By minimizing size datasets like the Sloan Digital Sky Survey [10] can scalably collect high resolution imagery. Scientific and surveillance applications especially need this type of support, although many other domains benefit from it as well.

Compression may be used to expedite processing by allowing data to be read from disk faster. This is caused by disk access being the bottleneck for many processing applications. Allowing this phase to go by faster may make processes execute faster on data despite the processor overhead of decompression.

In this study we will target scientific data, especially in the form of arrays to find the best compression scheme for a given workload. In this context "best" may be defined in a few ways including times for encoding, decoding, space saved by compression or a trade-off among these criteria. We aim to provide a tool to help users make a decision to best fit their needs in an efficient manner.

In this work we will first survey common compression techniques from a variety of domains. We will look at well-known compression schemes such as Huffman Encoding and Lempel-Ziv. In addition we will examine simple shortcuts such as Suppression and Run-Length Encoding.

Next we will discuss the trade-offs associated with compression in terms of time taken in encoding and decoding as well as space taken by compressed data. There are situations under which each characteristic is beneficial and the user will need to indicate where their priorities lie. We will plot these characteristics for specific datasets on a three or four-dimensional plane and analyze our options by generating a Pareto frontier. We will then select from this set of Pareto-optimal points based on user needs and algorithm performance.

Finally we will look at additional enhancements possible for our compression schemes. We will explore extending these algorithms to exploit multi-dimensionality. After that we will discuss how to generalize these algorithms to support controlled levels of lossiness as a means to save space and time. We will also examine extending these algorithms to work in the temporal domain, capitalizing on similarity between successive measurements of a dataset. We will talk about future work involving temporal compression and hybrid compression approaches.

II. Related Work

Until now research on compression has mostly focused on a specific problem domain or been a very simple shortcut. For example, JPEG [8] is

finely tuned to compress photographs well. Lempel-Ziv [12] encoding is geared towards text, encoding losslessly while looking for repeating patterns such as those typical to words. A simple shortcut is something such as Suppression, where instances of the most common value is omitted from the output of a dataset with appropriate annotation regarding where it occurred. We will use these algorithms as the base choices for our decision-making framework.

The trade-off of fidelity for space in lossy compression was first studied by Shannon in 1948 [9]. He created the Rate-Distortion Theory in which he determined that the minimum information (R) needed to transmit information over a channel without exceeding a distortion rate D can be modeled. R is generally modeled as bits per sample and D usually refers to the variance between the original and compressed data. This is useful for if we know how much lossiness we can tolerate and want to determine the output size as a consequence. In contrast, we are more focused on modeling our lossiness in terms of the range because this is a more intuitive approach for users and potentially makes uncertainty analysis easier.

Space-efficient bit map indexing by binning (i.e. encompassing ranges) has been studied in [6]. Here they primarily framed it as an indexing technique to speed up query evaluation for data warehousing. We are using similar methods to exploit the space-fidelity trade-off as a compression technique, but will also benefit from these attributes. The notion of binning is used to group similar values for lossy compression.

Combining various levels of compression was first studied in [1]. This system exclusively looked at image data and attempted to isolate areas of high contrast for lossless compression and encoded homogeneous regions lossily. It also exclusively codes in 8x8 pixel blocks. Our system both does not use domain-specific information and incorporates more than two compression options per encoding.

History-less multidimensional lossy data compression has been studied in [3]. In this work they looked at how to build a dictionary of basis functions to compress the data from this. In contrast to this work, they focused primarily on frequency domain algorithms. This does transform-based coding that is more sensitive to the data being passed in.

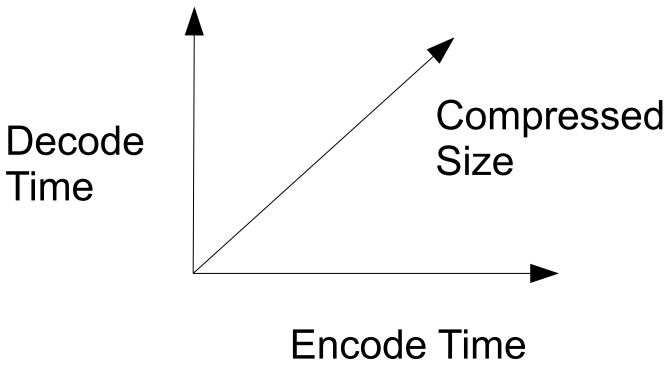

Fig. 1. Axes for a typical Pareto frontier.

For our compression algorithm select recommendations we generate Pareto frontiers [13]. Pareto frontiers were first studied in economics as a way to determine what decisions can make some players better off without making anyone else worse off. We extend this in a three dimensional plot on a training set using encoding and decoding time paired with compressed size.

III. Pareto-Optimal Decision Making

Pareto efficiency is a method in economics used to allocate income and other resources. A Pareto efficient point is one in which a party involved is at the best position it can be without adversely affecting other points with the same value. At its simplest it creates a "Pareto frontier" or a list of points that are most beneficial with regards to each dimension. In our case we will be looking for strong Pareto optimal points, where someone must benefit for a point to be chosen over another potential solution.

We developed a framework to narrow down our candidate list of compression schemes and pick the right one. To do this we will generate a three-dimensional Pareto frontier based on encode time, decode time and compressed size for a representative training set of the dataset to be encoded.

Our graph will be based on a set of axes such as the ones in figure 1.

IV. Compression Schemes

We use several off-the-shelf compression schemes for our initial analysis. Below we will outline their mechanisms, strengths and best uses.

Lempel-Ziv [12] looks for patterns in data and creates pointers to previous occurrences of a sequence of bytes. This is especially useful for natural language text and anything that will have variable length sequences of repetition. We are using a variant on the LZOP implementation [7] for our experiments.

