Turkish Journal of Computer and Mathematics Education *Vol.12 No.2 (2021), 3442-3450*

Research Article

A Novel Approach to Store an Image in QR Code

[1]Nuchu Yeswanth Surya Srikar, [2]Narisetty Srinivasa Rao, [3]Pothana Vamsi Naidu

[1] Software Engineer, Reputation.com, Hyderabad, India, suryanucchu@gmail.com
[2,3] Assistant Professor, Department of CSE, Lakireddy Bali Reddy College of Engineering (Autonomous), Mylavaram, Krishna District, Andhra Pradesh – 521230

Article History: Received: 11 January 2021; Accepted: 27 February 2021; Published online: 5 April 2021

Abstract: Quick Response Code is a machine-readable, two-dimensional barcode consisting of an array of black and white squares which can be scan quickly by any smartphone. It allows encoding over 4000 pulse characters in a two-dimensional barcode. It is used to store a small amount of information like web URL, item data, phone numbers and multimedia data. Due to restricted size in it, QR codes are presently limited in the extent to store the data. But the data in image format uses more space, if you want to store image data in QR code you must enlarge the storage capacity of QR code. In this paper, we propose a very simple form of lossy data compression, in which runs of data is dividing into blocks of equals size. The entire block is stored as a character rather than as the original run with generic HashMap which makes more robust and provides all sorts of security. Finally, our results are compared with other techniques to differentiate the optimality, efficiency of the new technique for producing optimal QR codes.

Keywords: Quick Response Code, Image Compression, Security, Encryption, Decryption

1. Introduction

QR Code [1] is a machine-generated and readable code generally used to store URL or other information. It is a 2D barcode which consists sequence of black squares over a white background arranged in a grid format. The barcode or Quick Response code is an important interface between real life and the virtual world nowadays [2].

Figure (a)

There are five different types of QR Codes

I. QR Code Model 1&2
II. Micro QR Code
III. iQR Code
IV. SQRC
V. Frame QR.

In this, we are using QR Code Model 1&2 as our data store.

Typical QR Code can store up to 3 KB of data. It is made up of rows and columns which form a sequence of small squares where each square is called a module [3]. Maximum there can be 177 rows and 177 rows that are 31,329 modules. Image is a 2D array of pixels where each pixel ranges from 0 – 225.

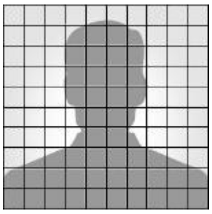


Figure (b)

There are three types of images.

I. Binary Images [2]
II. Grey-scale Images [3]
III. Color Images [4]

In this, we are using Gray-scale Image for explaining the flow and color images for testing.

2. DATA ENCODING AND DECODING

Encoding

We can enhance the capacity of the QR Code by using compression techniques [5]. In this, we are introducing a new lossy compression [6] algorithm named SN-6 for compressing an image to store in QR Code. Figure(c) describes the flow to keep a shot in the QR Code.

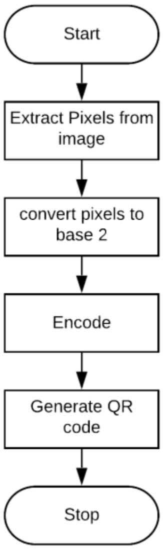


Figure (c)

Extract pixels from Image:

Image is formed by a sequence of pixels [7] arranged in a two-dimensional array. Each pixel is ranged from 0-255 for monochrome image and color image; it is the combination of RGB each will go from 0-255. Figure (d) describes the above.

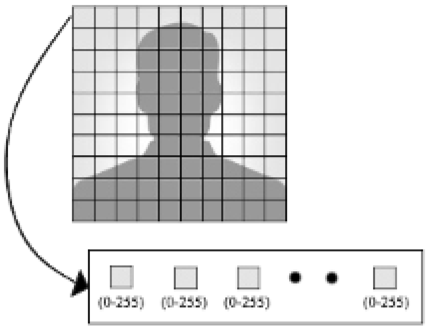


Figure (d)

Convert pixels to base 2:

After extracting all the pixels from the image and storing them in a byte array, we need to convert all the pixel values into base 2. Figure (e) describes the above.

