HOCHSCHULE
MITTWEIDA
University of Applied Sciences

MASTERARBEIT

Frau
Susanna Juliane Beck

Streambasierte Analyse und Klassifizierung von Datenstrukturen

Mittweida, Oktober 2023

Fakultät **Angewandte Computer- und Biowissenschaften**

MASTERARBEIT

Streambasierte Analyse und Klassifizierung von Datenstrukturen

Autorin:
Susanna Juliane Beck

Studiengang:
Cybercrime/Cybersecurity

Seminargruppe:
CY19wC-M

Erstprüfer:
Prof. Ronny Bodach

Zweitprüfer:
Dipl.-Ing. Stefan Hoheisel

Einreichung:
Mittweida, 17.10.2023

Verteidigung/Bewertung:
Mittweida, 2023

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Stream-Based Analysis and Classification of Data Structures

Author:
Susanna Juliane Beck

Course of Study:
Cybercrime/Cybersecurity

Seminar Group:
CY19wC-M

First Examiner:
Prof. Ronny Bodach

Second Examiner:
Dipl.-Ing. Stefan Hoheisel

Submission:
Mittweida, 17.10.2023

Defense/Evaluation:
Mittweida, 2023

Bibliografische Beschreibung:

Beck, Susanna Juliane:
Streambasierte Analyse und Klassifizierung von Datenstrukturen. – 2023. –
161 Seiten, 69 Abbildungen, 8 Tabellen.
Mittweida, Hochschule Mittweida – University of Applied Sciences,
Fakultät Angewandte Computer- und Biowissenschaften, Masterarbeit, 2023.

Referat:

Im Reverse Engineering und in der Malware-Analyse wurden bereits verschiedene Ansätze zur Visualisierung von Binärdaten entwickelt. Mit diesen lässt sich schnell ein Überblick über Dateien gewinnen, sodass beispielsweise verschiedene Regionen einer Datei identifiziert oder eine bösartige Datei einer Malware-Familie zugeordnet werden kann. In der vorliegenden Masterarbeit wird versucht, diese Ansätze auch sektorweise auf einen Datenstream anzuwenden. Dafür wird ein Demonstrator erstellt, mit dem Sektoren automatisiert nach Dateitypen klassifiziert werden können. Ziel ist es, einen Ansatz zur Verbesserung der aktuellen, signaturbasierten IT-forensischen Methoden zur Wiederherstellung von fragmentierten oder gelöschten Daten zu finden.

Abstract:

Researchers in reverse engineering and malware analysis previously developed some approaches to visualize binary data. These can be useful to get a quick overview of a file and identify various regions within or to classify a malicious file into its respective malware category. The master's thesis on hand is trying to adapt these approaches such that they can be used to analyze sectors within a stream of data. Thus, a demonstrator that can be used to classify sectors into file types is implemented. The goal is to find an approach that can improve the signature-based methods currently used in IT forensics to recover deleted or fragmented data.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Abkürzungsverzeichnis

apk . . . . . Application Package (Dateiformat für Android-Applikationen)

ASCII . . . . . American Standard Code for Information Interchange

Blob . . . . . Binary Large Object

bmp . . . . . Bitmap (Dateiformat für Bilddateien)

BSI . . . . . Bundesamt für Sicherheit in der Informationstechnik

cc . . . . . correct class

docx . . . . . Office Open XML Document (Dateiformat für Microsoft-Word-Dokumente) [1]

elf . . . . . Executable and Linking Format (Dateiformat für Linux-Applikationen und zugehörige -Bibliotheken)

epub . . . . . Electronic Publication (Dateiformat für E-Books)

exe . . . . . Executable (Dateiformat für Windows-Applikationen)

FN . . . . . false negative

FP . . . . . false positive

GB . . . . . Gigabyte

gif . . . . . Graphics Interchange Format (Dateiformat für Bilddateien)

GUI . . . . . Graphische Benutzeroberfläche (engl. Graphical User Interface)

HK . . . . . Hilbertkurve

HSB . . . . . Hue - Saturation - Brightness (Farbmodell)

HSL . . . . . Hue - Saturation - Lightness (Farbmodell)

HSV . . . . . Hue - Saturation - Value (Farbmodell)

jpg . . . . . Joint Photograhic Experts Group (Dateiformat für Bilddateien)

KB . . . . . Kilobyte

KI . . . . . Künstliche Intelligenz

MB . . . . . Megabyte

ML . . . . . Machine Learning

mp4 . . . . . Moving Picture Experts Group-4 (Dateiformat für Videodateien)

nan . . . . . Not a Number

pc . . . . . predicted class

pdf . . . . . Portable Document Format (Portables Dateiformat für Dokumente)

png . . . . . Portable Network Graphic (Dateiformat für Bilddateien)

pptx . . . . . Office Open XML Presentation (Dateiformat für Microsoft-Powerpoint-Präsentationen) [1]

RE . . . . . Reverse Engineering

RGB . . . . . Red - Green - Blue (Farbmodell)

svg . . . . . Scalable Vector Graphics (Dateiformat für Bilddateien)

TB . . . . . Terabyte

TN . . . . . true negative

TP . . . . . true positive

txt . . . . . Text (Dateiformat für Textdokumente)

xDFAI . . . . . Explainable Digital Forensics Artificial Intelligence

xlsx . . . . . Office Open XML Workbook (Dateiformat für Microsoft-Excel-Tabellendokumente) [1]

xml . . . . . Extensible Markup Language (Dateiformat für Dokumente)

1 Einleitung

Ein Bild sagt mehr als tausend Worte. - Sprichwort

Dieses weithin bekannte Sprichwort lässt sich auch auf die Analyse von Binärdaten anwenden. So können mit der richtigen Darstellung auf derselben Monitorfläche 900-mal mehr Informationen dargestellt werden, wenn eine Visualisierungsmethode anstelle des klassischen Hex-Editors gewählt wird [2]. Zudem kann der Mensch Bilder und visuelle Muster deutlich schneller erfassen und verarbeiten als bespielsweise Binärdaten in hexadezimaler Darstellung [3, 4]. Die Visualisierung von Binärdaten bietet also großes Potenzial zur Auswertung von Daten. Trotzdem ist dieses Gebiet bisher nur begrenzt erforscht.

Die bereits bestehenden Ansätze zur Analyse von Daten auf Basis von Visualisierungen beschäftigen sich zumeist mit ganzen Dateien und versuchen diese zu klassifizieren. Andere Ansätze probieren bekannte Strukturen in unbekannten Dateien zu finden und so deren Analyse zu erleichtern. Auch die IT-Forensik könnte von Analyse-Methoden auf Basis dieser Visualisierungsmethoden profitieren, jedoch besteht noch Anpassungsbedarf, um eine Nutzbarkeit dieser Methoden in der Forensik zu erhöhen. Eine potenzielle Anwendungsmöglichkeit der visuellen Analyse besteht hier insbesondere im Bereich der Wiederherstellung von fragmentierten oder gelöschten Daten, welche anschließend als digitale Beweismittel für kriminalpolizeiliche Ermittlungen verwendet werden können.

1.1 Inspiration

In der Vergangenheit wurde vor allem im Bereich der Malware-Analyse und des Reverse Engineering (RE) die Analyse von Daten anhand visueller Darstellungen erforscht. Diese Methoden erlauben nicht nur die Unterscheidung von Daten in die Kategorien „nicht verschlüsselt/nicht komprimiert“ oder „verschlüsselt/komprimiert“, sondern konnten auch zur Erkennung einzelner Daten- und Malwaretypen genutzt werden. Dies deutet darauf hin, dass die verwendeten Techniken und Methoden auch im Rahmen der forensischen Datenwiederherstellung von Nutzen sein können, da über die Zuordnung von Dateitypen auch die Zugehörigkeit von Bereichen des Datenstreams zu einer Datei festgestellt werden kann. Dies würde gegebenfalls auch dann die Wiederherstellung gelöschter Daten erlauben, wenn die zugehörigen Header nicht mehr im Speicher liegen und könnte der IT-Forensik somit neue Möglichkeiten eröffnen.