Huffman Encoding [4] finds the minimum length of a code (in bits) to properly compress an alphabet one byte at time. It does this by analyzing the dataset and building a dictionary where the symbols that occur most frequently are encoded with a smaller number of bits and less frequent dictionary entries are encoded with longer codes. The codes are built with a binary tree with the most frequent dictionary entries on top and less frequent entries occurring further down. This encoding scheme is beneficial for data sets with a skewed distribution of values, where several values occur more frequently than the rest of the alphabet.

Run-Length Encoding [11] compresses data by converting it to a set of runs, typically of the form $(value, start\ offset, run\ length)$ to denote a run comprised a single value starting at the offset specified and continuing for the length given. This is primarily geared toward sets of data where like values are found together and can be quite beneficial under those circumstances as demonstrated by [2].

Suppression omits the most commonly occurring value in a data set. It accomplishes this by converting the dataset into a set of runs of the form $(start\ offset, run\ length, non-common\ data)$. This works best for skewed dataset where one value is highly dominant to the rest.

V. Base Experimental Results

We set up a series of experiments to demonstrate how the Pareto frontiers narrow down options for a couple of image-based data sets.

A. Sloan Digital Sky Survey

In this experiment we took 2-d arrays of images from the Sloan Digital Sky Survey [10]. We used imagery from data release 6, stripe 94, camera column 4. We selected only the infrared channel to have a simple two-dimensional example. Each frame is 2048x1489, with one byte index values. We stream 201 frames for this test.

In this experiment all but one of our algorithms was on the Pareto frontier. Huffman encoding lost out because its output was just too big.

Suppression did well in terms of its compression ratio because the background is dominant in most star scenes. It is sufficiently simple and consequently fast such that it is a competitive option. Lempel-Ziv benefitted in similar ways, only looking for more complex patterns of stars rather than simply background.

Run Length Encoding made its mark by being faster than any of the others. Its compression ratio was not excellent, but it could not be dominated due to its speediness among this set.

B. Charlotte

In this test we are compressing a short video of complex data. We are looking at a series of 320x240x3 images with 8-bit color depth. This is a video of people walking around to simulate a complex workload in which there is temporal correlation, but limited spatial correlation. It is also a good case for how this system will perform with traditional surveillance data. As with the previous set, we use 201 frames.

The Pareto frontier consisted of one point for this experiment: Lempel-Ziv. It is both faster and produces smaller output making the compression algorithm selection process trivial. The next closest algorithm was Suppression, but it still lagged behind in every dimension.

VI. Enhancements

We can better tailor compression decisions to scientific and other array-based data by extending the algorithms we have to support more features in the form of lossiness, hybrid encoding, temporal support and multidimensionality.

A. Multidimensionality

Arrays are useful for scientific and visualization data and frequently generate large datasets. Thus they make a good test case for compression needs. Multidimensional data presents unique challenges and opportunities for compression. By exploiting spatial relationships along multiple dimensions we can theoretically encode complex patterns with less bits. Examining arrays as multidimensional entities

also requires more analysis to correctly select what feature size best fits a dataset to isolate repetition, as well as more complex for data partitioning options. Most of these problems will be explored further in future work. Below I will discuss algorithms that have been extended into multidimensional forms.

Run-length encoding has been generalized to multiple dimensions by allowing runs to traverse all dimensions in an array. In this case our runs now look like $(value, start\ point, run\ length\ in\ each\ dimension)$. This extension is only useful for datasets where similarity extends along multiple dimensions. If data does not satisfy this requirement then performance may suffer dramatically due to the size of overhead (an extra 4 bytes per dimension per run).

Huffman Multidimensional extends traditional Huffman encoding to make dictionary entries $tiles$ instead of individual bytes. Tiles are non-overlapping multidimensional subarrays that may represent features in a dataset. They form a grid over the entire array to be encoded which is analyzed to generate Huffman codes. Tiles must be selected judiciously to be small enough to have a high probability of repetition, but not so small that there is significant overhead in encoding them due to more outputs.

JPEG [8] is a common compression scheme for images. It requires that all of its inputs be either two or three dimensional. It works in several phases. First it is converted into luma-chroma space (YCbCr) from RGB space. Then it cuts the chroma (color component) down by half of its accuracy after it is coverted (to capitalize on human eyes being less sensitive to color variations than light ones). After it is brought into the frequency domain via a discrete cosine transform. Next the data is quantized with a specialized matrix where different numbers of bits are truncated depending on the indexes. Lastly the data is compressed with entropy encoding using a zig-zag pattern.

B. Lossiness

Lossiness may allow us to profitably trade accuracy for time and space. If one considers the data we are processing already a discretization of reality this is simply recording it at a lower fidelity (or irreversible compression). We define lossiness by allowing the user to specify a range of values the data may occupy. At the simplest level all of our previous algorithms may be extended to support lossiness by "binning" the data before encoding it into its appropriate ranges. For example, if we have a range of five and a sequence of values (0, 7, 13, 125) it would be binned as (0, 5, 15, 125) by rounding each value into its closest neighbor in our new continuum. This makes it so like values are guaranteed to be compared correctly without accounting for all potential cases.

In addition to making our algorithms produce smaller output lossiness may make our algorithms run faster. For example in Run-Length Encoding it may make the runs longer thus making it so we do not have to query the data many times to build our runs. Naturally it also makes decoding faster.

Also lossiness may make some of our algorithms more scalable. In the case of Huffman Multidimensional its biggest bottleneck is sorting the frequency counts on every dictionary entry. In the lossless case this dictionary may grow exponentially. For example, while a 1 byte dictionary entry only has 256 potential values, a 2x2 tile may have up to 2^32 or 4,294,967,296 potential values. By imposing binning we reduce the number of combinations a tile could produce. If we have a range of 10 each byte can only be one of 26 values. Now our 2x2 tile can only have 26^4 or 456,976 possible combinations. So, as you can see, just a little lossiness may bring down our codebook by almost four orders of magnitude. This makes our huffman codes shorter as well as making the algorithm run faster by lessening our search and code creation time.

