Regressional Compression

A dissertation on a novel and unexplored approach

Akshay Vasudeva Rao

1 Introduction

This paper presents a new and novel approach to file compression with linear regression at its very core. It attempts to devise a compression method capable to mapping any piece of information to just six numbers.

The paper investigates the said problem with a great deal of depth, starting from the mathematical basis and the encoding procedures to the actual core algorithm and the asymptotic complexities of the underlying procedures.

The proposed algorithm uses the concept of linear regression along with other tools in order to develop a well-behaved one-to-one mapping from the information to be compressed to the above mentioned six numbers, i.e. the compressed information.

The dissertation begins with a brief study of the mathematical basis of linear regression. It goes on to explain the proposed model in detail, examining and detailing key aspects such as encodings, compression procedure, and of course the inverse procedure, i.e. the decompression procedure. Next, the paper attempts to evaluate the proposed model, making note of its merits, demerits and numerous other observations. Finally, the papers concludes in a section detailing probable use cases for the proposed model along with the future scope of work and optimizations on the same.

2 Literature Review

In this section, a mathematical basis for linear regression is established along with the theoretical underpinnings of Heap's algorithm (an important tool in the implementation of the proposed compression model).

2.1 Linear Regression

Our data consists of n paired observations of the predictor variable X and the response variable Y , i.e., (x_1, y_1), . . .(x_n, y_n). We wish to fit the model

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

where E [ε|X = x] = 0, Var [ε|X = x] = σ^2 , and ε is uncorrelated across measurements.

Now, the basic matrices are defined.

All the observations of the response are grouped into a single column (n x 1) matrix y.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Similarly, we group both the coefficients into a single vector (i.e., a 2 × 1 matrix).

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

Next, we group together the observations of the predictor variable.

$$\mathbf{x} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

This is an n×2 matrix, where the first column is always 1, and the second column contains the actual observations of X. We have this apparently redundant first column because of what it does for us when we multiply x by β:

$$\mathbf{x}\beta = \begin{bmatrix} \beta_0 + \beta_1 x_1 \\ \beta_0 + \beta_1 x_2 \\ \vdots \\ \beta_0 + \beta_1 x_n \end{bmatrix}$$

That is, xβ is the n × 1 matrix which contains the point predictions.

Now, a formalization of the mean squared error is obtained.[1]

At each data point, using the coefficients β results in some error of prediction, so we have n prediction errors. These form a vector:

$$e(\beta) = y - x\beta$$

$$MSE(\beta) = \frac{1}{n}\sum_{i=1}^{n} e_i^2(\beta)$$

$$MSE(\beta) = \frac{1}{n}\mathbf{e}^T\mathbf{e}$$

$$MSE(\beta) = (1/n)((y - x\beta)^T\ (y - x\beta))$$

$$MSE(\beta) = (1/n)((y^T - \beta^T x^T)(y - x\beta))$$

Thus,

$$MSE(\beta) = (1/n)((y^T y - 2\beta^T x^T y + \beta^T x^T x\beta)$$

Now, the MSE is to be minimized.[2]

First, the gradient of MSE is found with respect to β.

$$\nabla MSE(\beta) = (1/n)((\nabla y^T y - 2\nabla\beta^T x^T y + \nabla\beta^T x^T x\beta)$$

$$= (1/n)(x^T x^{\beta} - x^T y)$$

We now set this to zero at the optimum, $\hat{\beta}$.[3]

$$\mathbf{x}^T\mathbf{x}\widehat{\beta} - \mathbf{x}^T\mathbf{y} = 0$$

Solving,

$$\widehat{\beta} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}$$

$$\hat{\beta}_1 = \frac{c_{XY}}{s_X^2} = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2}$$

$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1\overline{x}$$

Thus, we now have a formalization of the core of the algorithm.