1.2 Zielstellung

Basierend auf der eben gestellten These soll mit der vorliegenden Masterarbeit ein Demonstrator für die streambasierte Erkennung und Klassifizierung von Dateitypen und -strukturen erarbeitet werden. Der Fokus liegt darauf, dass auf Basis der in anderen Themengebieten bereits verwendeten visuellen Darstellung eine Methode zur Erkennung von verschiedenen Dateitypen entwickelt wird, die möglichst zuverlässig funktioniert. Idealerweise erkennt die zu entwickelnde Methodik diese Dateitypen auch, ohne eine Auswertung von vorhandenen Header- und Footer-Signaturen durchzuführen.

Dazu soll zunächst ein Überblick über bereits vorhandene Methoden gewonnen und einige ausgewählte davon praktisch umgesetzt werden. Auf Basis dieser Methoden sollen außerdem eigene Ansätze entwickelt und umgesetzt werden. Anschließend sollen alle umgesetzten Methoden bezüglich ihrer Eignung zum Klassifizieren von Daten bewertet werden. Da es sich um Visualisierungstechniken handelt, sollte diese Bewertung zunächst visuell erfolgen. So kann festgestellt werden, welche der Methoden für eine weitere Implementierung eines Klassifikators geeignet sind. Mit den ausgewählten Methoden sollte dann ein Algorithmus zur automatisierten Bewertung erarbeitet und getestet werden.

1.3 Abgrenzung

Der Großteil der bisherigen Ansätze zu Visualisierung von Binärdaten ist im Bereich des RE und der Malware-Analyse einzuordnen. Mit der vorliegenden Arbeit soll eine Methodik entwickelt werden, die insbesondere in der digitalen Forensik zur Datenanalyse und -rekonstruktion verwendet werden kann.

Während die Visualisierungstechniken in der Malware-Analyse zur Klassifikation verschiedener Malwaretypen verwendet wird, wird im RE meist versucht, einzelne, für den Betrachter interessante, Bereiche innerhalb einer Datei zu identifizieren. Dazu zählen zum Beispiel die Stellen einer Datei, die Textzeichen enthalten, da diese Strings Informationen im Klartext enthalten können. Ebenso von Interesse sind Bereiche mit hoher Entropie, da diese unter anderem kryptographische Schlüssel enthalten können. Hier liegt der Fokus also darauf, eine Datei insgesamt zu betrachten und die für die jeweils zu erledigende Aufgabe relevanten Stellen zu finden.

Mit der vorliegenden Arbeit soll die Analyse und Unterteilung eines größeren Datenfragments hinsichtlich des wahrscheinlichsten Dateityps einzelner Abschnitte in diesem Fragment erfolgen. Dies soll zum Überblick über die innerhalb des Fragments vorkommenden Daten beitragen.

1.4 Aufbau

Die weitere Arbeit gliedert sich wie folgt: In Kapitel 2 Grundlagen wird ein grundlegender Überblick über die digitale Forensik und die Notwendigkeit der Datenwiederherstellung sowie über bereits bestehende Konzepte und Methoden zur Visualisierung von Binärdaten gegeben. Außerdem werden zum Verständnis benötigte mathematische Grundlagen aufgezeigt. In Kapitel 3 Vorüberlegungen werden erste Ideen zur Umsetzung besprochen. Die darauf folgenden Kapitel 4 Versuchsaufbau und 5 Methoden beschreiben die praktische Umsetzung der Arbeit. Schließlich wird in Kapitel 6 Demonstrator der im Rahmen der vorliegenden Arbeit entwickelte Demonstrator vorgestellt, bevor in Kapitel 7 Zusammenfassung und Ausblick weitere Ideen für die Zeit nach Fertigstellung der vorliegenden Arbeit erläutert werden.

2 Grundlagen

In diesem Kapitel sollen einzelne Aspekte aus den verschiedenen, für die vorliegende Arbeit relevanten Wissensgebieten vorgestellt werden. Dabei soll eine kurze Einführung in das jeweilige Thema gegeben werden, die ein grundlegendes Verständnis des Konzeptes und damit der vorliegenden Arbeit fördert. Die Ausführungen zu den einzelnen Punkten sind in aller Regel nicht abschließend.

2.1 Mathematische Konzepte

Zunächst werden einige Ansätze aus verschiedenen Bereichen der Mathematik erläutert. Diese finden in den im Folgenden beschriebenen Grundlagen sowie im weiteren Verlauf der vorliegenden Arbeit mehrfach Verwendung. Sie sind daher essentiell für ein besseres Verständnis der beschriebenen Themen.

2.1.1 Statistische Lagemaße

Die Statistik bietet diverse Kennwerte zur Beschreibung von Daten und Verteilungen. Den vermutlich bekanntesten Kennwert stellt das arithmetische Mittel $\bar{x}$, auch Mittelwert genannt, dar [5]. Gebildet wird es, indem die Summe aller Werte gebildet und durch die Anzahl dieser geteilt wird. Mathematisch ausgedrückt wird das arithmetische Mittel als

$$\bar{x} = \frac{1}{n}(x_1 + ... + x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2-1}$$

für eine Verteilung $x_1...x_n$ definiert. Neben dem arithmetischen Mittel gibt es weitere Kennwerte, welche die zentrale Tendenz einer Verteilung anzeigen [5, 6]. Davon werden in der vorliegenden Arbeit das getrimmte Mittel und der Median vorgestellt, es existieren aber beispielsweise auch das winsorisierte Mittel und der Modus [5, 6].

Das getrimmte Mittel $\bar{x}_g$ ist ähnlich dem arithmetischen Mittel, soll aber den Einfluss von Ausreißern, also Werten die besonders stark vom Rest der Verteilung abweichen, begrenzen [5]. Dafür wird eine Datenreihe zunächst der Größe nach sortiert, sodass aus der Verteilung $x_1...x_n$ eine geordnete Urliste $x_{(1)} \leq ... \leq x_{(i)} \leq ... \leq x_{(n)}$ wird [5]. Dann können am oberen und unteren Ende der Reihe Werte anhand eines zuvor gesetzten Wertes, beispielsweise 10%, abgeschnitten, also getrimmt, werden [5]. Das getrimmte Mittel wird nun aus den verbliebenen Werten analog zum arithmetischen Mittel berechnet [5].

Auch der Median x_{med} ist Ausreißern gegenüber resistent [5]. Um ihn zu bestimmen, wird die vorliegende Datenreihe zunächst wieder der Größe nach sortiert [6]. Beim Median handelt es sich nun schlichtweg um den in der Urliste mittig angeordneten Wert [5, 6]. Ist die Anzahl Elemente in der Urliste n gerade, so wird der Median als das arithmetische Mittel der beiden in der Mitte befindlichen Werte bestimmt [5, 6]. Als an mittlerer Position in der Liste sind dabei immer genau die

Werte anzusehen, welche die gleiche Menge Werte oberhalb und unterhalb von sich haben [5, 6]. Anders ausgedrückt ist

$$x_{med} = \begin{cases} x_{\left(\frac{n+1}{2}\right)} & \text{für } n \text{ ungerade} \\ \frac{1}{2}\left(x_{(n/2)} + x_{(n/2+1)}\right) & \text{für } n \text{ gerade.} \end{cases} \tag{2-2}$$

Welches der vorgestellten Maße sinnvoll angewendet werden sollte, ist immer vom Kontext der Fragestellung sowie den zugrundeliegenden Daten abhängig [5].

2.1.2 Hilbertkurve

Die Hilbertkurve ist ein Vertreter der raumfüllenden Kurven. Das heißt, dass es sich um eine Kurve handelt, die alle Punkte in einem Quadrat ohne Unterbrechung abläuft [7].