Quantization lossily encodes data by chopping off the lower order bits. This provides an approximation of the original data in an efficient manner. Unfortunately it is coarse grained because it can only approximate in powers of two.

Downsampling combines multidimensionality with lossiness by breaking an array into tiles and taking the average of each tile for output. This can be made coarser or finer grained by varying tile size. It is good for data where similarity occurs across dimensions.

Both of these techniques are only beneficial for lossy compression due to their natures.

C. Temporal Encoding

Temporal data is a stream of arrays provided in a sequence of recordings. In a manner reminiscent of MPEG we can use relationships between subsequent measurements to limit redundant records.

Delta Encoding statically breaks the array into uniform n-dimensional tiles and compares them to their predecessors. When the arrays have changed enough then we pass on update tiles for the cells. The user specifies the degree of change acceptable as the maximum deviation from the cached version of a tile.

Adaptive Huffman brings Huffman Multidimensional encoding into a scalable online technique. In Adaptive Huffman we dynamically build codes using the Faller-Gallager-Knuth method [5]. This is different from Huffman Multidimensional in that the codes reflect the current known frequency of a symbol, whereas the other implementation works strictly with global knowledge. It allows for faster updates by doing incremental sorts of the tree. Adaptive Huffman consequently exploits local optima in code generation.

In the same vein as Huffman Multidimensional this algorithm's biggest weakness is scalability as the code book grows. In addition lossiness we keep the dictionary manageable in Adaptive Huffman by maintaining a cache of values rather than maintaining "true" global knowledge. This has the advantage of making encoding go faster due to a smaller code book (because codes are recalculated at every insertion). The downside is that we potentially may have to output the same tile twice uncompressed if it is evicted and reinstated.

We experimented with a couple of strategies to limit the dictionary growth (and subsequent growth in code length). First we examined the least recently used eviction pattern. In this approach we timestamp all of our dictionary entries by the last time they were accessed. When our cache is full we evict the one that was accessed longest ago, replacing it with the new entry. We record the evicted entry with the tile number it was evicted from for reconstruction.

Secondly we experimented with least frequently used eviction, where the code with the lowest number of accesses gets evicted when the cache fills. The results of these schemes on Charlotte data are in figure 2.

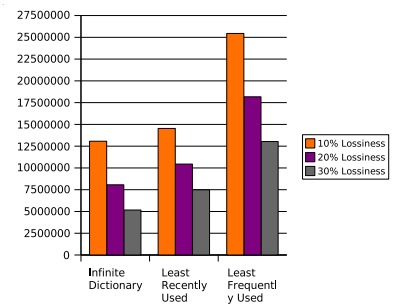

Fig. 2. Adaptive Huffman eviction strategies on Charlotte data

Least recently used performed significantly better than least frequently used. This is primarily because least frequently used caused a lot of thrashing in the cache. For LFU an entry can easily be evicted, immediately reinstated with a frequency of one and now appear least frequently used again. This is wasteful in that it would cause us to output the tile twice uncompressed for our dynamic dictionary. Thus, unless evictions are very infrequent this is not a viable option.

Least recently used performed better because it did not cause thrashing. In this case entries were only deleted if we had not seen them for longer than all of their neighbors. We called each incoming array an epoch and randomly selected a victim from the oldest epoch in our dictionary. For the rest of our experimental results we omit caching for simplicity.

Motion Vectors are an extension to delta processing. This is a composition of Delta Encoding with Adaptive Huffman. For this approach we first had delta encoding filter out which tiles to encode. We maintain a cache and only mark tiles that had changed significantly enough for encoding. Next Adaptive Huffman encodes the selected tiles for output. This allows us to both filter for change on the temporal dimension while allowing selected changes to be compared to neighbors. This manifests in searching our known vocabulary for a tile, regardless of where it had appeared last. Approaching the search for a prior match is much more efficient than a traditional sliding windowed search, but in exchange relies on the matching tile occurring along tile boundaries.

D. Hybrid Encoding

We can achieve better compression ratios at the expense of time by systematically encoding different areas of an array with different schemes. We investigated several of these hybrid compression approaches for this work with mixed results. In all of our schemes we break our array up into a grid of non-overlapping areas and encode each one individually.

1) Exhaustive Approach: For the highest compression ratio possible in this framework we can iteratively encode each area with each scheme and find the algorithm which takes the minimum amount of space for compressing an area. This is basically the extreme case of hybrid compression where we pledge the most time in exchange for an approximately guaranteed best performance.

2) Downsampled Decision: This is a slightly smarter version of the exhaustive approach. Here we take our input array and significantly downsample it. Then we iterate over every algorithm we have on the downsampled version of an area and select the minimum output size for to encode the original version of our chunk. It saves significant time by not experimenting with the whole version of the chunk (usually the sampled area is about the square root of the original size), but may not always pick the right option. Frequently it picks an option that is good enough though as we shall see later.

3) Decision Tree: We have started investigating a simple decision tree for choosing the right scheme for a set of data. We do this by hierarchically characterizing what makes an algorithm save space. For example, one of our highest branch points checks to see if the distribution of the input data is skewed. This observation eliminates about half of our algorithms and we continue down this tree until we reach a branch. Unfortunately this design is not very extensible to adding new algorithms and may require some artificial intelligence or automation to make it well tuned to individual datasets.

