# A Brief Overview of Byte Digraphs

James Oswald

August 18, 2020

## 1 Introduction

The work of Sergey Bratus and Greg Conti in Digraph data visualization for binary sequences is under-explored mathematically and is largely unformalized. This has lead to it being largely forgotten outside of a few open source cyber-security projects. I hope to introduce digraph matrices and provide a few formal definitions for working with them.

## 2 Binary Digraph Matrices (BDM)

### 2.1 Formalization of The Core Concept

Bratus *et al.*[1] posit that even binary data we expect to be unstructured or random is in fact structured in a way we wouldn't expect. The key idea is that each byte $b_k$ in a byte sequence $B$ of length $n$ has an implicit binary relation with the byte following it , $b_{k+1}$. We can define this relation in set builder notation as:

$$R = \{(b_k, b_{k+1}) | b_k \in B \wedge k < n\} \tag{2.1}$$

We can then graph this relation to image coined a "Digraph Bitmap", which is simply a plot of $R$ in $\mathbb{N}^2$ which gives rise to noticeable artifacts that can be used to classify data. Alternatively, I propose a digraph can be represented using a binary matrix $M_{256\times256}$ who's entries are set according to:

$$M_{ij} = \begin{cases} 1 & \text{if } (b_i, b_j) \in B \\ 0 & \text{if } (b_i, b_j) \notin B \end{cases} \tag{2.2}$$

Since these digraphs are represented using a binary matrix, I will be referring to them as "binary digraph matrices" or "BDM"s for short.

## 2.2 Formalizing a BDM Generator Algorithm

The BDM Generator Algorithm I pose here uses the process of constructing the binary matrix described by (2.2).

**Algorithm 1:** Generate Binary Digraph

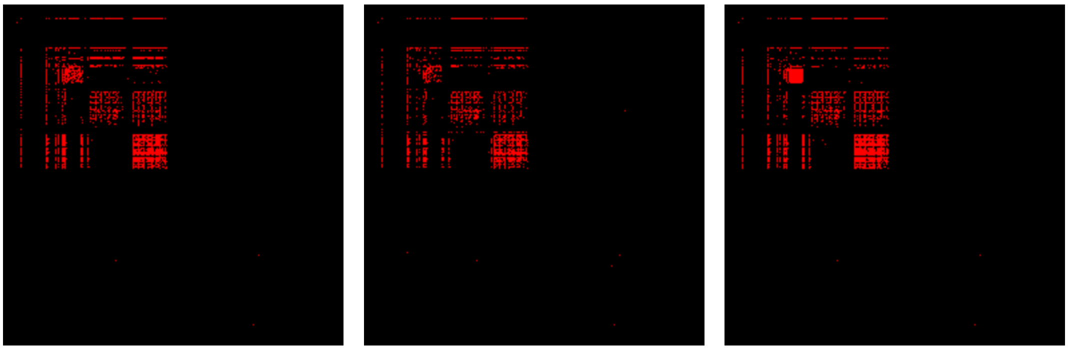
```
input  : A byte sequence $B = \{b_0, b_1, ..., b_n\}$
output: A Binary Digraph in the form of the binary matrix $M_{256\times256}$
$M = \mathbf{0}_{256\times256}$
for $i = 0$ to $n - 1$ do
    $M_{b_i b_{i+1}} = 1$
```

It is trivially proved that this algorithm is $\mathcal{O}(n)$

## 2.3 Visualizing BDMs

By taking our binary matrix $M$ and converting it to a bitmap such that 0 is black and 1 is red, we can instantly see shocking artifacts that allow us to distinguish the type of data we're looking at.

For these first three BDMs, I have converted three full books into byte strings and applied algorithm 1 generating the following bitmaps:



(a) Digraph of The Republic (b) Digraph of Alice in Wonderland (c) Digraph of The Bible

Figure 1: Book Digraphs

The obvious question that comes to mind when viewing these BDMs is: Why do all 3 of these completely different books have very similar digraphs? The answer lies in ASCII encoding: these patterns are the result of the relationship between subsequent ASCII characters. This can be broken down if we zoom in and begin classifying regions based on the sequence of bytes (or in our case, characters) that gave rise to them.