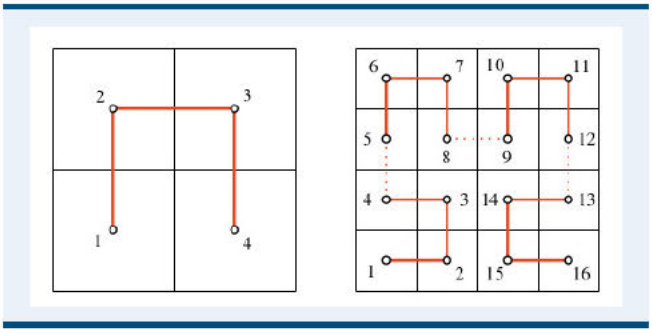


Abbildung 2-1: Konstruktion der Hilbertkurve, aus [8]

Um die Hilbertkurve besser zu verstehen, soll eine Möglichkeit der Bildung der Kurve im zweidimensionalen Raum betrachtet werden. Dafür wird das Einheitsquadrat zunächst in vier Teilquadrate unterteilt [8]. Diese werden durch die sogenannte Urkurve miteinander verbunden, wie in Abbildung 2-1 links gezeigt. Im nächsten Schritt wird jedes der Teilquadrate wiederum in vier Subquadrate aufgeteilt. Somit kann in den vier Teilquadraten aus dem vorherigen Konstruktionsschritt jeweils die Urkurve eingesetzt werden. Abhängig von der Ausrichtung der Kurve im vorhergegangenen Schritt, kann dies entweder im Original oder in rotierter bzw. gespiegelter Form erfolgen [8–10]. Abbildung 2-2 gibt eine Übersicht, welche Form der Urkurve jeweils verwendet wird.

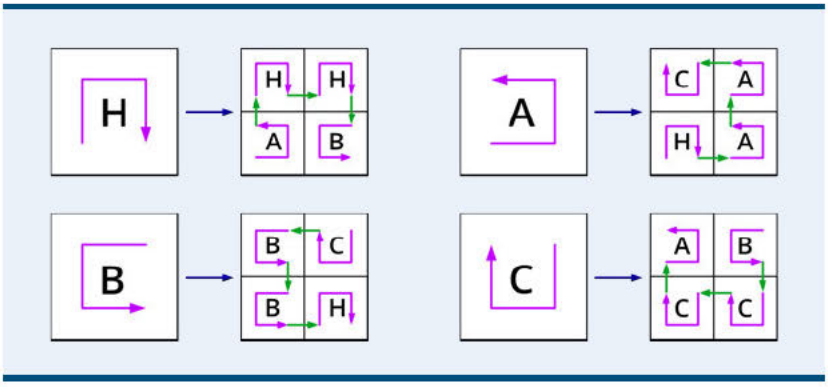


Abbildung 2-2: Verfeinerungsschritte nach Hilbert, nach [10]

Die Hilbertkurve kann beliebig viele Iterationen durchlaufen und wird dabei immer feiner, das heißt mit jeder Iteration können mehr Punkte verbunden werden. Dies sei beispielhaft in Abbildung 2-3 gezeigt, welche die Hilbertkurve der zweiten, dritten, vierten und fünften Ordnung graphisch darstellt. Konkret verbindet die Hilbertkurve der n-ten Ordnung 4^n Teilquadrate [7]. Der Begriff der n-ten Ordnung wird dabei analog zur n-ten Iteration verwendet.

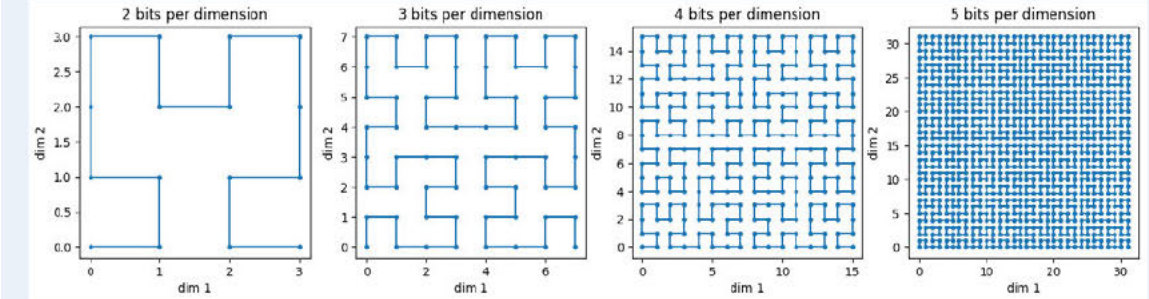


Abbildung 2-3: Beispiele für zweidimensionale Hilbertkurve, aus [11]

Auch wenn sich Hilberts ursprüngliche Beschreibung der nach ihm benannten Kurve auf den zweidimenisonalen Raum beschränkt, so lässt sie sich auch in mehrdimensionalen Räumen darstellen und bilden [9, 11]. Zur Anschauung soll hier Abbildung 2-4 dienen, welche die Darstellung der Hilbertkurve der ersten bis vierten Ordnung im dreidimensionalen Raum enthält.

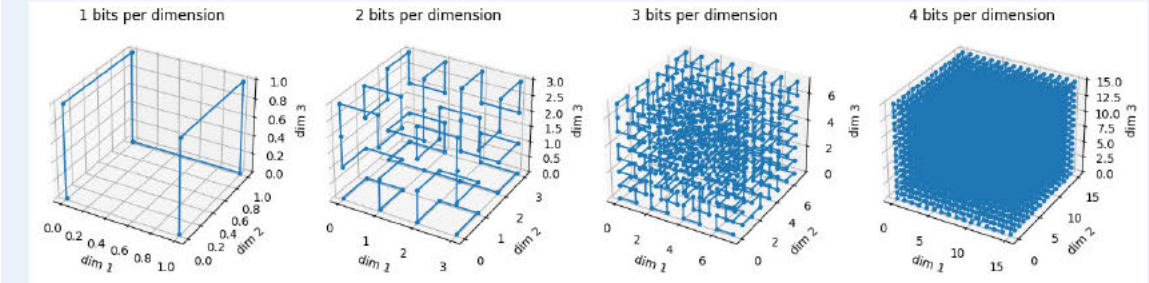


Abbildung 2-4: Beispiele für dreidimensionale Hilbertkurve, aus [11]

Die durch die Hilbertkurve verbundenen Punkte im n-dimensionalen Raum lassen sich gemäß der Definition der Kurve auf ein eindimensionales Intervall abbilden [8, 10], wie in Abbildung 2-5 gezeigt. Dies lässt sich nutzen, um Punkte oder Daten aus einem n-dimensionalen Raum in ein eindimensionales Format zu überführen. Die entstehende Datenreihe behält sogar teilweise die Lokalität der Daten bei. So sind sich Punkte, die in der eindimensionalen Darstellung nah beieinander liegen, auch im Zweidimensionalen tendenziell nah [9]. Auch umgekehrt sind viele der im Raum beieinander liegenden Punkte in der Datenreihe nicht weit voneinander entfernt, allerdings gibt es auch immer Punkte, die im Eindimensionalen weiter auseinander liegen als im mehrdimensionalen Raum [9, 12]. Ein Beispiel dafür sind die Punkte/Quadrate 2 und 15 bzw. 3 und 14 in Abbildung 2-1 rechts, welche in der Darstellung direkt nebeneinander liegen, auf der Strecke aber - erkennbar an der Nummerierung - nicht. Trotzdem sie nicht vollständig erhalten bleibt, ist die Lokalität ausreichend, damit die Hilbertkurve auch bei Clustering-Aufgaben gut geeignet ist [12]

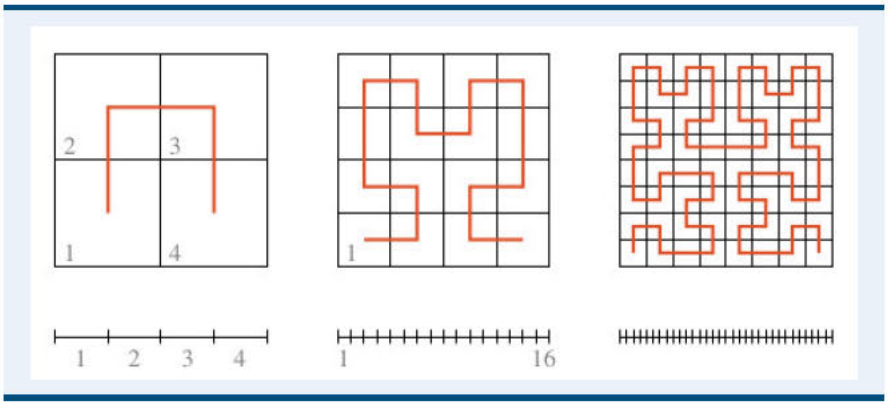


Abbildung 2-5: Verfeinerungsschritte nach Hilbert, aus [10]

2.1.3 Shannon-Entropie

Während die Shannon-Entropie ursprünglich dazu gedacht war, den Informationsverlust bei der Übertragung einer Nachricht zu messen, wird sie heute ganz allgemein dazu verwendet, die Unregelmäßigkeit einer Verteilung p zu berechnen [13].