4) Similarity Scoring: Here we exhaustively encode every nth area in a stream of arrays. When we do this we add it a our vocabulary, a list of known tiles paired with its "correct" scheme. We then generate a signature for each vocabulary entry. The signature characterizes the distribution of values, how often features (i.e., tiles) are repeated and other salient characteristics. For each are that is not to be added to the vocabulary we generate a signature and compare it to the values in the vocabulary. We then encode it with the correct scheme for its closest match. In the background we also make sure that our vocabulary only maintains entries that are sufficiently different from each other. We also evict old entries in our vocabulary to prevent the search for a match from becoming more expensive than the exhaustive approach.

E. Restricted Hybrid

This is a blend of the previous approaches. First we prune our search to only the algorithms found on the Pareto frontier. Next we apply the exhaustive approach, but only iterating over the ones that were already on the Pareto frontier. This makes it so that our search will probably not take too long and every algorithm considered performs well for at least one dimension of our criteria on the dataset.

In our experimental section we compare the performance of Downsampled Decision, Restricted Hybrid and Exhaustive Approach. The other two have not been finely tuned enough yet to produce meaningful results.

VII. Sampling

In order for the Pareto frontier to make efficient recommendations we need to judiciously sample the dataset. We start out given a percentage of the dataset that the analyzer may access and need to distribute the samples around the set. Presently we do this by sampling a fixed length band n times where n is equal to the sample frames divided by the band size. In future work we may experiment with varying both the band size and number of bands.

VIII. Experimental Results with Enhancements

We ran experiments with the same datasets discussed in the base experimental results section. In this set of experiments we examined the compression algorithm performance along four dimensions. Our first dimension was space consumed by compressed data. The second metric we used was lossiness, as expressed by a range that individual indices may occupy. We varied this parameter between 0 and 15, in increments of 5. Finally we examined

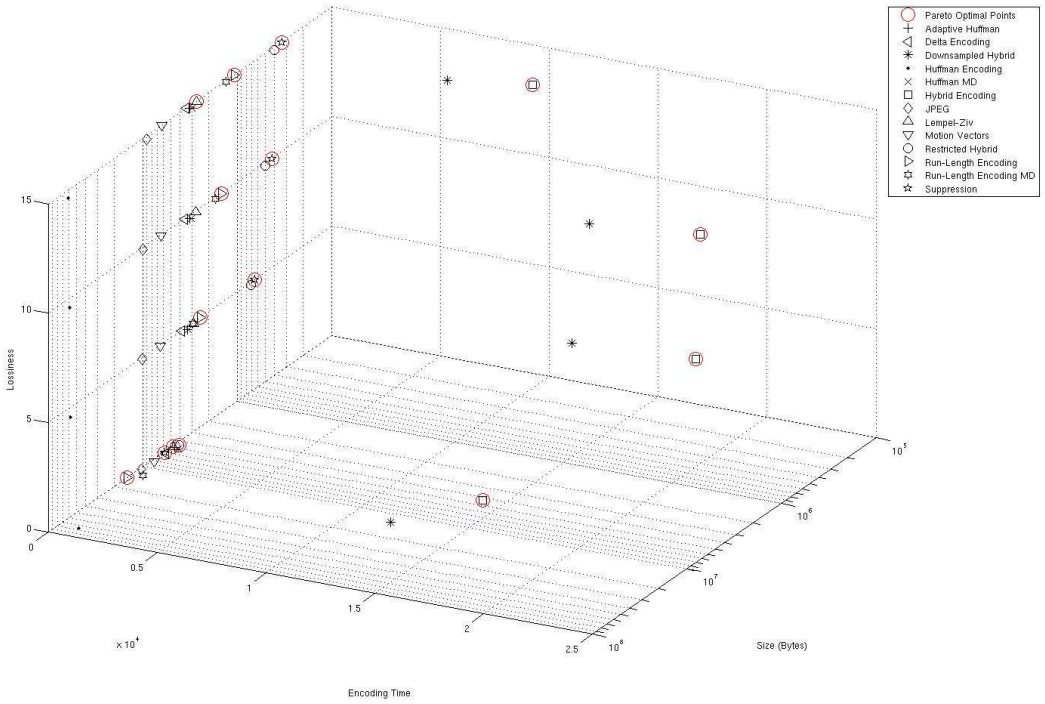

Fig. 3. Pareto frontier for base and enhanced compression algorithms on SDSS data. Comparison of lossiness vs. compression size vs. encoding time with detail of Pareto frontier.