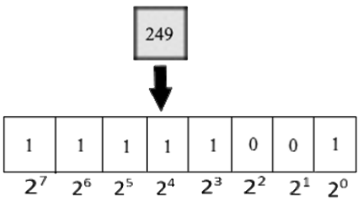


Figure (e)

Encode and Compress:

After converting the byte array to bit array, pass it as an input to SN-6.

SN-6 Compression:

The value of pixel will increase exponentially from right to left. So, if we remove the last few bites, it won't

impact the overall pixel value at a greater extent. For example,
Two hundred forty-nine can be represented as 11111001. If we remove the last four digits, then the value of the pixel will be 240. Here the overall impact factor is every less. Figure (f) describes the above.

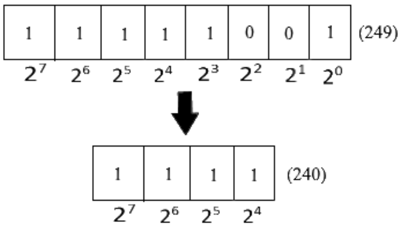


Figure (f)

Divide the pixel into two parts, append the first part to a stream and XOR the second half.
Now divide the runs of data into blocks of equal size and replace each block with the corresponding value from Hash Map table. Figure (g) describes the above.

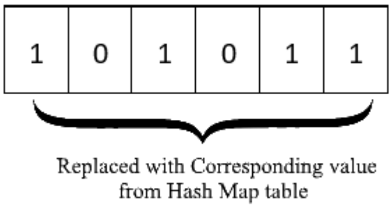


Figure (g)

Before constructing the Hash Map, we need to fix the block size, which will act as a key in our Hash Map and corresponding value can be any printable ASCII character. Here we consider only printable characters because at the end we need to store them in a QR Code which can hold only printable characters.
Total ASCII printable characters [8] are 97, which is in between $2^6 < 97 < 2^7$. So, the best possible size for a block is six why because if the block is seven, then the max value will be 128 (1111111) which exceeds 97. Here we can't replace the above value (128) with any printable character.

Constructing Hash Map:

After finalizing the block size to 6, the minimum value for a block is 00000, which is 0, and the maximum value is 111111, which is 64. With 97 printable characters, we can form 64*97 Hash Maps where each HashMap contains 64 elements. Figure (h) describes one such Hash Map out of 64*97.

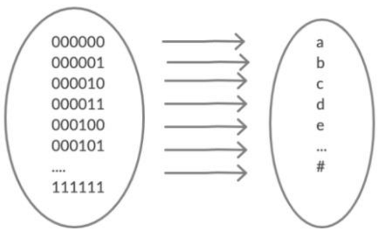


Figure (h)

To make SN-6 more secure, we are concatenating all the keys in order and form a stream of characters and sending as input to any of the secured hashing algorithms to get 32-bit Hash Key. This key will change if we change the order of keys in HashMap.
Total we have 64 keys in HashMap [9]. So, 64 keys can be arranged in 64! times. Based on this, we can generate 97*64*64! HashMap and will take exponential time to find the correct HashMap used.

No of printable chars * no of keys * total number of arrangements in keys

Generate QR Code:

After replacing all the blocks with corresponding hash value and attaching the hash key at the end of the string, we need to store the entire series into a QR Code. You can prefer any language to do this. I used the Python library named "*QR-code*" to generate Figure (i).

Figure (i)

Decoding

In this process, we decode and decompress the QR Code using SN-6 decompression technique. We follow the same methods as above but in the backwards direction, as shown in the Figure (j).

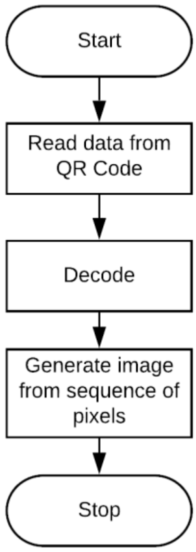


Figure (j)

Read data from QR Code:

Extract the data from the QR Code using any physical device or custom code. The size of the QR Code may vary based on the data. Store them in a stream of characters, as shown in Figure (k).