In der Informationstheorie wird die Shannon-Entropie dazu verwendet, den Informationsgehalt einer Nachricht zu messen [14]. Dabei wird davon ausgegangen, dass eine selten gesendete Nachricht einen höheren Informationsgehalt hat als eine häufig gesendete Nachricht. Nach Brownlee [14] lässt sich dies so erklären, dass eine seltene Nachricht überraschender auf den Sender wirkt als eine regelmäßig gesendete Nachricht.

$$h(x) = log_2 \frac{1}{P(x)} \tag{2-3}$$

Die Definition des Informationsgehalts und damit der Shannon-Entropie ist in Gleichung 2-3 gegeben. Dabei gibt das Ergebnis $h(x)$ an, wie viele Bits benötigt werden um die Information x wiederzugeben. Wird statt log_2 der natürliche Logarithmus ln, also der log_e der eulerschen Zahl e, berechnet, so wird das Ergebnis in sogenannten Nats angegeben [14].

Die Entropie von Daten ist in den Bereichen der IT-Forensik und des Reverse Engineering unter anderem deshalb relevant, weil sie Aufschluss über die Art der Daten geben kann. Insbesondere verschlüsselte oder komprimierte Daten weisen in der Regel eine hohe Entropie auf [15]. Auch die zugehörigen Schlüssel, Zertifikate und Hashes sind meist an einer hohen Entropie erkennbar. [15]

2.1.4 Naive-Bayes-Klassifikator

Als Teilbereich des Machine Learning (ML) werden Klassifikationsmodelle dann angewendet, wenn einem Datensatz X ein Klassenlabel y zugeordnet werden soll [16]. Das heißt, der Klassifikator soll $y = f(X)$ bestimmen. In der einfachen Form eines binären Klassifikators sind diese Klassenlabel häufig mit „positive“ und „negative“ oder mit „true“ und „false“ benannt. Der Datensatz besteht aus einem oder mehreren Merkmalen, auch Features genannt. Im Folgenden sind die Klassenlabel der Klassen 0 bis k mit $y_0, ..., y_i, ..., y_k$ gekennzeichnet, die Merkmale 0 bis n mit $x_0, ..., x_j, ..., x_n$.

Der Naive-Bayes-Klassifikator ist ein Beispiel eines solchen Klassifikationsmodells, welches unter anderem in den Bereichen der Textklassifizierung und des Spam-Filterns eingesetzt wird [17]. Dieser Klassifikator berechnet auf Basis der gegebenen Merkmale $x_0, ..., x_i, ..., x_n$ für jede Klasse y_i die Wahrscheinlichkeit, dass es sich um ein Element der Klasse handelt [16, 17]. Diese Wahrscheinlichkeit wird auch als $P(y_i|x_0, ..., x_n)$ bezeichnet. Anhand dieser Werte wird der Klassifikator einen Datensatz immer der Klasse y_i mit dem höchsten Wert für $P(y_i|x_0, ..., x_n)$ zuordnen [16–18].

Entsprechend müssen die Klassenwahrscheinlichkeiten bereits vor dem eigentlichen Klassifizierungsprozess feststehen. Dafür werden sogenannte Trainingsdaten verwendet, anhand derer der Klassifikator „erlernt“, welche Merkmale die Klassenzugehörigkeiten wie beeinflussen [16]. Dieses Vorgehen, bei dem ein ML-Modell anhand von kuratierten, bereits gelabelten Datensätzen trainiert wird, bezeichnet man auch als Supervised Learning [16, 17, 19]. Es ist damit das Gegenteil des Unsupervised Learning, bei dem ein ML-Modell nur die Datensätze erhält und mögliche Klassenlabel selbst bestimmen muss [19]. Als Zwischenform gibt es außerdem das Semi-Supervised Learning, bei dem die Trainingsdaten teilweise mit und teilweise ohne Label bereitgestellt werden [19].

Um die Klassenwahrscheinlichkeiten $P(y_i|x_0, ..., x_n)$ möglichst einfach zu berechnen wird das Bayes-Theorem verwendet [16, 17]. Diese in 2-4 dargestellte Formel ermöglicht die Berechnung dieser Wahrscheinlichkeiten anhand von Werten, die aus den Traingsdaten relativ einfach abzulesen oder berechnen sind [17].

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2-4}$$

Angepasst auf den Klassifikator, lässt sich die Wahrscheinlichkeit $P(y_i|x_0, ..., x_n)$ mit dem Bayes-Theorem berechnen, wie in Gleichung 2-5 gezeigt [16, 18].

$$P(y_i|x_0, ..., x_n) = \frac{P(x_0, ..., x_n|y_i) \cdot P(y_i)}{P(x_0, ..., x_n)} \tag{2-5}$$

Die in Gleichung 2-5 gezeigten Wahrscheinlichkeiten lassen sich in „a-priori“ und „a-posteriori“ unterteilen. Der Begriff „a-priori“ wird dabei verwendet, um von vorneherein bekannte Wahrscheinlichkeiten zu kennzeichnen, während „a-posteriori“-Wahrscheinlichkeiten die Zusammenhänge von Klassen und Merkmalen widerspiegeln [18]. Im gegebenen Beispiel sind also die relativen Klassen- und Merkmalshäufigkeiten $P(y_i)$ und $P(x_0, ..., x_n)$ a-priori. Die Wahrscheinlichkeit, dass es sich bei einem Datensatz mit den Merkmalen $x_0, ..., x_n$ um Klasse y_i handelt ($P(y_i|x_0, ..., x_n)$) und die Wahrscheinlichkeit, dass ein Element der Klasse y_i die Merkmale $x_0, ..., x_n$ annimmt ($P(x_0, ..., x_n|y_i)$), sind dagegen a-priori-Wahrscheinlichkeiten [17].

Das Bayes-Theorem basiert auf der Annahme, dass die verschiedenen Features $x_0, ..., x_n$ voneinander abhängig sind [16, 17]. Insbesondere bei einer großen Menge Trainingsdaten oder auch bei einer Vielzahl von zu beachtenden x_j führt dies dazu, dass die Berechnung der Wahrscheinlichkeiten sehr komplex und rechenaufwändig wird [16–18]. Daher wird beim Naive-Bayes-Klassifikator zusätzlich die „naive“ Annahme getroffen, dass die Merkmale voneinander unabhängig sind [16–18].

