Elementary Cellular Automata as an Error Minimized Hash

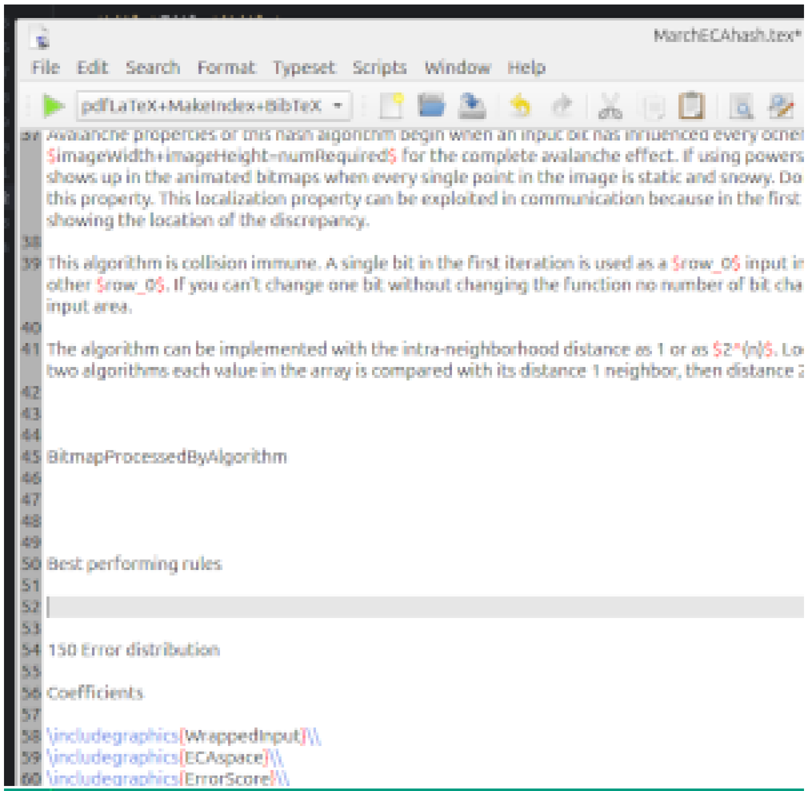
Daniel McKinley

November 2024

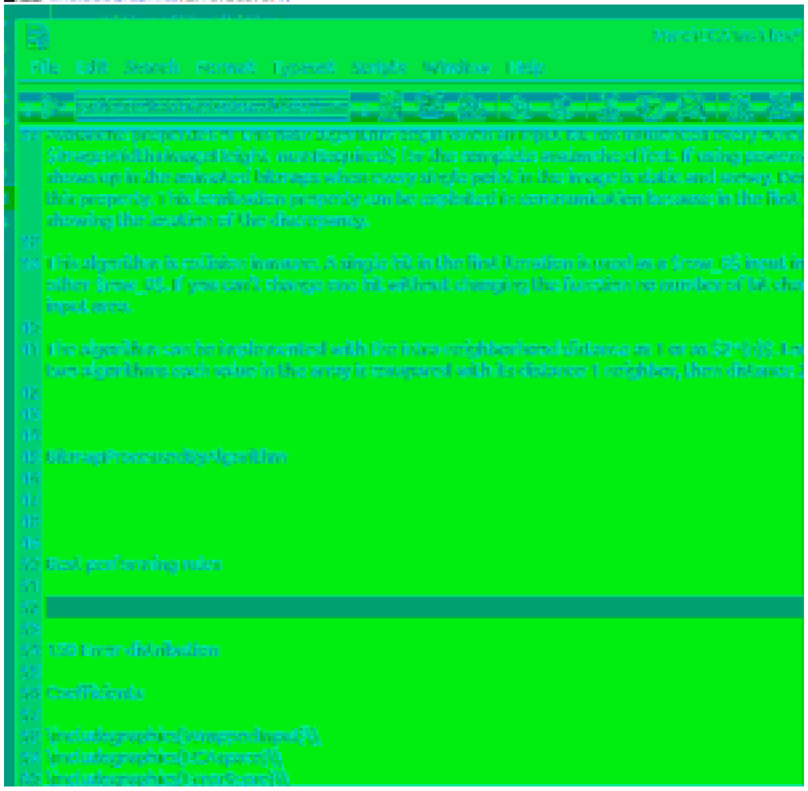
1 Introduction

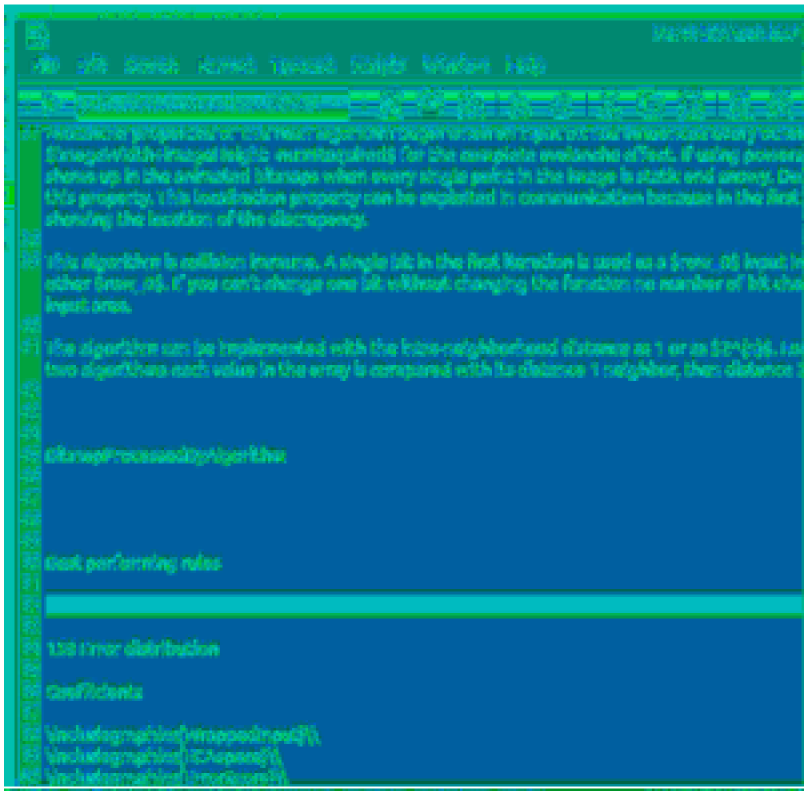
Elementary cellular automata (ECA) are 8 bit extensions of 4 bit logic gate truth tables, done linearly in parallel. [2] Here a subset of 8 of the 256 ECA rules are explored as a cryptographic hash function that tracks the Fast Fourier Transform (FFT) and Fast Walsh-Hadamard Transform. General algorithm, specific ECA rules, and aggregate properties are discussed. It is implemented in Java at [1] along with a sample of the algorithm operating on a given bitmap image at the website in the references.

2 Main Algorithm


MarchECAhash.tex*
File Edit Search Format Typeset Scripts Window Help
pdfLaTeX+MakeIndex+BibTeX

```
Avalanche properties of this hash algorithm begin when an input bit has influenced every other
$imageWidth+imageHeight-numRequired$ for the complete avalanche effect. If using powers
shows up in the animated bitmaps when every single point in the image is static and snowy. Do
this property. This localization property can be exploited in communication because in the first
showing the location of the discrepancy.

This algorithm is collision immune. A single bit in the first iteration is used as a $row_0$ input in
other $row_0$. If you can't change one bit without changing the function no number of bit cha
input area.

The algorithm can be implemented with the intra-neighborhood distance as 1 or as $2^{n}$. Lo
two algorithms each value in the array is compared with its distance 1 neighbor, then distance 2



BitmapProcessedByAlgorithm




Best performing rules



150 Error distribution

Coefficients

\includegraphics{WrappedInput}\\
\includegraphics{ECAspace}\\
\includegraphics{ErrorScore}\\
```





[illegible]
[illegible] for the complete avalanche effect. If using powers
shows up in the animated bitmap when every single point in the image is static and snowy. De
this property. This localization property can be exploited in communication because in the first
showing the location of the discrepancy.