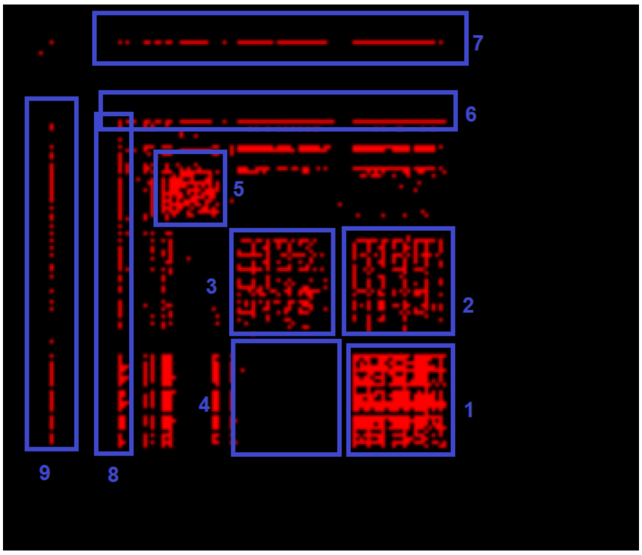


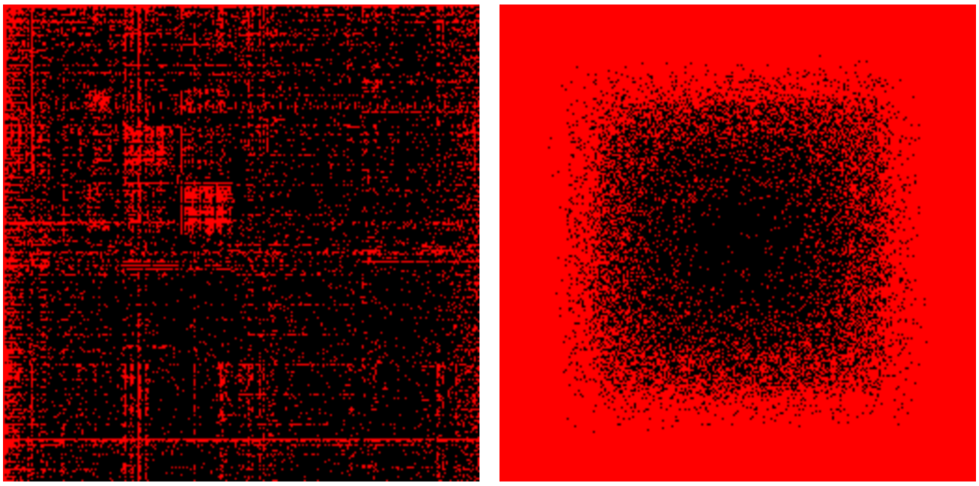
Figure 2: Digraph of The Republic

- 1)(lowercase, lowercase)
- 2)(uppercase, lowercase)
- 3)(uppercase, uppercase)
- 4)(lowercase, uppercase)
- 5)(digit, digit)
- 6)(character, space)
- 7)(character, new Line)
- 8)(space, character)
- 9)(new Line, character)

Notice that this produces results that are intuitively easy to understand as well; we would expect an English book to be primarily composed of mostly lowercase letters followed by lowercase letters, while we would expect no lowercase letters followed by uppercase letters, which is exactly what we see in figure 2.

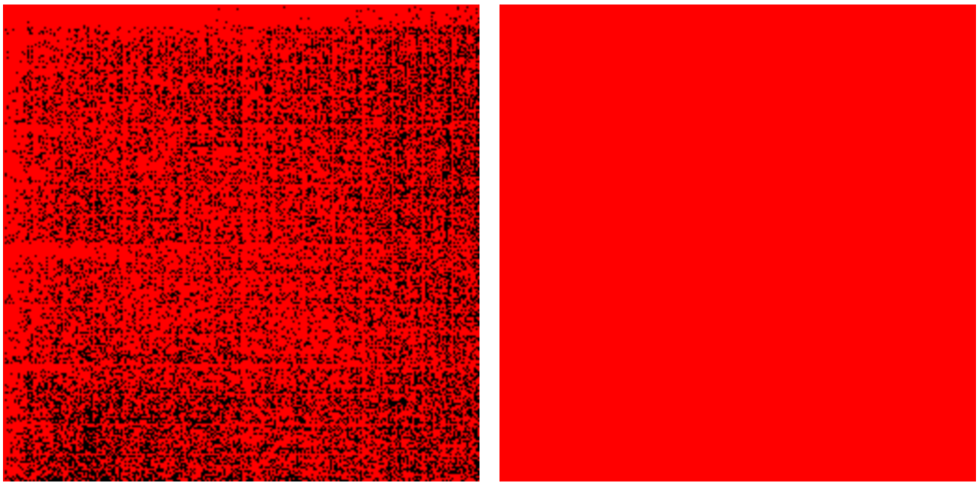
## 2.4 Limitations of BDMs

In the last section we looked at BDMs of text data, in this section we'll explore BDMs of more types of data and see the limitations of BDMs that arise from large amounts of data.