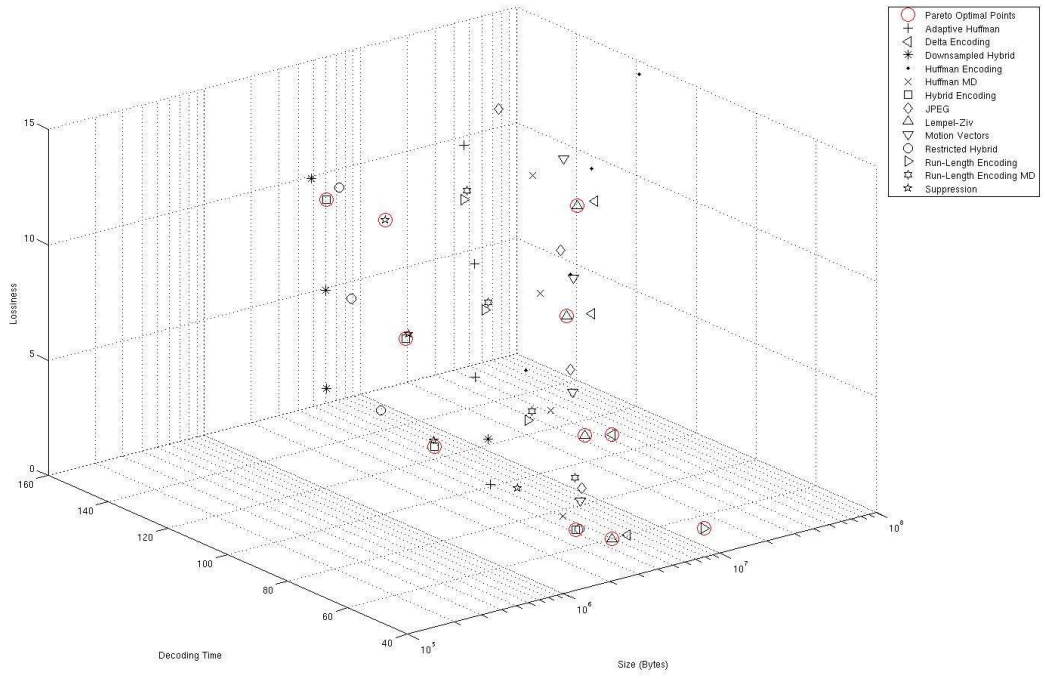

Fig. 4. Pareto frontier for base and enhanced compression algorithms on SDSS data. Comparison of lossiness vs. compression size vs. decompression time with detail of Pareto frontier.

encoding and decoding times, to assess how long each algorithm will take end-to-end on a sample set of the data. For each data set we created two 3-d scatterplots. Each contained space and lossiness axes and was paired with one time dimension, either encoding or decoding. This is primarily to simplify the visualization of our results.

We then generate a Pareto frontier for all four dimensions. We will examine the results we get with this superset of information and how it differs from the results on three (selecting only one of the time dimensions). Theoretically the more dimensions you use the better your results will be, but we are still investigating this hypothesis. If this pans out then it is possible to trade efficient decisionmaking for a reduction in quality by limiting the number of dimensions we explore. The downside to using more dimensions though is that this approach will generally produce more points on the frontier because it is harder for a candidate point to be dominated in all dimensions at the same time by another point.

A. Sloan Digital Sky Survey

This dataset was the same. We applied 5x5 tiles for Huffman Multidimensional dictionary entries and other compression algorithms that required tiles. For hybrid encoding we broke each frame into an approximate 5x5 grid. The results for various degrees of compression are displayed in figures 3 and 4.

In figure 3 we.looked at the Pareto frontier as a function of compressed size, encoding time and lossiness. This graph clearly demonstates that as lossiness increases we effectively reduce the cost of compression in terms of time and of course get better compression ratios. For the first frontier we found our most effective points to be hybrid encoding (at all degrees of lossiness), Lempel-Ziv (for lossiness ranges 0 and 15), Suppression (at all degrees of lossiness) and Run-Length Encoding 1-D at all degrees of lossiness. Restricted encoding also appeared for 0 lossiness. For the piecewise evaluation Lempel-Ziv also occurred on lossiness 10.

Suppression produced a good point for trading off between space and time. It achieved this by having output that was only nominally larger than hybrid while being up to two orders of magnitude faster than it. This is unsurprising for this dataset given that the images are dominated by a dark background. Restricted Hybrid worked for similar reasons. It was not dominated by any point for at least one dimension.

Hybrid maintained its place on the curve by being smaller than everyone else, but as mentioned at the cost of speed. This is simply a side effect of the cost of running every algorithm for every chunk of the array. It is very hard to beat hybrid sizewise unless a dataset is completely dominated by one algorithm at which point the cost of metadata starts to become a factor.

Lempel-Ziv and Run-Length Encoding 1-D both occupied the niche of being very fast with reasonably-sized output. When they were both on the frontier one would fail to dominate the other by the slimmest of margins, sometimes the difference being down to a second in encoding time difference. Run-length encoding does well in this set because of the skewed distribution of values (that also happen to be grouped together often). Lempel-Ziv is more complex, but has a highly optimized implementation.

One thing this clearly result set demonstrates for star data encoding is that lossiness does not heavily impact algorithm selection choices. This is because the data is highly polarized. That is to say that almost everything in the scene is either a bright star or a black background and limited shades of gray are used. As a consequence changing the lossiness does not really change our compressed data size all that much as well as our selections. As the range increases values rarely change what bins they fall into due to this contrast.

For the trade-off among lossiness, compression space and decoding time we plotted the results of SDSS experiments in figure 4. It is worth noting that encode and decode times for some compression algorithm are highly asymmetrical. Thus algorithms that were clear leaders in the past (such as null suppression) may lose out here because they have symmetrical decode times whereas others make huge gains in comparison to their encode time. Hybrid encoding is a good example of this. Its encoding time is extremely expensive, but the decoding process is a deterministic decompression

Compression Scheme	Lossiness	Encoding Time	Decoding Time	Compressed Size
Run-Length Encoding	0	18	54	14706213
Suppression	0	47	96	6011631
Lempel-Ziv	0	68	60	4945143
Restricted Hybrid	0	178.93	69	4549022
Hybrid Encoding	0	14046	69	4297949
Run-Length Encoding	5	13	72	2446674
Delta	5	21	55	3942358
Suppression	5	29	74	657537
Lempel-Ziv	5	87	58	3012493
Hybrid Encoding	5	20083	71	588170
Lempel-Ziv	10	13	62	2774295
Run-Length Encoding	10	15	75	1466493
Delta Encoding	10	22	60	3638448
Suppression	10	30	73	436126
Hybrid Encoding	10	19461	71	382610
Lempel-Ziv	15	12	58	2702585
Run-Length Encoding	15	13	75	1078089
Suppression	15	28	75	338904
Hybrid Encoding	15	11314	92	299398

TABLE I

PARETO POINTS ON FOUR DIMENSIONAL FRONTIER FOR SDSS DATA.