Auch wenn diese Annahme in Realität meist nicht zutrifft, sorgt diese Vereinfachung nicht zwingend für eine Verschlechterung der Klassifikationsergebnisse [16, 18]. Gleichzeitig verringert diese Annahme den Berechnungsaufwand jedoch drastisch, da bei mehreren Merkmalen die a-posteri-Wahrscheinlichkeiten für jedes Merkmal einzeln berechnet und dann multipliziert werden können, wie in Gleichung 2-6 ersichtlich [16–18].

$$P(y_i|x_0, ..., x_n) = P(y_i|x_0) \cdot ... \cdot P(y_i|x_n) \tag{2-6}$$

Darüber hinaus kann der Nenner im Bruch in Gleichung 2-5 vernachlässigt werden, da $P(x_0, ..., x_n)$ für jede Klasse y_i gleich ist [16, 18]. Zusammengefasst lässt sich $y = f(X)$ für einen Naive-Bayes-Klassifikator also folgendermaßen darstellen:

$$y = \underset{y_i}{\operatorname{argmax}}\{P(y_i) \cdot \prod_{j=0}^{n} P(x_j|y_i)\} \tag{2-7}$$

Begünstigt durch die getroffenen Vereinfachungen ist der hier vorgestellte Algorithmus für Naive-Bayes-Klassifikatoren selbst bei großen Datenmengen insgesamt recht schnell und genau [17]. Gleichzeitig können jedoch immer nur solche Datensätze korrekt klassifiziert werden, die auch mit mindestens einem Tupel in den Trainingsdaten enthalten sind. Eine Kombination aus Klasse und Merkmalen, für die nicht wenigstens ein Datensatz trainiert wurde, wird mit einer Wahrscheinlichkeit von 0 geführt und kann somit vom Klassifikator nicht richtig zugeordnet werden [17].

2.1.5 Performancemaße für Klassifikatoren

Um zu bewerten, wie gut ein Klassifikator für ein gestelltes Problem geeignet ist, werden Performancemaße verwendet. Zunächst bietet eine Konfusionsmatrix eine gute Übersicht über die Performance eines Klassifikators. In dieser werden die tatsächlichen Klassenzugehörigkeiten den vom Klassifikator vorhergesagten Klassenzugehörigkeiten gegenüber gestellt. Tabelle 2-1 zeigt dies exemplarisch für eine binäre Klassifikation mit den beiden Klassen „positive“ und „negative“.

Tabelle 2-1: Konfusionsmatrix, nach [20]

	positive prediction	negative prediction
positive class	true positives (TP)	false negatives (FN)
negative class	false positives (FP)	true negatives (TN)

Aus der Konfusionsmatrix lassen sich weitere Kennzahlen berechnen. Das wohl einfachste Maß ist die Gesamtgenauigkeit, auch unter dem englischen Begriff Accuracy bekannt [20]. Dieses Maß bildet sich aus der Anzahl aller korrekt klassifizierten Datensätze geteilt durch die Menge aller verarbeiteten Datensätze. Auf Basis der obigen Konfusionmatrix ist die Gesamtgenauigkeit also wie in Gleichung 2-8 definiert.

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2-8}$$

Weitere Gütemaße bilden zudem die Genauigkeit (engl. Precision) und die Sensitivität (engl. Recall) des Klassifikators [18, 20]. Mit der in Gleichung 2-9 gegebenen Precision lässt sich angegeben, welche der als positiv vorhergesagten Daten auch tatsächlich der positiven Klasse angehören [18, 21].

$$precision = \frac{TP}{TP + FP} \tag{2-9}$$

Der Recall (siehe Gleichung 2-10) gibt dagegen an, welche der tatsächlich der positiven Klasse angehörigen Datensätze auch als positiv klassifiziert wurden [18, 21]. Dieses Maß wird daher auch als Trefferquote (engl. Hit Rate) bezeichnet [20, 21]

$$recall = \frac{TP}{TP + FN} \tag{2-10}$$

Darüber hinaus wird häufig das F1-Maß (auch balanciertes F-Maß oder F1-Score) verwendet, um die beiden vorgenannten Maße zusammenfassend in einem Wert wiederzugeben [21]. Es handelt sich dabei um eine Form des in Gleichung 2-11 gegebenen F-Maßes [20].

$$F_\beta = (1 + \beta) \cdot \frac{precision \cdot recall}{\beta \cdot precision + recall} \tag{2-11}$$

Über den Parameter β lässt sich bestimmen, wie stark die Precision und der Recall das F-Maß beeinflussen sollen [20]. So erhält der Recall größere Beachtung, wenn $\beta > 1$. Soll die Precision mehr ins Gewicht fallen, sollte daher ein $\beta < 1$ gewählt werden. Beim F1-Maß wird $\beta = 1$ gesetzt, sodass beide Werte gleich gewichtet sind [20]. Gleichung 2-12 zeigt die angepasste Formel und formt diese auch gleich um, um eine Berechnung anhand der oben vorgestellten Konfusionsmatrix zu ermöglichen.

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall} = \frac{2 \cdot TP}{(TP + FP) + (TP + FN)} \tag{2-12}$$

Die vorgestellten Maße eignen sich so zunächst nur für Klassifikatoren, die binäre Probleme verarbeiten. Um sie auch zur Bewertung der Performance von Klassifikatoren mit mehreren Klassen zu verwenden, gibt es zwei Ansätze: Micro- und Macro-Averaging [21, 22]. Beide wichten die auftretenden Klassen unterschiedlich.

Wird eine Konfusionsmatrix über alle Klassen gebildet und die Summe der TP, FP, TN und FN über Klassengrenzen hinweg bestimmt, um daraus die weiteren Maße zu berechnen, so spricht man von Micro-Averaging [21, 22]. Dadurch beeinflusst jeder Datensatz die Maße gleichermaßen, sodass häufiger vorkommende Klassen größeren Einfluss auf das Ergebnis haben als seltener vorkommende [21, 22].

Beim Macro-Averaging wird die korrekte Zuordnung der Daten zu jeder Klasse dagegen separat als binäres Problem betrachtet. Dann werden für jede Klasse die einzelnen Maße berechnet und anschließend der Durchschnitt gebildet [21, 22]. So nehmen alle Klassen gleichermaßen Einfluss auf die Maße [21, 22].

2.2 Farbmodelle

Der Begriff Farbmodell wird verwendet um die mathematische Darstellung von Farben zu beschreiben [23]. Dabei werden einer Farbe mehrere Zahlenwerte zugewiesen um sie eindeutig zu definieren [23]. In der vorliegenden Arbeit werden das RGB- und das HSL-Modell zur Darstellung verwendet, weshalb diese im Folgenden kurz vorgestellt werden.

2.2.1 RGB

Im Digitalen Bereich wird meist das RGB-Modell verwendet [23], da es zum Beispiel von Monitoren zur Darstellung von Farbe verwendet wird [24]. Auch deshalb lassen sich die meisten anderen Farbmodelle in das RGB-Modell umrechnen [25] und es sollte als grundlegendes Farbmodell kurz betrachtet werden.