This algorithm is collision immune. A single bit in the first iteration is used as a row_0 input in
other row_0. If you can't change one bit without changing the function no number of bit cha
input ones.

The algorithm can be implemented with the inter-neighborhood distance as 1 or as 2^{n}. [illegible]
two algorithms each value in the array is compared with its distance 1 neighbor, then distance 2

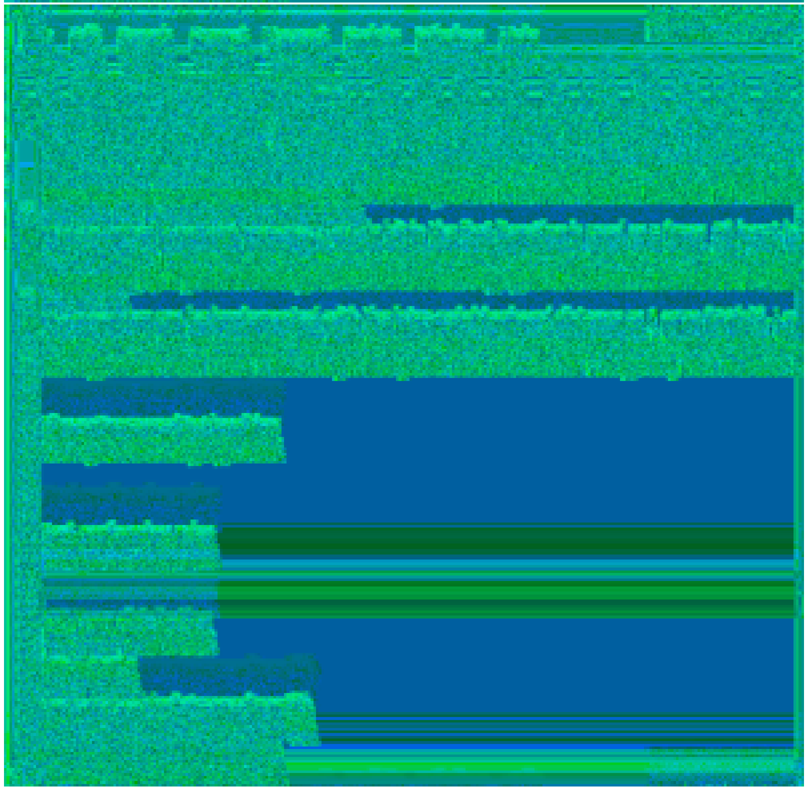
BitmapProcessedByAlgorithm

Best performing rules

13.3 Error distribution

Coefficients

\includegraphics{wrappedinput}\\
\includegraphics{CAspace}\\
[illegible]



Include graphics for: Input example, Codeword output example, Error map example
BitmapProcessedByAlgorithm