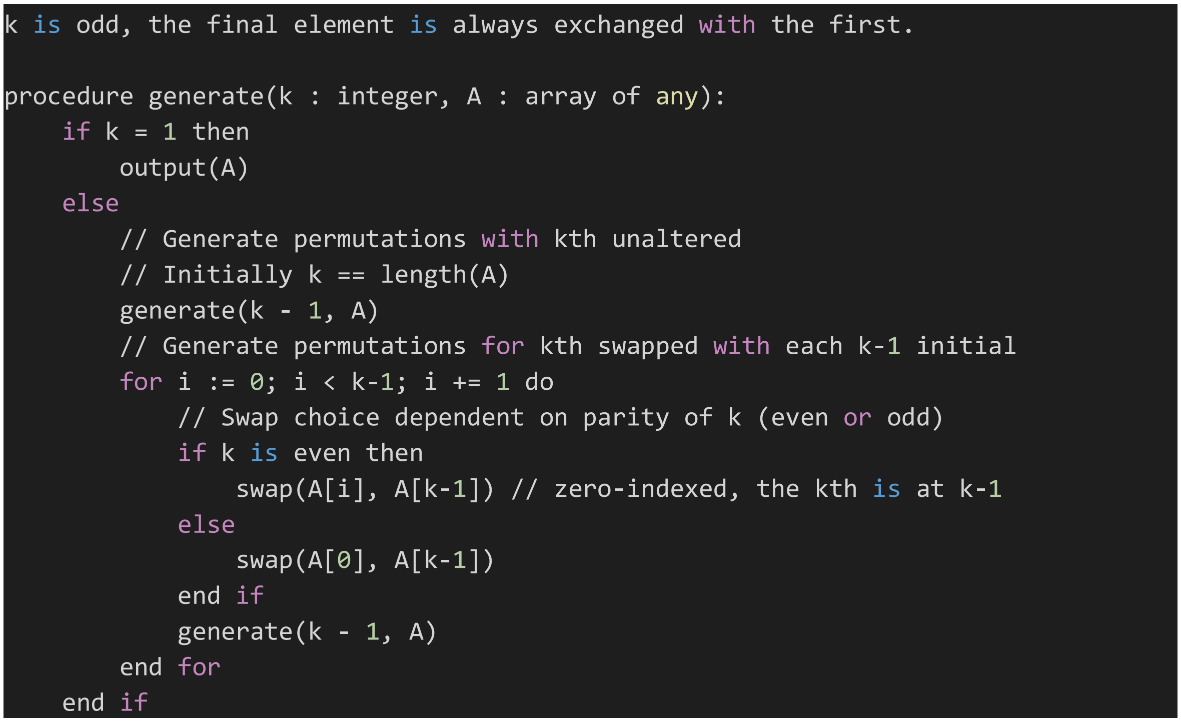
2.2 Heap’s Algorithm

Heap's algorithm generates all possible permutations of n objects. It was first proposed by B. R. Heap in 1963.The algorithm minimizes movement: it generates each permutation from the previous one by interchanging a single pair of elements; the other n−2 elements are not disturbed.[4][5]

For a collection C containing n different elements, Heap found a systematic method for choosing at each step a pair of elements to switch in order to produce every possible permutation of these elements exactly once.

Described recursively as a decrease and conquer method, Heap's algorithm operates at each step on the k initial elements of the collection. Initially k==n and thereafter k<n. Each step generates the k! permutations that end with the same n-k final elements. It does this by calling itself once with the kth element unaltered and then k-1 times with the (kth) element exchanged for each of the initial k-1 elements. The recursive calls modify the initial k-1 elements and a rule is needed at each iteration to select which will be exchanged with the last. Heap's method says that this choice can be made by the parity of the number of elements operated on at this step. If k is even, then the final element is iteratively exchanged with each element index. If k is odd, the final element is always exchanged with the first.

Algorithm:



```
k is odd, the final element is always exchanged with the first.

procedure generate(k : integer, A : array of any):
    if k = 1 then
        output(A)
    else
        // Generate permutations with kth unaltered
        // Initially k == length(A)
        generate(k - 1, A)
        // Generate permutations for kth swapped with each k-1 initial
        for i := 0; i < k-1; i += 1 do
            // Swap choice dependent on parity of k (even or odd)
            if k is even then
                swap(A[i], A[k-1]) // zero-indexed, the kth is at k-1
            else
                swap(A[0], A[k-1])
            end if
            generate(k - 1, A)
        end for
    end if
```

2.3 R-squared coefficient

R-squared evaluates the scatter of the data points around the fitted regression line. It is also called the coefficient of determination, or the coefficient of multiple determination for multiple regression. For the same data set, higher R-squared values represent smaller differences between the observed data and the fitted values.[6]

R-squared is the percentage of the dependent variable variation that a linear model explains.

$$R^2 = \frac{\text{Variance explained by the model}}{\text{Total variance}}$$

3 Present Investigation

In this section, a detailed scheme for a compression/decompression system is presented.