(a) Digraph of a .exe (Machine Code Data) (b) Digraph of a .wav (Raw Audio Data)

Figure 3: Sparse



(a) Digraph of a .gif (Compressed Image Data)

(b) Digraph of a .webm (Compressed Video Data)

Figure 4: Dense

These binary digraphs give us much insight into the intrinsic structure of this data and let us analyze entire files from just these snapshots. An analysis of 3a shows characteristic striations which are present in digraphs of all kinds of machine code [2]; we also see the ASCII regions are extra dense, so we can infer that there are a many embedded strings within this exe file. An analysis of 3b reveals yet another striking pattern produced by the digraph, characteristic of uncompressed audio data.

Figures 4a and 4b show where the usefulness of binary digraphs begin to run out; while 4a still shows a semblance of structure characteristic to compressed image data, the digraph in 4b is so dense that it provides us with no meaningful structure, only letting us know that the data is compressed.

# 3 Positive Discrete Digraph Matrices (PDDMs)

## 3.1 Surpassing BDMs with PDDMs

In order to remedy this issue, I have developed what I will refer to as positive discrete digraph matrices (PDDMs) to preserve more information and hence allow for better visualization. Rather then entires of the matrix $M$ being restricted to the set $\{0, 1\}$ as was the case with BDMs, I restrict entries to the non-negative integers $\mathbb{Z}^*$. The entries in the new matrix now represent the "count" of a sequential pair in $B$. In other words $M_{ij}$ is now the number of times $(b_i, b_j)$ appears in $R$.

## 3.2 PDDM Generator Algorithm

The algorithm is fundamentally the same as the digraph algorithm but we add one to the entry instead of setting it to one.

**Algorithm 2:** Generate Positive Discrete Digraph Matrix

```
input  : A byte sequence $B = \{b_0, b_1, ..., b_n\}$
output: A PDDM in the form $M_{256\times256}$
$M = \mathbf{0}_{256\times256}$
for $i = 0$ to $n - 1$ do
    $M_{b_i b_{i+1}} = M_{b_i b_{i+1}} + 1$
```

It is trivially proved that this algorithm is $\mathcal{O}(n)$

## 3.3 Visualizing binary data with PDDMs

### 3.3.1 Post-Processing Algorithm

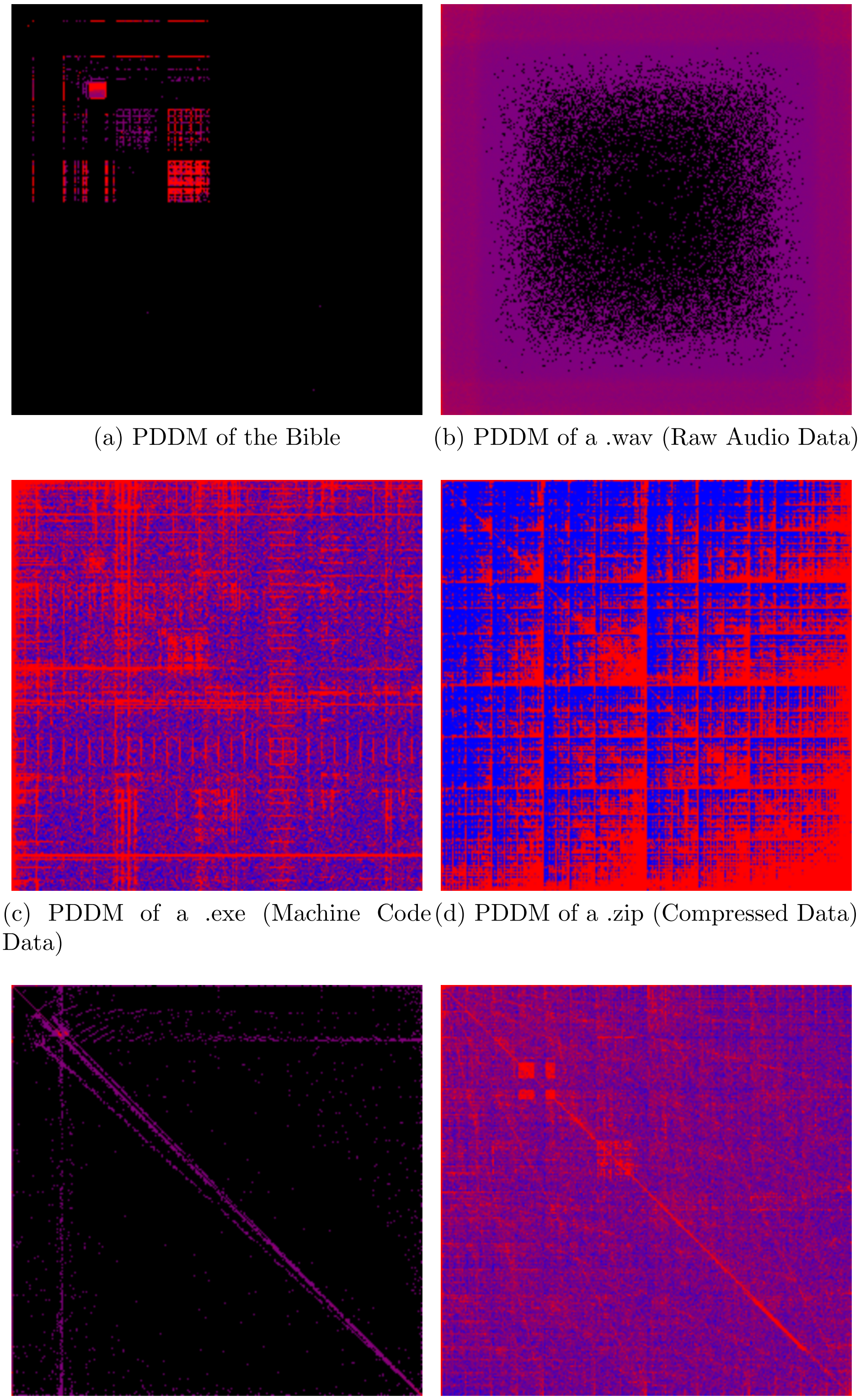
Unfortunately, coloring a bitmap solely from a PDDM alone produces lackluster results and doesn't show off the full potential of the information stored within the the positive discrete digraph. To get around this, I apply a post-processing algorithm that uses the median of the vectorization of the PDDM to help "center" the data for better viability. I then use this "centered" data to generate a bitmap, MAP, which for the purpose of this algorithm, will be a 256 by 256 matrix of 3-tuples representing RGB color channels who's values are integers between 0 and 255.