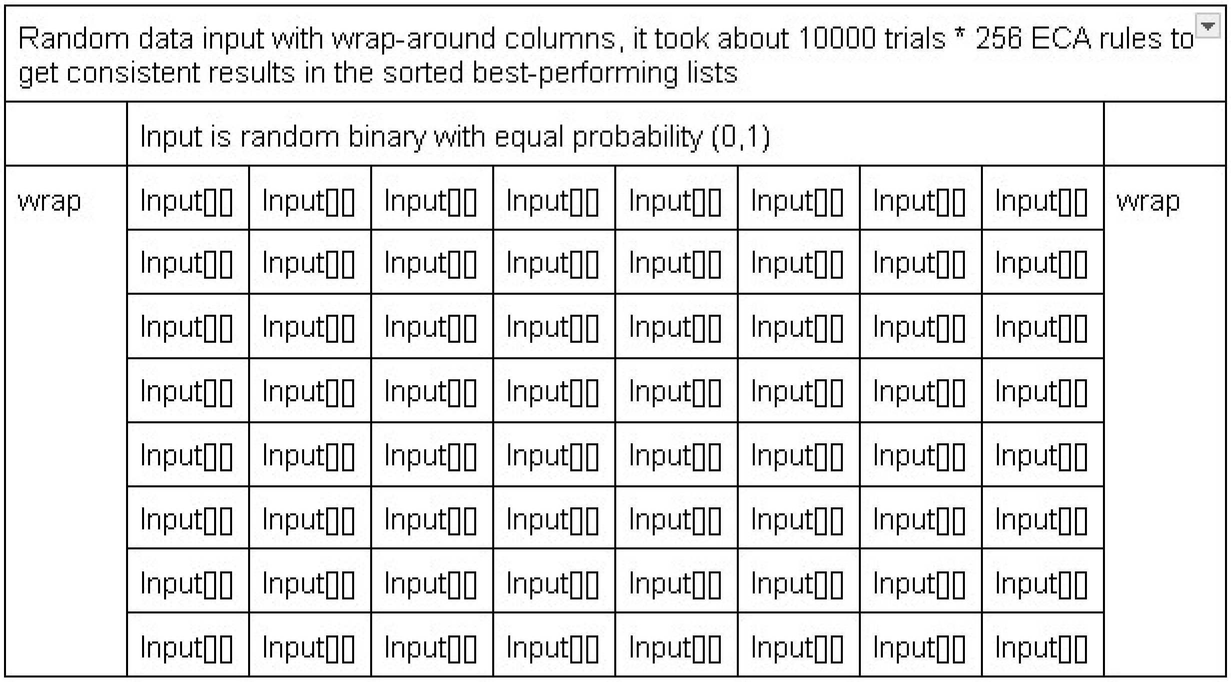
there are $2^(16) = 65536$ binary 4x4 arrays

where there are $2^{(}4)$ possible row_0 neighborhoods for a given ECA rule
each of these 16 input-ECAoutput arrays are scored by

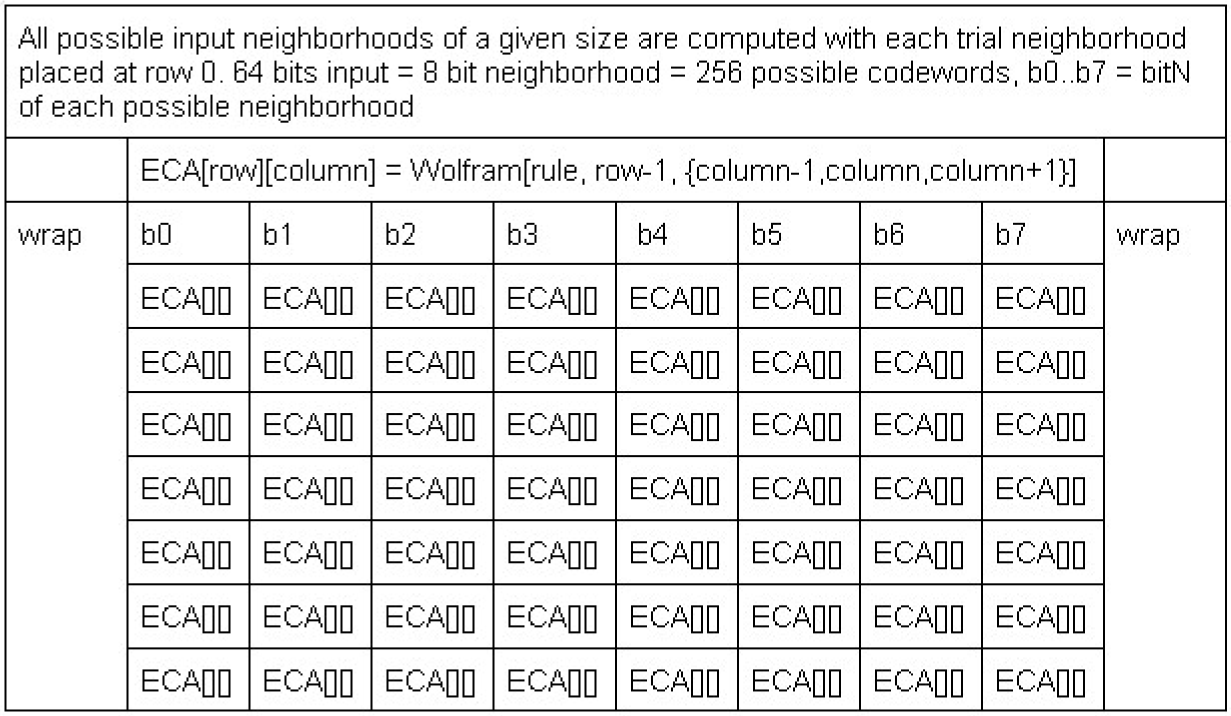
$$\sum_{r=0}^{3}\sum_{c=0}^{3} 2^r(compressionAttempt_{rc} \oplus original_{rc})$$