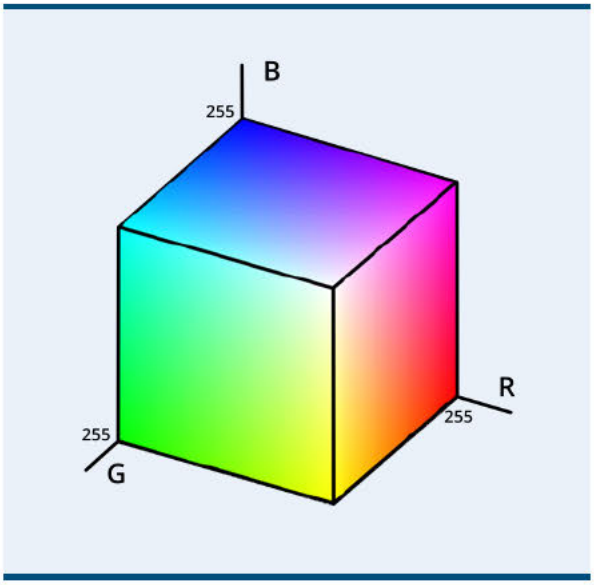


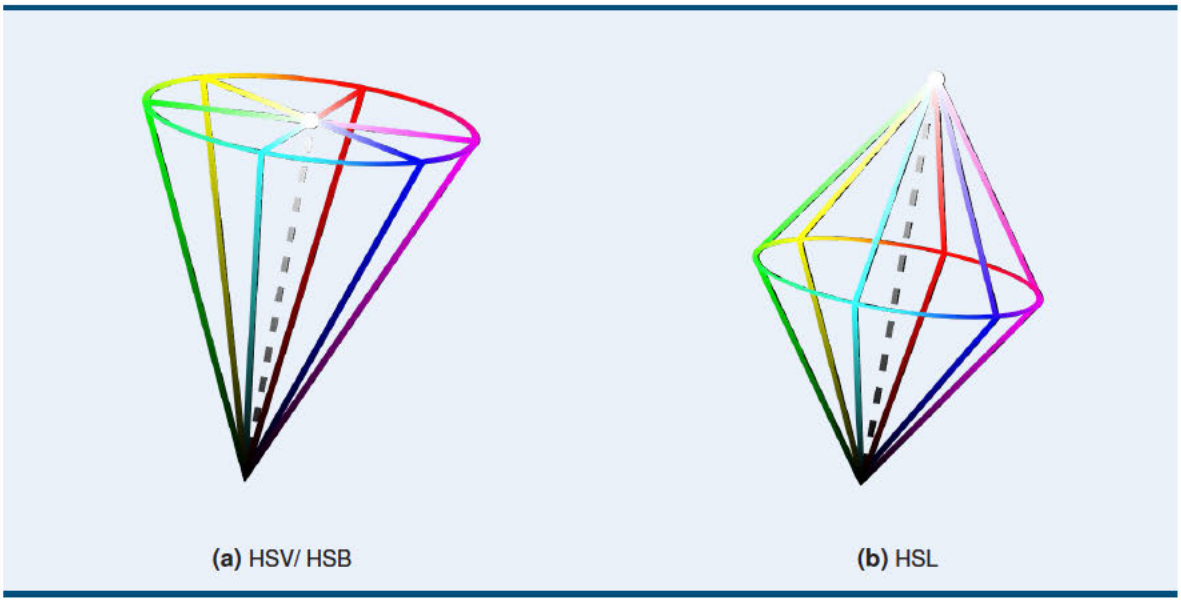
Abbildung 2-6: RGB-Farbmodell, aus [26, bearbeitet]

Das RGB-Modell basiert auf den drei Grundfarben rot (R), grün (G) und blau (B), welche auf einer Skala von 0 bis 255 angegeben werden [23]. Dargestellt wird es meist in einem kartesischen Koordinatensystem mit den drei Farben als Achsen [27]. So entsteht die typische Darstellung des RGB-Würfels, welche in Abbildung 2-6 gezeigt ist.

In diesem Modell werden Farben durch die additive Kombiation der drei Grundfarben gebildet [25]. Dabei wird die Farbe Weiß durch (255,255,255) definiert und Schwarz entsteht, wenn die Werte für alle Farben null sind, also (0,0,0) [23].

2.2.2 HSL

Zu den vom RGB-Modell abgeleiteten Farbmodellen zählt unter anderem das intuitivere HSL-Modell [28, 29]. Auch hier wird eine Farbe durch drei Werte beschrieben. Diese geben jeweils den Farbton H (von engl. hue), die Sättigung S (von engl. saturation) und die Helligkeit L (von engl. lightness) wieder [23, 28, 29]. Das Modell wird meist in Form eines Doppelkegels dargestellt (siehe Abbildung 2-7b), weshalb der Farbton H meist als Wert zwischen 0° und 360° dargestellt wird. Die Helligkeit L und die Sättigung S werden in der Regel in Prozent angegeben, wobei die Sättigung der Intensität der Farbe entspricht [23]. Ist $S = 0$, so variiert die beschriebene Farbe, unabhängig vom Wert des Farbtons, zwischen schwarz ($L = 0\%$) und weiß ($L = 100\%$), ist also meist ein Grauton [28, 29]. Dies ist in Abbildung 2-7b an der gestrichelten Linie erkennbar.



(a) HSV/ HSB **(b)** HSL

Abbildung 2-7: Vergleich HSL und HSV/HSB, eigene Darstellung nach [28]

Die Abbildung 2-7a zeigt außerdem die verwandten Modelle HSV und HSB. Hier wird statt L der Wert V (von engl. value) beziehungsweise die Farbbrillanz B (von engl. brightness) verwendet [28, 29]. Da sie sich sehr ähnlich sind, wurden sie in dieser Darstellung zusammengefasst [28]. Der größte Unterschied zum HSL-Format ist die Darstellung von HSV und HSB als einfacher Kegel. Auch hier verläuft die Helligkeit vertikal von schwarz bis weiß. Im Gegensatz zum HSL-Modell, erreicht man im HSV- oder HSB-Modell jedoch nur dann reines Weiß, wenn die Sättigung $S = 0\%$ ist. Im HSL-Modell ist eine Farbe Weiß, wenn die Helligkeit $L = 100\%$ ist. Der Wert der Sättigung S spielt hier keine Rolle.

Die einfachste Möglichkeit, den vom HSL-Modell abbildbaren Farbraum auch zweidimensional darzustellen, liegt darin die Sättigung auf $S = 100\%$ festzulegen. Dies wurde in Abbildung 2-8 umgesetzt. Die prozentualen Skalen von S und L werden hier alternativ von 0 bis 1 dargestellt.

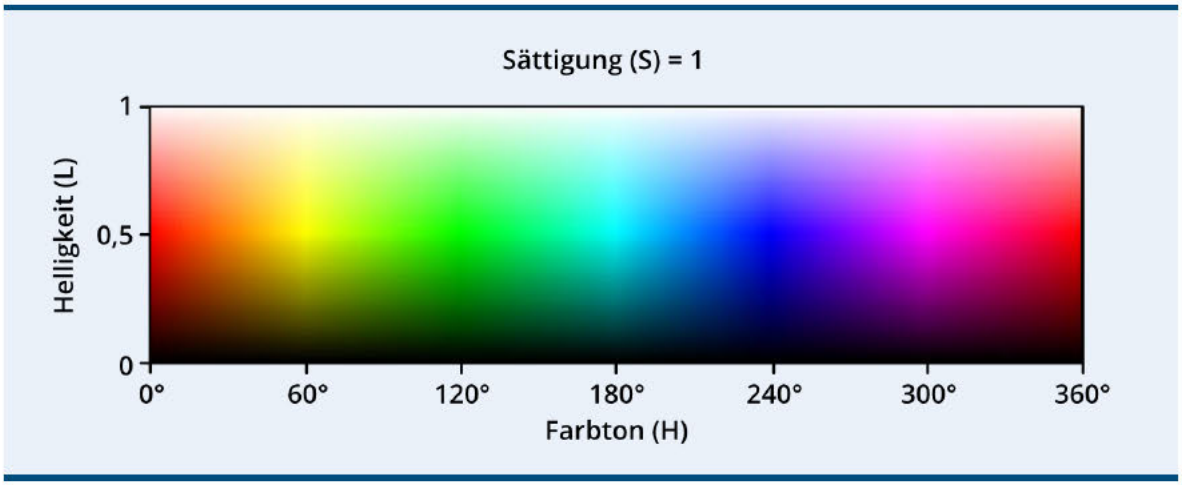


Abbildung 2-8: HSL-Farbraum, eigene Darstellung

Da im HSV-Modell die Farbe weiß nur dann erreicht wird, wenn die Sättigung $S = 0$ ist, würde Abbildung 2-8 für das HSV-Modell anders aussehen. Dort würde quasi nur die untere Hälfte der Darstellung mit $0 \leq L \leq 0,5$ zu sehen sein, obwohl B beziehungsweise V trotzdem von null bis eins dargestellt wären. Insofern lassen sich in zweidimensionalen Darstellungen mit dem HSL-Modell mehr Farben darstellen, als mit HSV/HSB. Daher wird später in der vorliegenden Arbeit auf das HSL-Modell zurückgegriffen.