**Algorithm 3:** Positive Discrete Digraph Matrix Post-Processing

```
input  : A PDDM, $M$ in the form of a 256 by 256 matrix
output: A Bitmap MAP in the form of a matrix of 3-tuples
        representing RGB Color channels
Med = Median(Vectorize($M$))
MAP = $\begin{bmatrix} (0,0,0)_{0,0} & \dots & (0,0,0)_{255,0} \\ \vdots & \ddots & \vdots \\ (0,0,0)_{0,255} & \dots & (0,0,0)_{255,255} \end{bmatrix}$
for $i = 0$ to 255 do
    for $j = 0$ to 255 do
        $\text{MAP}_{ij} = (M_{ij} - \text{Med} + 127, 0, -M_{ij} + \text{Med} + 127)$
```

### 3.3.2 Visual PDDM Results

(See Next Page)



(a) PDDM of the Bible

(b) PDDM of a .wav (Raw Audio Data)

(c) PDDM of a .exe (Machine Code Data)

(d) PDDM of a .zip (Compressed Data)

(e) PDDM of a .bmp (Raw Image Data)

(f) PDDM of an .msi (windows installer)

Figure 5: ”PDDM Visual Results”

### 3.3.3 Analysis of PDD Results

Under the new light provided by PDDM with post processing, we can see structure that was previously hidden. In fact, 5c, 5d, 5f, would all look the same when seen from a binary digraph bitmap, but using a PDDM we can see that all three actually have wildly different structures.

Upon re-analyzing 5a we see exactly what we expected from the ascii data, the red coloration in the (lowercase, lowercase) region tells us that all of these sequences occurred well above the median, while other regions were closer to the median and thus appear purple.

Re-analyzing 5b we see a bit more structure appear around the edges, however its easy to forget that the lack of real definition in the form of reds and blues is a sign that all central sequences appear at the same relative frequency, which itself is an important structural insight gained through a PDDM.

5c is a large x86 machine code file (a windows .exe) that contains the telltale machine code striations discussed in section 2.4. Its also good to note that none of these would have been visible on a binary digraph and this would have looked identical to the .gif file in image 4b.

5d is a 300MB zip file; it makes sense that every spot would be filled with that many bytes, but it has a surprising structure that, much like 5c, would not have been seen without the PDDM.

5e is a bitmap file storing raw image data, the diagonal stripe in the center is indicative of subsequent bytes having a similar value. [2]

5f is perhaps the most interesting PDDM yet. It is a Windows Installer file, and it looks like a combination of 5a, 5c, 5d, 5e– and that's because it is. We can infer from the digraph that it contains many bitmaps from the red streak down the center, a lot of ascii data from the squares in the top right, and machine code from the heavy parallel striations.

# References

[1] S. Bratus and G. Conti. Voyage of the reverser, a visual study of binary species. blackhat, 2010.

[2] C. Domas. The future of re dynamic binary visualization. Defcon, 2012.