the value of the neighborhoods of the minimum and maximum of these 16 sums
are noted as the solution for the original binary matrix

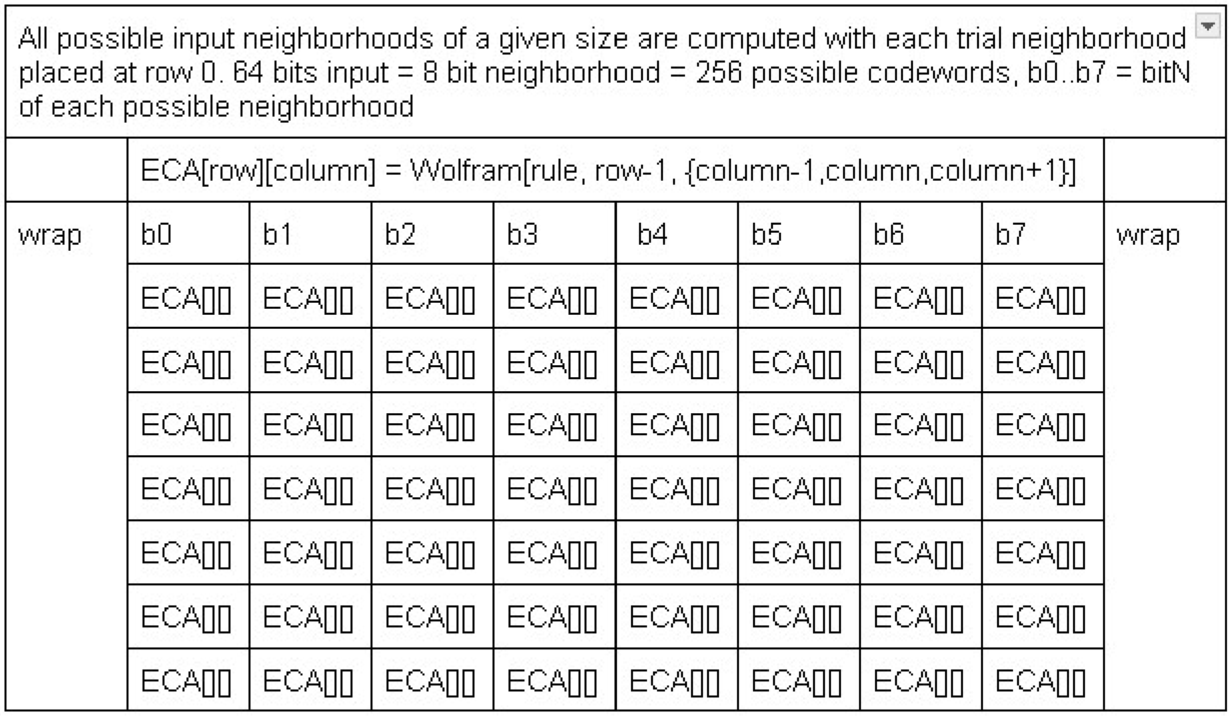
The algorithm works on $2^n * 2^n$ square matrices,2x2, 4x4, 8x8. For all $2^{(}16) = 655536$ possibilities of a 4x4 binary array, create a wrapped 4x4 array, use row 0 as all possible $2^4 = 16$ input neighborhoods, calculate the remaining three rows for all 16 possible input values using a Wolfram code. For each of all these 16 possible inputs, score the 4x4 array with a weighted sum of discrepancies between this codeword-produced output and the original 4x4 matrix Input[][]. The lowest and highest scoring neighborhood are then two length-four codewords for a possible neighborhood of size 4x4. The algorithm's solutions for all neighborhoods of a certain size become a hexadecimal Wolfram code for a small QR code. Changing the size is not fully studied due to computing constraints of mapping all $2^{(}64)$ possible 8x8 matrices on up. The paper assumes a square input matrix, the algorithm is implemented for any number of rows and columns.