Figure (k)

Decode and Decompress:

Separate the last 32bit characters from the stream and compare it with the newly generated hash key formed by concatenating all the keys from the Hash Map [11]. If both the hash keys are equal, then continue the process else terminate the process saying incorrect Hash Map.

If a user were using a different Hash Map other than the one that we used in the encoding process, then the decoding process will fail to say incorrect Hash Map. So, this increased the security to the next level.

If both the Hash keys are matched, then we replace each value in the stream with its corresponding key from the Hash Map. Figure (l) describes the above process.

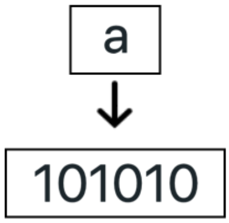


Figure (l)

After replacing all the values with its keys from the hash map, we need to divide the stream into equal parts of size four as shown in the Figure (m)

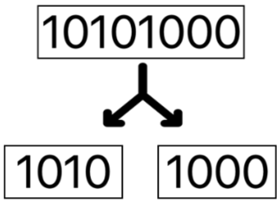


Figure (m)

Now we can concatenate any random 4 bits, or we can use the XOR value that we calculated during the Encoding process to a stream of bits. Divide the stream of bits into a byte array [12].

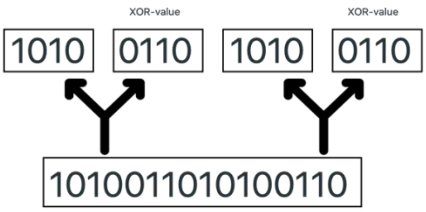


Figure (m)

Above figure(m) shows the process of generating the stream of bits by concatenating with XOR-value or any random 4-bits. We need to generate a byte array from the above stream of bits.

Convert the stream of bits into pixels by dividing the stream into blocks. Figure(o) describes the above.

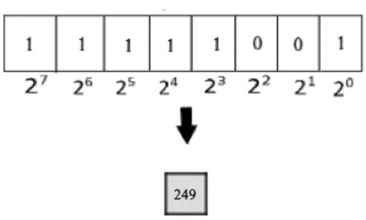


Figure (o)

Generate Image from a sequence of pixels:

After generating the byte array, we need to arrange the byte array into a two-dimensional array to form an image, as shown in Figure (p).

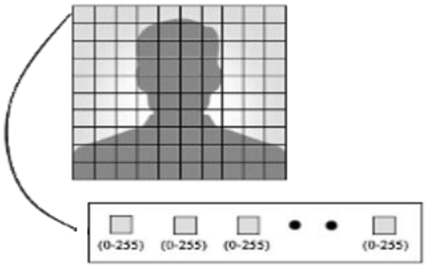
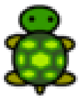


Figure (p)

3. Results



(a)

```
11101001001010001100011100000111001011000010
00000000010000000000001011001100001000000000
00000000000000000001000000000000000000000000
00000000000000010000000100000000000000000000
01001000111110100111101100111111111000101011
11010111101101011000111101011110101110000010
01111100111010010000101111010000111000111100
00101000100000010000011000010001111001111110
00011010110101100000000001010111011010000001
00111001010001001100111001010101000011010000
01000001100011000001110000000000001100100000
00000000100111111110001000011001001010010011
10000100100111000100001111100101011111011001
00000100100011000110001100100100100100111100
11110001010110010001000010110001001011000011
010000111010110101111011101101111110111110000
10100111111010111010011010010100001100000011
00010010010100011100001001000010110100111011
01110100000000010000010111110101110101001011
01001110110101010100011001000000010110111100
10001100100111011101101001001101011000000100
111011110111001100010000110111111100001010110
10110011010011101110000001001111010000010101
01101001011001010101110110111001111000110110
10011110110100110111011000010110101110000100
00011001101101111011010110101011010000001100
01010011100100010111100111010011101100111100
00110110101101111100010101000110001001011001
00010010000011110001000011101111010100101000
00101101000110100100101100110011000001110000
11001100111000100101101111110110100000000101
00010010010100011100001001000010110100111011
01110100000000010000010111110101110101001011
01001110110101010100011001000000010110111100
10001100100111011101101001001101011000000100
111011110111001100010000110111111100001010110
10110011010011101110000001001111010000010101
01101001011001010101110110111001111000110110
10011110110100110111011000010110101110000100
00011001101101111011010110101011010000001100
00010000010101011110000110011010000111000101
01101011000100011111001111101111110100100101
01100110100111110010001001010111001100111010
01001000111110100111101100111111111000101011
11010111101101011000111101011110101110000010
01111100111010010000101111010000111000111100
00101000100000010000011000010001111001111110
00011010110101100000000001010111011010000001
00111001010001001100111001010101000011010000
01000001100011000001110000000000001100100000
```