2.3 IT-Forensik

Mit der stetig steigenden Anzahl elektronischer Geräte, wie Handys, Laptops oder Kameras [30], steigt auch die Wahrscheinlichkeit, dass diese Technologien für kriminelle Zwecke eingesetzt werden. Die digitale oder auch IT-Forensik umfasst daher verschiedenste Techniken und Methoden zum Auffinden, Extrahieren, Analysieren und Interpretieren von Binärdaten [31]. Ziel dieser Methoden ist es, digitale Spuren zu finden, die eine im Ermittlungsprozess aufgestellte These stützen oder widerlegen [32] und so zur Aufklärung von strafbaren Handlungen beitragen können [33].

Auch wenn die IT-Forensik immer noch eine recht neue Wissenschaft ist [31, 34], wird viel Wert auf die Wissenschaftlichkeit der angewendeten Methoden gelegt [30]. Zu den Anforderungen an IT-Forensische Untersuchungen zählen daher unter anderem die Wiederholbarkeit und Nachvollziehbarkeit aller angewandten Methoden, eine angemessene Dokumentation aller Schritte und die Wahrung der Integrität des originalen Beweismittels [30, 35].

Binärdaten können an vielen Stellen, wie zum Beispiel im Arbeitsspeicher, im Dateisystem oder auch im Netzwerkverkehr, auftreten [36]. Die IT-Forensik zählt daher viele Teilbereiche. Zu ihnen zählen unter anderem die Netzwerk-Forensik, die Computer-Forensik, die Mobilgeräte-Forensik, die Live-Forensik oder die Dateisystem-Forensik [31, 32].

Als einer der genannten Teilbereiche der IT-Forensik beschäftigt sich die Dateisystem-Forensik mit der Analyse von Dateisystemen. Bei diesem Teilgebiet liegt der Fokus auf dem Auffinden von Dateien, aber auch auf der Wiederherstellung von gelöschten oder versteckten Dateien in einem Dateisystem [32]. Neben Dateiinhalten werden als Ergebnisse einer solchen Analyse auch Dateifragmente oder zu Dateien gehörende Metadaten erwartet [32].

Da in der vorliegenden Arbeit eine Methodik entwickelt werden soll, welche unter anderem zur Ergänzung der bereits bestehenden Techniken der Dateisystem-Forensik genutzt werden kann, werden in den folgenden zwei Abschnitten einige Grundlagen zu Dateisystemen und zu den bereits bestehenden Methoden zur Datenwiederherstellung beschrieben.

2.3.1 Speichern und Löschen von Dateien im Dateisystem

Zur Organisation von Dateien auf Speichermedien können verschiedene Dateisysteme verwendet werden. Diese verknüpfen den Namen einer Datei mit ihrem Inhalt im Speicher und helfen dem Nutzer so beim Wiederfinden von Dateien [32]. In der Umsetzung allokiert das jeweilige Betriebssytem beim Erstellen oder Ändern einer Datei durch das Dateisystem die entsprechend benötigte Menge freien Speicherplatz [32]. Dies geschieht aus Effektivitätsgründen immer blockweise in gleich großen Datenabschnitten, welche je nach verwendetem Dateisystem als Cluster oder Blöcke bezeichnet werden [32, 33]. Diese Abschnitte bilden die kleinste ansprechbare Speichereinheit und sind in der Regel einen oder mehrere Sektoren à 512 Byte groß [33]. Je nachdem, welche Strategie zum Allokieren der Sektoren verwendet wird, kann es bei der initialen Erstellung und Speicherung einer Datei vorkommen, dass diese nicht in einem Stück, sondern fragmentiert im Speicher abgelegt wird [32]. Dies kann die Wiederherstellung der Datei erschweren, sollte sie später einmal gelöscht werden.

Ob gelöschte Dateien überhaupt wiederherstellbar sind, hängt davon ab, wie sie gelöscht wurden. Während es Tools mit „Secure Erase“-Funktionen gibt und einige Betriebssysteme dies auch anbieten, markieren die meisten Betriebssysteme bei Löschen einer Datei die zugehörigen Sektoren als unallokiert, löschen deren Inhalt aber nicht [32]. So wird die Datei für den normalen Benutzer unsichtbar, jedoch ist es mit den Methoden der IT-Forensik oft möglich, Sektoren mit Informationen gelöschter Dateien wiederherzustellen, sofern diese nicht neu allokiert und beschrieben wurden [37]. Ein vollständiges Löschen der Dateien wäre sehr aufwändig und wird daher in der Regel aus Performancegründen nicht durchgeführt [37]. Soll ein Speichermedium doch einmal vollständig gelöscht werden, so kommen spezielle Lösch-Prozeduren zum Einsatz, welche den gesamten Speicherbereich des Mediums überschreiben. Dies wird jedoch nicht bei allen Speichermedien gleichermaßen umgesetzt, sodass teilweise auch in einer gewissen Zeitspanne nach Erteilung des Lösch-Befehls mit forensischen Methoden noch Daten wiederherstellbar sind [38].

Neben dem absichtlichen Löschen von Dateien zum Verbergen relevanter Daten [37], gibt es auch andere Eigenheiten in Datei- und Betriebssystemen, die eine Datenwiederherstellung nötig machen können. Beispielsweise besteht die Möglichkeit, dass ein Nutzer die Speicheradressen, an denen er wichtige Informationen abgelegt hat, als defekt markiert [32]. Diese Funktion wird normalerweise verwendet, um zu verhindern, dass Daten in defekte Sektoren geschrieben werden. Ohne diese

Kennzeichnung wäre nicht garantiert, dass die Daten später wieder fehlerfrei ausgelesen werden können. Als defekt markierte Sektoren werden vom Betriebssystem ignoriert und somit nicht mehr im Dateisystem aufgeführt, weshalb sie nur mit speziellen Methoden wieder auffindbar sind [32].

2.3.2 Arten der Datenwiederherstellung

In der Forensik kommen hauptsächlich drei verschiedene Methoden zur Anwendung, wenn Daten wiederhergestellt werden sollen. Diese Methoden werden jeweils von einer ganzen Reihe verschiedener forensicher Tools implementiert, jedoch finden nicht alle Tools alle Daten im unallokierten Bereich [39]. Daher sollten zur Datenwiederherstellung nicht nur verschiedene Methoden, sondern auch unterschiedliche Tools angewendet werden [39].

Die einfachste der drei Wiederherstellungsmethoden bildet die String-Search [40]. Bei dieser wird der zu analysierende Speicher entweder nach spezifischen Strings durchsucht, bei Bedarf auch unter Beachtung des Kontextes [41], oder es wird generell nach Abfolgen druckbarer Zeichen gesucht, um anhand dieser weitere Daten zu finden [36, 40, 42]. Bei einer weiteren Methode werden Metadaten als Überbleibsel gelöschter Dateien ausgewertet. Teilweise beinhalten diese immer noch Zeiger, die auf die Speicheradresse des zwischenzeitlich gelöschten Dateiinhaltes verweisen [32].

Darüber hinaus wird häufig das sogenannte Carving angewendet [39]. Dabei wird der Speicher nach für einen bestimmten Dateityp charakteristischen Signaturen durchsucht, die im Header und Footer von Dateien dieses Typs vorkommen [32, 39, 40]. Beispielsweise wird der Beginn einer jpg-Datei durch die Bytes `0xFFD8` angezeigt, während das Ende mit `0xFFD9` markiert ist [32, 40]. Werden solche Signaturen gefunden, wird der Speicherbereich zwischen der Header- und der Footer-Signatur extrahiert und als neue Datei aufbereitet [32, 40]. Wird eine Header-Signatur gefunden, ohne dass eine zugehörige Footer-Signatur festzustellen ist, wird das Dateiende gemäß Benutzervorgabe oder anhand der dateitypspezifischen Maximallänge festgelegt [32, 40].