3.1 Encoding scheme

The objective of the encoding scheme is that it must transform a set of characters into a set of points on a 2-dimensional plane so that the best-fit line may subsequently be found.

Table 3.a Y-encodings for text files

Character	Y-encoding	Character	Y-encoding
‘ ’ (space)	0	‘x’	24
‘a’	1	‘y’	25
‘b’	2	‘z’	26
‘c’	3	‘A’	27
‘d’	4	‘B’	28
‘e’	5	‘C’	29
‘f’	6	‘D’	30
‘g’	7	‘E’	31
‘h’	8	‘F’	32
‘i’	9	‘G’	33
‘j’	10	‘H’	34
‘k’	11	‘I’	35
‘l’	12	‘J’	36
‘m’	13	‘K’	37
‘n’	14	‘L’	38
‘o’	15	‘M’	39
‘p’	16	‘N’	40
‘q’	17	‘O’	41
‘r’	18	‘P’	42
‘s’	19	‘Q’	43
‘t’	20	‘R’	44
‘u’	21	‘S’	45
‘v’	22	‘T’	46

‘w’	23	‘U’	47
‘V’	48	‘{’	67
‘W’	49	‘}’	68
‘X’	50	‘[’	69
‘Y’	51	‘]’	70
‘Z’	52	‘\’	71
‘!’	53	‘|’	72
‘@’	54	‘:’	73
‘#’	55	‘;’	74
‘$’	56	‘”’	75
‘%’	57	‘’’	76
‘^’	58	‘<’	78
‘&’	59	‘>’	79
‘*’	60	‘,’	80
‘(’	61	‘.’	81
‘)’	62	‘?’	82
‘-’	63	‘/’	83
‘_’	64	‘`’	84
‘+’	65	‘~’	85
‘=’	66	0 - 9	85 + (1 to 10)

Table 3.b Y-encodings for binary files

Bit	Y-encoding
0	0
1	1

The X-encoding of a given character/bit is simply its position in the given file, where position is indexed from 0.

3.2 Compression

During compression, the first and most immediate task is to identify the type of the file to be compressed (text/binary) so that a suitable encoding scheme can be implemented. Next, the file must be parsed and encoded suitably, generating two lists: X_encodings and Y_encodings. Now, the line of best-fit must be found $\forall Pi$ where $Pi \in \{x_i, y_i \mid x_i \in$ X_encodings, $y_i \in$ Y_encodings$\}$, i.e. for all the points in the encoded file. This produces the two coefficients β_0_est and β_1_est.

Next, the goodness-of-fit of the line is calculated. We now have r_sq.

Finally we must find Heaps_occurrence, which is a number between 1 and factorial of the number of characters/bits in the original uncompressed file which is representative of the position of occurrence of the string in Heap's algorithm assuming that it was initiated with the alphabetically ordered string.

It is also important to make a note of the original file's length, say originalLen.

Algorithm:

```
procedure RegCompress(fileContent: string, fileType: int)
    X_encodings = X_encode(fileContent)
    Y_encodings = Y_encode(fileContent, fileType)
    betaZero_est, betaOne_est = linearRegression(X_encodings, Y_encodings)
    r_sq = goodness_of_fit(X_encodings, Y_encodings, betaZero_est, betaOne_est, "
r-squared")
    Heaps_occurrence = modifiedHeaps(Y_encodings.sort())
```