Random data input with wrap-around columns, it took about 10000 trials * 256 ECA rules to get consistent results in the sorted best-performing lists									
	Input is random binary with equal probability (0,1)								
wrap	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	wrap
	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	
	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	
	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	
	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	
	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	
	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	
	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	Input▯▯	



All possible input neighborhoods of a given size are computed with each trial neighborhood placed at row 0. 64 bits input = 8 bit neighborhood = 256 possible codewords, b0..b7 = bitN of each possible neighborhood									
	ECA[row][column] = Wolfram[rule, row-1, {column-1,column,column+1}]								
wrap	b0	b1	b2	b3	b4	b5	b6	b7	wrap
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	



All possible input neighborhoods of a given size are computed with each trial neighborhood placed at row 0. 64 bits input = 8 bit neighborhood = 256 possible codewords, b0..b7 = bitN of each possible neighborhood									
	ECA[row][column] = Wolfram[rule, row-1, {column-1,column,column+1}]								
wrap	b0	b1	b2	b3	b4	b5	b6	b7	wrap
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	
	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	ECA▯▯	

Doing this algorithm for every cell in a bitmap quadruples its size. After an initilization frame, the algorithm is applied to every cell in the bitmap with the neighborhood square's rows and columns operating on a $2^{d}epth$ basis. Frame one's neighborhoods are next door neighbors, frame two's neighbors are two doors away, frame four's are four, then eight and so on. This comparison of neighbors of distance 2^{n} is similar to the FFT and Fast Walsh-Hadamard process.

This algorithm can both minimize the errorScore in a lossy compression, and maximizing the errorScore.

There are 8 [0,15,51,85,170,204,240,255] of the 256 8 bit ECA truth tables that display the properties of unique codewords for any given input and perfectly even distribution of codewords. 0 and 255 are included because in the 4 rows of the output matrix, 1 is neighborhood input and 3 are output, and still produce an errorScore. This subset works with both errorScore minimization and errorScore maximization. These two min max 8-tuples are implemented as a cryptographic hash.

Avalanche properties of this hash algorithm begin when an input bit has influenced every other input bit. If using sequential hashes, you need $imageWidth + imageHeight = numRequired$ for the complete avalanche effect. If using powers of two the avalanche effect begins at $log_2(width) + log_2(height)$. This property shows up in the animated bitmaps when every single point in the image is static and snowy. Doing fewer iterations than the avalanche's diffusion threshold but more than zero limits this property. This localization property can be exploited in communication because in the first few iterations the data is irreversible enough to be imperfectly unencryptable while showing the location of the discrepancy.

This algorithm's collision resistance is under investigation. While every input has a unique solution at any level of iteration, it looks like immediately local collisions can be engineered though the properties of engineering deeper iterative collisions over wider areas don't scale well.

The algorithm can be implemented with the intra-neighborhood distance as 1 or as $2^(n)$. Loops in the implementation using the powers 2 directly parallel the FFT and FWH. In those two algorithms each value in the array is compared with its distance 1 neighbor, then distance 2, then distance 4, then 8, then 16 and so on.

The weight in the errorScore sum can be $2^(row($ or $2^(column)$, both produce the same set of 8 tuples with the unique solution property, though other ECA rule's properties don't necessarily carry over.

The size of the input array can be easily be any power of 2 squared. Wolfram code lengths of $2^(16)$ are acceptable, lengths of $2^(64)$ are in principle doable but not practical. The 8 tuple's uniqueness and distribution properties may apply at size 8 squared, it may not; it is only completely tested for size 4. Running statistics on random samples of size 8 have not been done yet.

Best performing rules

150 Error distribution

Reconstruction

voting() subfunction to increase accuracy

References

[1] Daniel McKinley. github.com/dmcki23/, 2024.

[2] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.