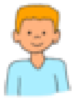
(b)

[0000;s';.q';.000ofjlidsjfidsfisjfjsofpasjfoasjfopdsofdaspo[s
Dfosadfp00000000000H8Y40000^^h%000056TtTsCG0000
00000000000000DtTtTtTs400000DtTtTtTtTX00fdol;agrnj
g0000001TtCtTtTp9c0000dfgfdg000005dTtTtTtTtG00000;'
.a';csd0000003TtTtTtTtOG0000dfgsfdgsdfg0000cTtTtTtKG
000000000000000000548H4ZC0000000000000H4H00000
00194H4H3400000000000000002Hb42HKH4H4HJ00000b';;
m';m.j00005YOCKPtPKH5PdPI5LCG000N40000000Ardcf..]

(c)

(d)

Figure(q) - (a) color image (b) before and (c) after compression (d) QR Code



(a)

10110011010011101110000001001111010000010101
01101001011001010101110110111001111000110110
10011110110100110111011000010110101110000100
00011001101101111011010110101011010000001100
01010011100100010111100111010011101100111100
00110110101101111100010101000110001001011001
00010010000011110001000011101111010100101000
00101101000110100100101100110011000001110000
11001100111000100101101111101101000000000101
00010010010100011100001001000010110100111011
01110100000000010000010111110101110101001011
01001110110101010100011001000000010110111100
10001100100111011101101001001101011000000100
11101111011100110001000011011111100001010110
10110011010011101110000001001111010000010101
01101001011001010101110110111001111000110110
10011110110100110111011000010110101110000100
00011001101101111011010110101011010000001100
00010000010101011110000110011010000111000101
01101011000100011111001111101111110100100101
01100110100111110010001001010111001100111010
01001000111110100111101100111111111000101011
11010111101101011000111101011110101110000010
01111100111010010000101111010000111000111100
00101000100000010000011000010001111001111110
00011010110101100000000000101011101101000001
00111001010001001100111001010101000011010000
01000001100011000001110000000000000110010000
11001100111000100101101111101101000000000101
00010010010100011100001001000010110100111011
01110100000000010000010111110101110101001011
01001110110101010100011001000000010110111100
10001100100111011101101001001101011000000100
11101111011100110001000011011111100001010110
10110011010011101110000001001111010000010101
01101001011001010101110110111001111000110110
10011110110100110111011000010110101110000100
00011001101101111011010110101011010000001100
00010000010101011110000110011010000111000101
01101011000100011111001111101111110100100101
01100110100111110010001001010111001100111010
01001000111110100111101100111111111000101011
11010111101101011000111101011110101110000010
01111100111010010000101111010000111000111100
00101000100000010000011000010001111001111110
00011010110101100000000000101011101101000001
00111001010001001100111001010101000011010000
01000001100011000001110000000000000110010000
00110110101101111100010101000110001001011001
00010010000011110001000011101111010100101000
00101101000110100100101100110011000001110000
11001100111000100101101111101101000000000101
00010010010100011100001001000010110100111011
01110100000000010000010111110101110101001011
01001110110101010100011001000000010110111100
10001100100111011101101001001101011000000100
11101111011100110001000011011111100001010110

(b)

[G0000001DMU9Y9Y9TG80WUhkxkxkxkwOc0Qxkxkxkxkxk
uC0022Bkxkxkxkxkxk2C0GwkxkxkxkxkxkwD03bgxkgg9kwg
gkxf500QxbuYAc8cOYAkZG01hgCpCpCpCpBUv8m07kQpC
pCpCpCoQW200UulCpCpCpCpAc0401hdCpCpCpCpCk7rcip
BpCpClCotGW08fxpBISpCbBpBQM00dApCoRpBovpCpCS0
2SpCpCouYSpCpCjG07pCpCpClCpCpAkZOBE7dCpCpCpCpC
UN0G011ipCgRkepCaa5W002JhCpAXulCam0001ApCpCpB
SW00000DwpCouG00000000DeXuO0000000008kxo…]