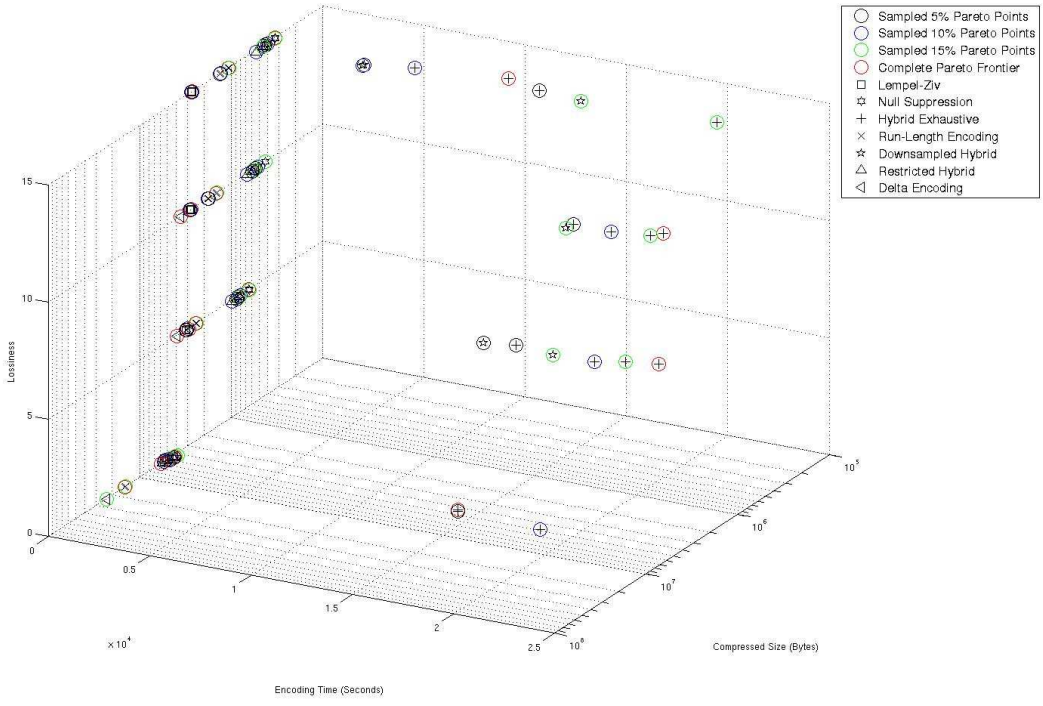

Fig. 5. Pareto frontiers for sampling of star data at 5%, 10%, 15% and all samples. Compressed size vs. encoding time vs. quality

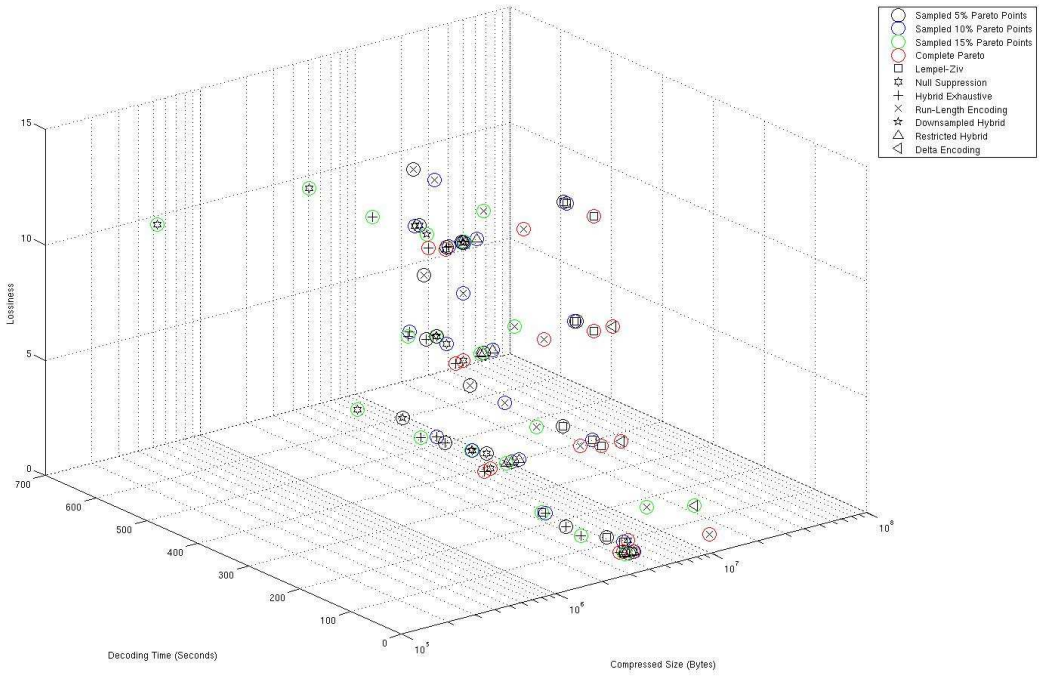

Fig. 6. Pareto frontiers for sampling of star dataset at 5%, 10%, 15% and all samples. Compressed size vs. decoding time vs. quality