3.3 Decompression

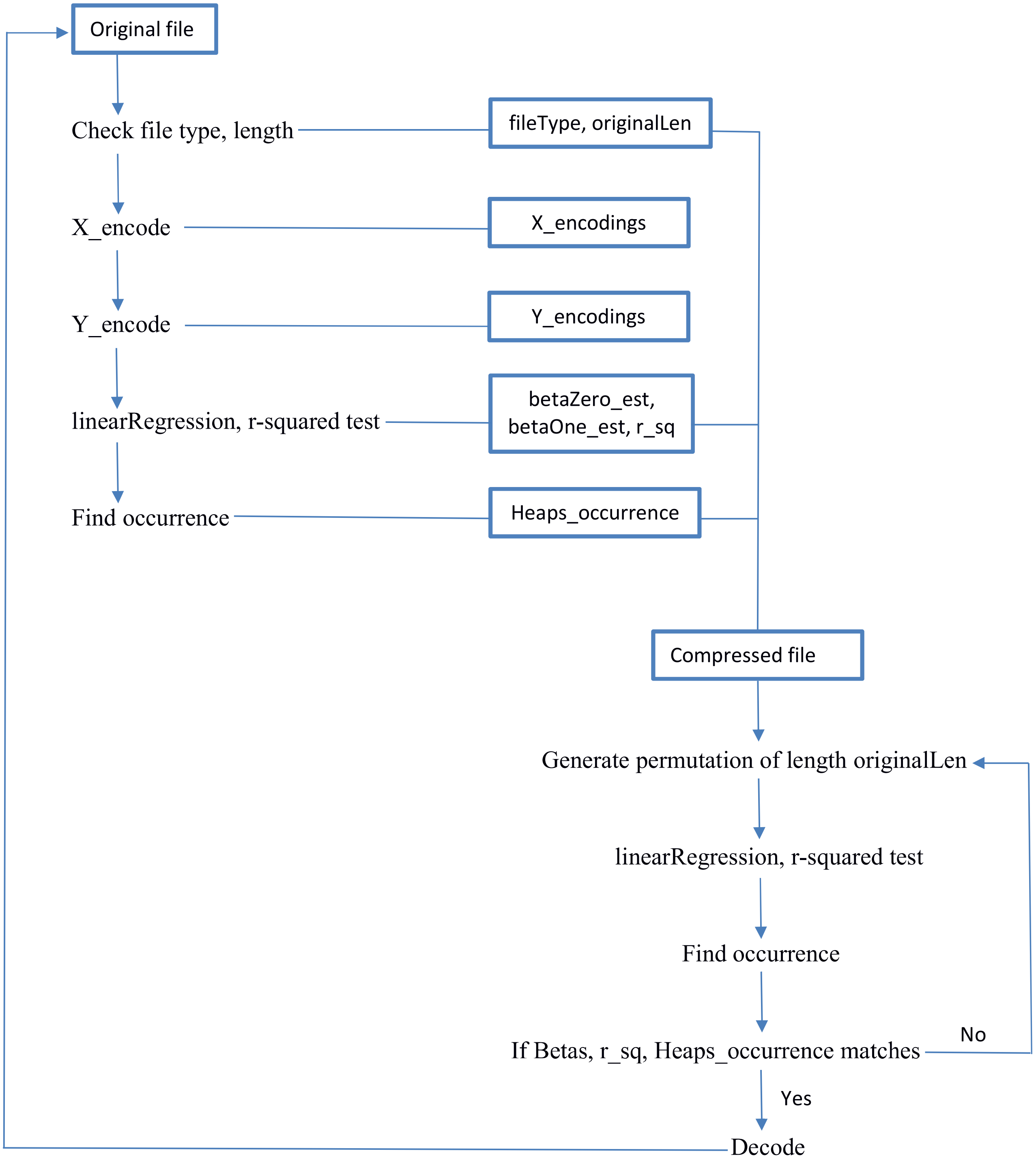
The compressed file contains 6 elements. Namely, β_0_est, β_1_est, r_sq, Heaps_occurrence, originalLen and fileType. The aim is to now retrieve the original file from just 6 elements.

Decompression is accomplished by a brute-force approach that checks all possible strings of length originalLen, generating the best-fit lines for each and checking the constraints β_0_est, β_1_est, r_sq and Heaps_occurrence.

This yields the original file as the mapping is one-to-one.

4 Results and Discussions

4.1 Procedural flow


Original file
Check file type, length
fileType, originalLen
X_encode
X_encodings
Y_encode
Y_encodings
linearRegression, r-squared test
betaZero_est, betaOne_est, r_sq
Find occurrence
Heaps_occurrence
Compressed file
Generate permutation of length originalLen
linearRegression, r-squared test
Find occurrence
If Betas, r_sq, Heaps_occurrence matches
No
Yes
Decode

4.2 Graphical representation

For a natural and intuitive understanding of a concept, a graphical, more tangible representation is an important tool.