(c)

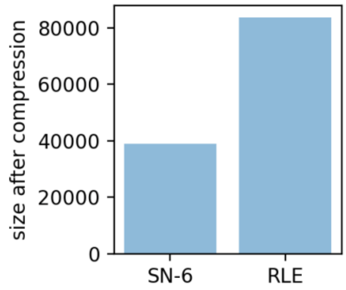
(d)

Figure(r) - (a) color image (b) before and (c) after compression (d) QR Code

IV. COMPARISON

This section presents the performance analysis results comparing with RLE [10] algorithm by applying these techniques on various test images.

Test case – 1:



Figure(s)

Test case - 2

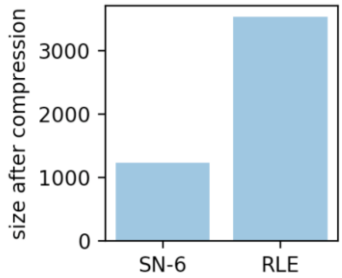


Figure (t)

V. CONCLUSION

In this paper, we have designed and developed a method that enhances the capacity of the QR code. We have developed and implemented a software code that compresses an image file which is to be stored in into the QR code in a more secure way. Doing so, we were able to expand the capacity of the QR code and store data securely.

REFERENCES

[1] M. S. Ahamed and H. Asiful Mustafa, "A Secure QR Code System for Sharing Personal Confidential Information," 2019 International Conference on Computer, Communication, Chemical, Materials and Electronic Engineering (IC4ME2), Rajshahi, Bangladesh, 2019, pp. 1-4x.

[2] Y. Wang, C. Sun, P. Kuan, C. Lu and H. Wang, "Secured graphic QR code with infrared watermark," 2018 IEEE International Conference on Applied System Invention (ICASI), Chiba, Japan, 2018, pp. 690-693.

[3] L. F. Freitas, A. R. Nogueira and M. E. V. Melgar, "Data Validation System Using QR Code and Meaningless Reversible Degradation," 2019 International Conference on Applied Electronics (AE), Pilsen, Czech Republic, 2019, pp. 1-4.

[4] Stricker, Markus Andreas, and Markus Orengo. "Similarity of color images." Storage and retrieval for image and video databases III. Vol. 2420. International Society for Optics and Photonics, 1995.

[5] Rabbani, Majid, and Paul W. Jones. Digital image compression techniques. Vol. 7. SPIE press, 1991.

[6] Al-Shaykh, Osama K., and Russell M. Mersereau. "Lossy compression of noisy images." IEEE Transactions on Image Processing 7.12 (1998): 1641-1652.

[7] Piella, Gemma. "A general framework for multiresolution image fusion: from pixels to regions." Information fusion 4.4 (2003): 259-280.

[8] Moore, Keith. MIME (multipurpose internet mail extensions) part three: Message header extensions for Non-ASCII text. RFC 2047, November, 1996.

[9] Lamatz, Nico. "LamatzSolver-v0. 1: A grounded extension finder based on the Java-Collection-Framework." System Descriptions of the First International Competition on Computational Models of Argumentation (ICCMA'15) (2015): 29-32.

[10] Abdmouleh, Med Karim, Atef Masmoudi, and Med Salim Bouhlel. "A new method which combines arithmetic coding with RLE for lossless image compression." (2012).

[11] Srinivasa Rao Narisetty, Shaik Farzana, Potnuri Maheswari, "L-Semi-Supervised Clustering for Network Intrusion Detection" International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 –

8958, Volume-8, Issue-3S, February 2019
[12] N. Srinivasa Rao, J. Pavan Kumar Reddy, "An Application for Secured Deduplication of Multimedia Data in Cloud" Jour of Adv Research in Dynamical & Control Systems, Vol. 12, Issue-02, 2020