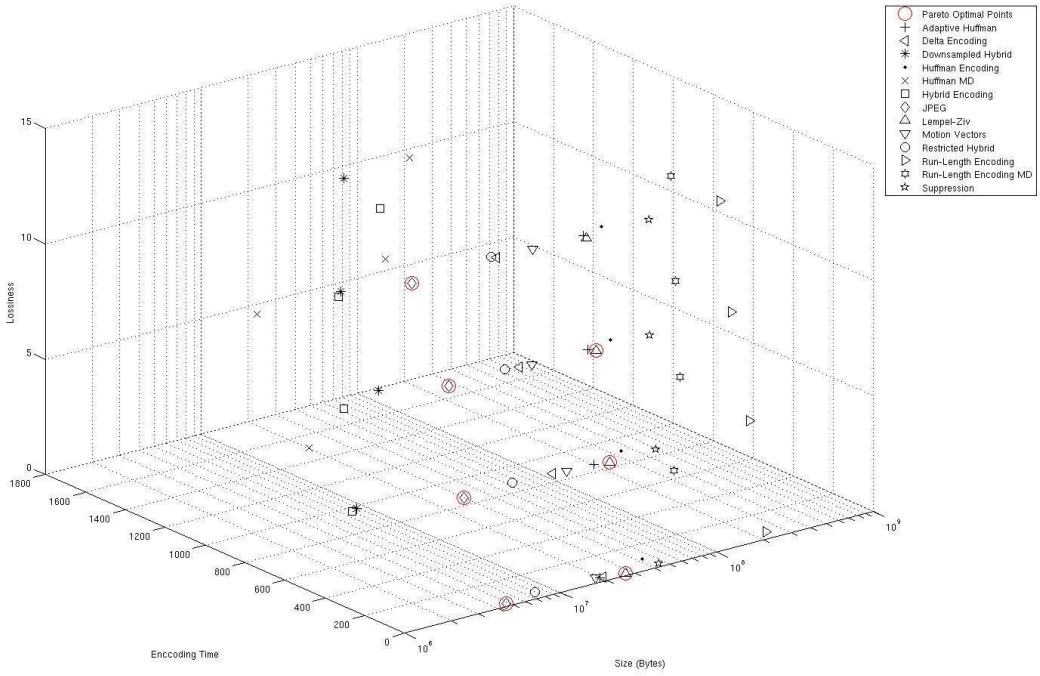

Fig. 7. Pareto frontier for base and enhanced compression algorithms on Charlotte data. Comparison of lossiness vs. compression size vs. encoding time with detail of Pareto frontier.

using whatever algorithm was elected during the compression phase so it is relatively fast. Hybrid and Lempel-Ziv were the only algorithms to appear on the Pareto frontier for every degree of lossiness. Lempel-Ziv maintained its position on the frontier throughout due to a reasonable compressed data size paired with a very fast decompression time. Again, this is primarily because we are using a highly tuned implementation of a well-known algorithm. In contrast run-length encoding made the frontier only for the lossless case. In decoding its timing and larger output size made it lose out to some of the faster algorithms.

The four dimensional frontier is basically consistent with the three-dimensional ones. The only big difference is that now delta encoding earned a spot on the curve. By adding a fourth dimension it makes it harder for a point to be dominated in every dimension so its fast times pay off with a slot on the curve. This is because the comparisons used in delta are very fast and the majority of its overhead is disk access. Disk access on this algorithm is roughly comparable in terms of cost to the other algorithms.

Preliminary results for the sampling algorithm are in figures 5 and 6. We plotted a time, either encoding or decoding against space consumed and lossiness. Sampling selections were generally close to the desired ones and the Pareto frontier of the complete one only contained values that had appeared on one or more of the sampled ones.

B. Charlotte

Charlotte used the same setup but with a 4x4x3 tile set. Hybrid encoding broke up the array into 40x40x3 areas. Again we ran trials on the same frames as in the base case. Charlotte data is characterized as having extremely limited spatial correlation, but strong temporal correlation.

In figure 7 we examine the trade offs of lossiness versus size versus encoding time. Naturally most of the three-dimensional Pareto-optimal points are closer to the zero lossiness. Lempel-Ziv occurs on the Pareto frontier for all levels of lossiness except 15. Again, this is because it is fast and has a good enough compression ratio to not be strongly dominated by any points up until the lossiness weighting pulls it from the curve (lower lossiness is always considered a more efficient solution). JPEG is very well designed for this sort of dataset because it handles the lack of correlation in the spatial domain very well with frequency domain transformations. It appears on the Pareto curve for every degree of lossiness.

Figure 8 depicts the scattergram for decoding time. Again, JPEG is preferred in most cases (all of the JPEG points are selected for the 3-d curve except lossiness 10, where its slightly slower decompress time eliminates it).

Surprisingly Huffman Multidimensional occurred for lossinesses 0, 5 and 10. This is because it decompresses very fast (due to it knowing the dictionary already and thus not having to calculate codes). Also, the Charlotte set is given to having patterns between successive arrays, so the dictionary works well at capturing those features succinctly.

Motion Vectors appeared on the three dimensional curve for lossiness level 5. Much like Huffman Multidimensional this is because it decompresses very fast. Also it has a reasonably good compression ratio because the underlying Adaptive Huffman is exploiting the same recurring patterns. This paired with the filtering step produces a solid compression ratio.

Restricted hybrid appears for the lossless case. It benefits from a fast (deterministic) decode time and a relatively good compression ratio.

The 3-D curve also contained Downsampled Hybrid for lossiness 5 and 10. These points had very good compression ratios paired with a relatively fast decompression times. The five point also fairly low on the lossiness scale which helped its ranking.

Table II shows a solid superset of the three dimensional graphs. Motion Vectors gets buoyed up by its fast compress and decompress times. This is a benefit of the online technique of Adaptive Huffman and strong temporal redundancy. Lempel-Ziv also performs well due to the pattern-oriented nature of this data. Restricted Hybrid also showed well here because it is faster and generally selects a method well enough so that the choice does not too negatively impact the output despite it perhaps not being the optimal choice.