Figure 4.a Representation for fileType = 0 i.e "text"

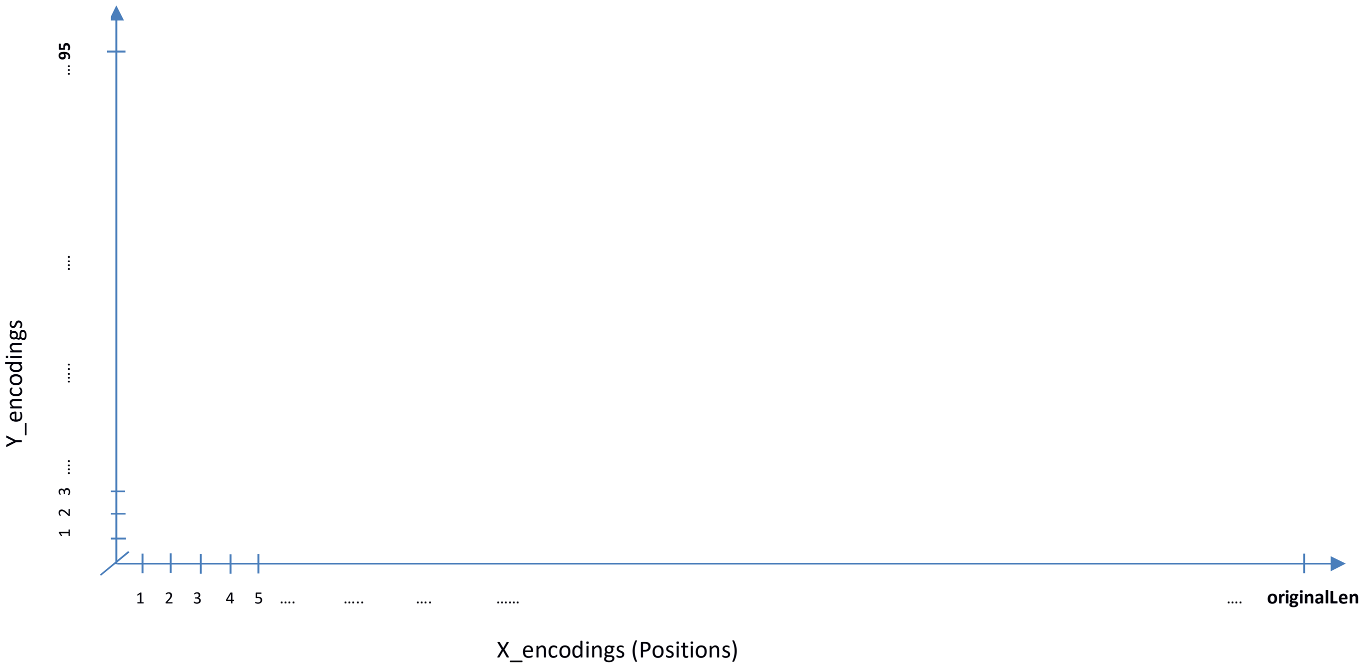
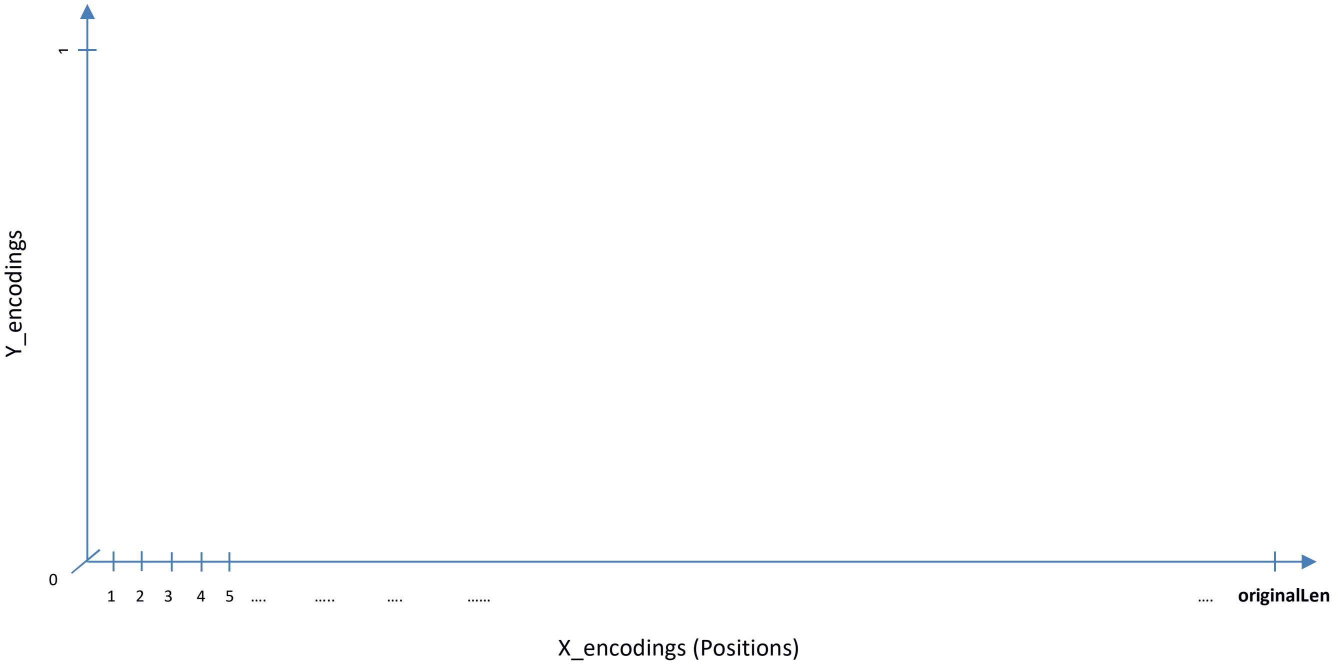
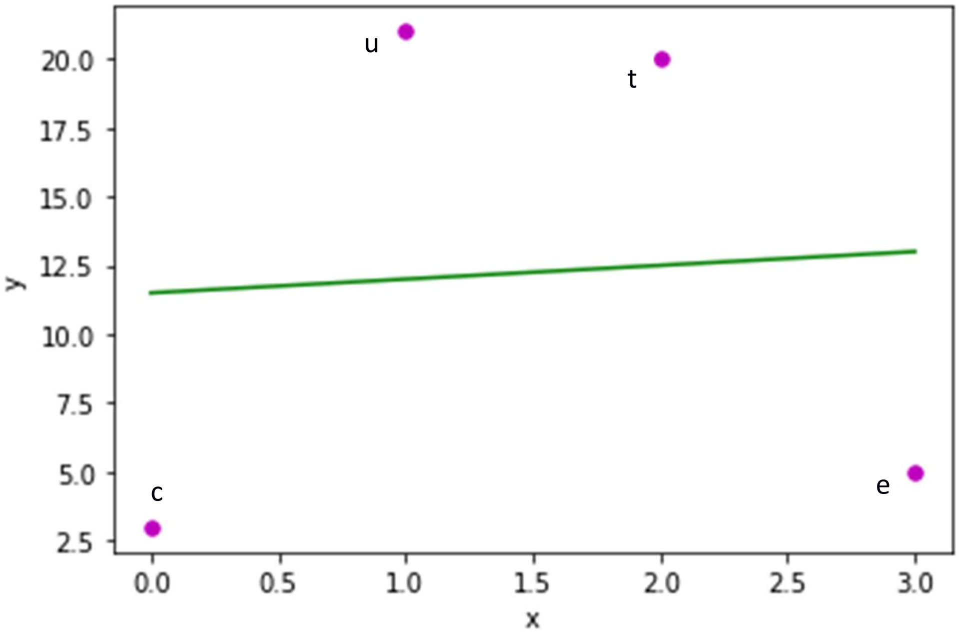


Figure 4.b Representation for fileType = 1 i.e "binary"



4.3 Example

In this sub-section, as an example, the text string "cute" is compressed.



originalLen = 4

fileType = 0

betaZero_est = 11.5

betaOne_est = 0.5

r_sq = 0.0045495905368516665

Heaps_occurrence = 16

Hence, the compressed file contains the above 6 elements.