What is a more interesting result in this set is the Hybrid Encoding never makes it on a Charlotte Pareto frontier. This is because it is prohibitively

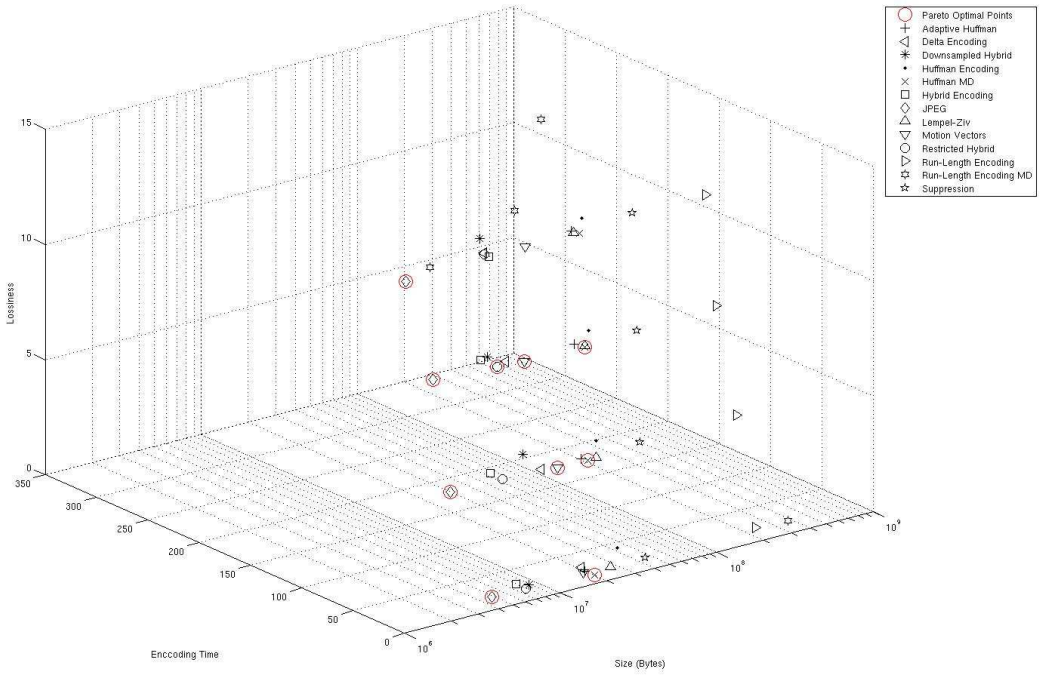

Fig. 8. Pareto frontier for base and enhanced compression algorithms on Charlotte data. Comparison of lossiness vs. compression size vs. decompression time with detail of Pareto frontier.

Compression Scheme	Lossiness	Encoding Time	Decoding Time	Compressed Size
Lempel-Ziv	0	16	18	27446418
JPEG	0	24	19	4817158
Huffman MD	0	1500	13	20177202
Lempel-Ziv	5	14	16	21388621
Motion Vectors	5	33	15	12016162
JPEG	5	35	20	2661262
Huffman MD	5	1731	14	18467505
Lempel-Ziv	10	15	14	17633017
Motion Vectors	10	32	14	7226869
JPEG	10	35	22	2133109
Restricted Hybrid	10	51.85	17	5084225
Huffman MD	10	1058	12	17022065
JPEG	15	10	8	1150613

TABLE II

PARETO POINTS ON FOUR DIMENSIONAL FRONTIER FOR CHARLOTTE DATA.

expensive to try every algorithm when one scheme is optimal the vast majority of the time (JPEG in this case). As mentioned earlier the cost of maintaining each chunk header outweighs the benefits of selecting the optimal chunk.

As always JPEG occurs for every point it can on the set. It may not always be the fastest, but it produces a great compression ratio. While considering that with encoding and decoding speeds that are "good enough" it makes this choice a solid trade-off for Charlotte's data.

IX. CONCLUSIONS AND FUTURE RESEARCH

Compression algorithm performance varies dramatically based on input data and parameter selection. In this work we generate a Pareto frontier using a representative training set to aid users in

determining the best compression algorithm for their needs.

In future work we will examine how to use the recommendations provided by this curve to aid in algorithm selection. We will vary tile selection sizes, hybrid area sizes. We may adapt to support better hybrid partitioning schemes as well.

We will also look at other recommendations such as composite objective functions. We are considering weighting approaches (where a user assigns weights for each dimension and the advisor works along the decision space accordingly). Another potential avenue is prioritization of user needs or constraint application.

We will also further enhance our compression signature generation and improve the decision tree. Semantic compression (based on context) could also potentially improve our results. Future work may include trying this data on other datasets (such as text or filesystems) and accounting for time in our hybrid decision-making.

Regardless of the next step, building a Pareto frontier is a cost-effective way of pruning our decision space for compression algorithm selection.

REFERENCES

[1] M. E. Banton. System and method for segmentation dependent lossy and lossless compression. In *US patent number 6198850*.

[2] S. M. Daniel J. Abadi and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *SIGMOD Proceedings*, pages 671–682. ACM, June 2006.

[3] M. B. de Carvalho and E. A. B. da Silva. A universal multidimensional lossy compression algorithm. In *International Conference on Image Processing*. IEEE, October 1999.

[4] D. A. Huffman. A method for the construction of minimum-redundancy codes. In *Proceedings of the Institute of Radio Engineers*.

[5] D. E. Knuth. Dynamic huffman coding. In *Journal of Algorithms*, volume 6, pages 163–180. ACM, 1985.

[6] N. Koudas. Space efficient bitmap indexing. In *Conference on Information and Knowledge Management*. ACM, 2000.

[7] M. F. Oberhumer. http://www.lzop.org/, 2008. Online; accessed 24 January 2008.

[8] W. B. Pennebaker and J. L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand, New York, NY, 1992.

[9] C. E. Shannon. *The Mathematical Theory of Communication*. University of Illinois Press, Champaign, IL, 1948.

[10] S. D. S. Survey. http://www.sdss.org/.

[11] Wikipedia. Run-length encoding — Wikipedia, the free encyclopedia, 2007. [Online; accessed 3-November-2007].

[12] Wikipedia. Lempel-ziv-welch — Wikipedia, the free encyclopedia, 2008. [Online; accessed 3-September-2008].

[13] Wikipedia. Pareto efficiency— Wikipedia, the free encyclopedia, 2008. [Online; accessed 3-September-2008].