4.4 Applicability Condition

Since the goal of compression is to obtain an ad hoc file with all the information of the original file but considerably smaller in size such that the decrease in size justifies the cost of implementing the compression (and subsequently the decompression), the viability of the process must be studied.

Condition: Size of compressed file < Size of original file

- ⇨ sizeof(typeof(originalLen)) + sizeof(typeof(fileType)) + sizeof(typeof(betaZero_est)) + sizeof(typeof(betaOne_est)) + sizeof(typeof(r_sq)) + sizeof(typeof(Heaps_occurrence)) < Size of original file
- ⇨ sizeof(int) + sizeof(int) + sizeof(double) + sizeof(double) + sizeof(double) + sizeof(int) < Size of original file
- ⇨ **Size of original file > 36 bytes**

(Considering sizes of data types as specified by C11 documentation)

5 Summary and Conclusions

This section contains some concluding remarks offering some key insights into the compression scheme detailed so far.

5.1 Compression Statistics

According to the outlined scheme, an uncompressed file, no matter how large* will always be brought down to a compressed file of only 36 bytes.

Compression ratio = Uncompressed size / Compressed size = N/36

The scheme is seen to exhibit a compression ratio of **N:36** where N is the size of the input file (uncompressed file) in bytes.

Space saving = 1 – (Compression ratio)$^{-1}$ = 1 – 36/N = (N - 36)/N

The space saving is an impressive **(N – 36):N**.

***Note:** As the input file size exceeds a certain size, it is possible that the 6 elements will fail to map to the original file in their standard size (of the individual data types). At this point, the elements may have to grow marginally to accommodate the larger file sizes. However, this is not foreseeably a problem as the rate of growth of element sizes (and therefore, size of the compressed file) will be substantially lesser than the rate of growth of the corresponding input files. Hence, asymptotically speaking, this will not be an issue. However, the quoted compression ratio is likely to grow, but, this growth would be well-behaved and non-obstructive in the applicability and deployment of the scheme.

5.2 Optimizations

1. Building custom encodings for specialized text files: Specialized text files may only contain a small subset of the standard text-encoding character set. By, building a new encoding table dynamically and on the fly, the brute-force decompression algorithm works at a highly accelerated pace.
   Say the subset size for the specialized text file is only S characters long (where S < 95). Then, the speedup observed would approximately be $\mathbf{95^N/S^N}$. The cost incurred, however, would be the additional space occupied by the custom encoding.

2. Additional elements indicating lower and upper limit of standard encoding: Merely an alternative to point-1 where the sub-set is a continuous (or nearly continuous) range of elements of the standard encoding.

3. Superior decompression algorithm: The decompression procedure is a major bottleneck in the entire scheme. Developing an approach that solves the problem without using the brute-force technique would alleviate this problem to a great extent.

5.3 Applications

The most foreseeable application of the compression technique is Archiving, where files are usually very large, numerous in count and not very frequently accessed, thereby enabling them to be compressed in a format that is not very yielding to the process of decompression. These applications can afford the cost of slow decompression.

5.4 Future scope

1. Studying the suggested implementations as detailed in section 5.2.
2. Formalizing the study of the growth of the compression ratio with the file size and subsequently, the space saving.
3. Practical deployment in real-time systems.
4. Study compression behavior on previously compressed data when:
    a. The previously compressed data was compressed using the same scheme.
    b. The previously compressed data was compressed using a different compression scheme.

6 Bibliography

[1] Lecture 13: Simple Linear Regression in Matrix Format, 36-401, Section B, Fall 2015, Carnegie Mellon University.

[2] Gilbert Strang. Linear Algebra and it's Applications, Fourth Edition.

[3] Sanford Weisberg. Applied Linear Regression, Third Edition.

[4] Heap, B. R. (1963). "Permutations by Interchanges" - The Computer Journal.

[5] Sedgewick, R. (1977). "Permutation Generation Methods" - ACM Computing Surveys.

[6] Hughes, Ann; Grawoig, Dennis (1971). Statistics: A Foundation for Analysis.