2.4 Visualisierung von Binärdaten

Verschiedenste Tätigkeiten wie das Zusammensetzen von Dateifragmenten, das Identifizieren von kryptographischen Schlüsseln oder das Erkennen von Malware erfordern ein tiefes Verständnis der Zusammensetzung von Binärdaten [36]. Es ist daher nicht verwunderlich, dass bisherige Ansätze, bei denen Daten zu Auswertungszwecken visualisiert wurden, aus den Bereichen der Forensik, des Reverse Engineering und der Malware-Analyse stammen. Im Folgenden werden einige dieser Ansätze näher betrachtet, da sie die Grundlage für die vorliegende Arbeit bilden und einen Überblick über ähnliche Arbeiten geben.

2.4.1 im Reverse Engineering

Ebenso wie in der IT-Forensik müssen auch in der IT-Sicherheit große Datenmengen verarbeitet und analysiert werden [2], weshalb Methoden und Tools, die die Analyse dieser Datenmengen in möglichst geringer Zeit ermöglichen, immer wichtiger werden. Im Gegensatz zur Beweismittelsuche in der Forensik geht es im Reverse Engineering meist darum, fremde oder proprietäre Software zu analysieren um so ihre Funktionsweise zu verstehen oder um mögliche Gefahren, wie zum Beispiel

Schadcode, zu erkennen [36]. Weitere Ziele des Reverse Engineering sind das Finden von Sicherheitslücken und das Verstehen von proprietären Verschlüsselungs- und Authentifizierungsverfahren, mit dem Ziel, diese angreifbar zu machen. So können beispielsweise Angriffe auf diese Verfahren entwickelt werden, welche wiederum im Bereich der IT-Forensik Anwendung finden können.

Traditionellerweise wird beim RE zwischen den beiden Teilbereichen des kontextunabhängigen und des semantischen Reverse Engineering unterschieden. Beim kontextunabhängigen RE hat der Analyst kein Vorwissen über die Datei, sodass zur Betrachtung der Daten hauptsächlich Hex-Editoren oder eine Bytehäufigkeitsanalyse zur Auswahl stehen [42]. Beim semantischen RE hat der Analyst bereits etwas Vorwissen; beispielsweise ist der Dateityp bekannt, sodass eine Analyse der Datei mittels Dissassembler oder Debugger möglich ist [42]. Ist bekannt, dass es sich bei einer zu analysierenden Datei um eine ausführbare Datei handelt, so stehen im Teilbereich des Reverse Code Engineering die statische Analyse der Datei ohne Ausführung des Codes oder die dynamische Analyse, bei der das Verhalten und die Interaktionen der Datei während der Ausführung beobachtet werden, zur Auswahl [42].

Während die kontextunabhängige Analyse mit Hex-Editoren und Häufigkeitsanalyse sehr exakt und auch flexibel interpretierbar ist, ist diese Methode auch sehr komplex und benötigt geübte und erfahrene Analysten [3, 42]. Die Auswertung von dissassemblierten Befehlen ist dagegen klarer, allerdings lassen sich die Befehle nur dann korrekt deuten, wenn die vom Ersteller der Daten verwendeten Compiler und Assembler bekannt sind [42]. Auf der Suche nach flexibleren Analysemethoden, die die bisherigen bei der Analyse und Identifikation von unbekannten Datenstrukturen ergänzen können, wurde daher auch im Bereich des RE die Visualisierung von Daten bereits erforscht. Im Folgenden werden die Forschungsergebnisse von drei Personen aus dem Reverse Engineering vorgestellt.

2.4.1.1 nach Gregory Conti

Gregory Conti beschäftige sich mit verschiedenen Methoden zur Visualisierung von Daten. Er begann zunächst damit, Dateien visuell darzustellen, indem er den enthaltenen Bytes aufgrund ihres Wertes einen Grauton zuwies, wobei `0x00` schwarz und `0xFF` weiß entsprach [36]. Die entstandenen Pixel wurden dann aneinander gereiht, sodass ein Bild entstand [36, 43]. Dabei stellte er fest, dass selbst in unstrukturierten Daten Strukturen erkennbar sind [3].

Zudem fiel ihm auf, dass aufeinanderfolgenden Bytes in Beziehung zu einander zu stehen scheinen [4]. Diese Beziehung visualisierte er mittels einer Digramm-Analyse. [3, 4, 43]. Dabei werden die einzelnen Elemente einer Bytefolge in Zweiergruppen betrachtet und in ein Diagramm eingetragen. Der erste Wert eines Bytepaars wird immer entlang der x-Achse, der zweite Wert entlang der y-Achse gemessen. Dieses Vorgehen wird in Abbildung 2-9 beispielhaft anhand des Wortes „Digramm-Analyse“ gezeigt.

Bei Betrachtung des entstandenen Diagramms fällt auf, dass sich die Punkte in mehreren Bereichen zu sammeln scheinen. Die größte Gruppe bilden die Digramme, die aus zwei Kleinbuchstaben bestehen. Auch die beiden Punkte die jeweils ein Digramm aus einem Großbuchstaben und einem Kleinbuchstaben darstellen, liegen recht nah beieinander. Nur die beiden Digramme, die ein Sonderzeichen enthalten, liegen im Diagramm etwas abseits. So entstehen Muster im Diagramm, die

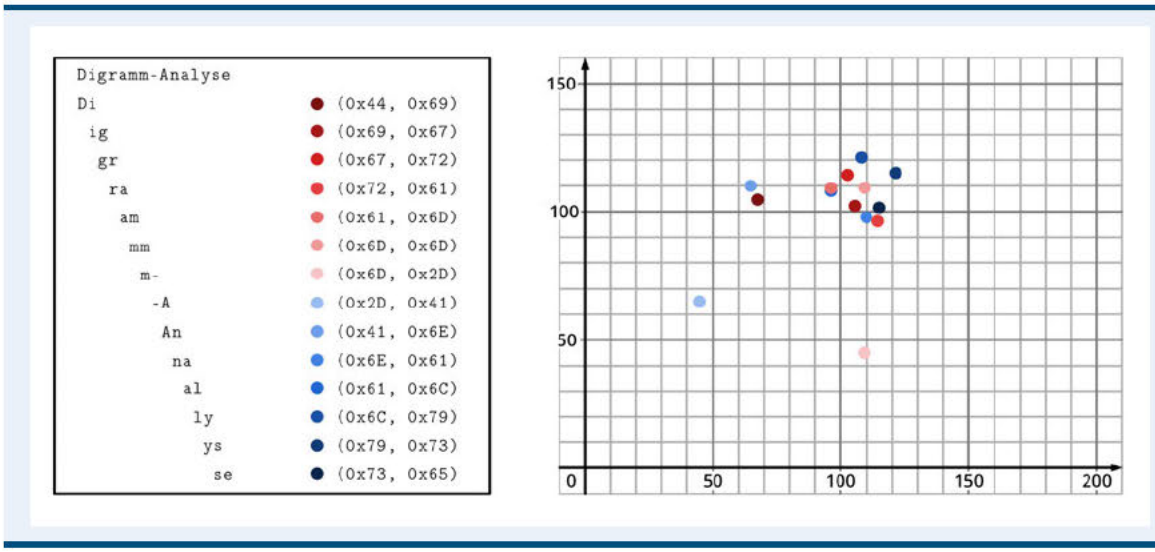


Abbildung 2-9: Beispiel Digramm-Analyse, eigene Darstellung

einem geübten Betrachter dabei helfen können, einen Überblick über die Inhalte einer Datei zu gewinnen. Beispielsweise lassen sich innerhalb einer Datei verschiedene Strukturen unterscheiden oder interessante Regionen ausfindig zu machen [2].

Neben der Digramm-Analyse beschäftige sich Conti auch mit verschiedenen anderen statistischen Methoden, wie zum Beispiel der Berechnung der Bytefrequenz oder Entropie einer Datei, um Dateien zu visualisieren und darauf aufbauend seine Analysen zu verbessern [42]. Diese und weitere Methoden zur graphischen Darstellung von Binärdaten bilden Module der Tools BinVis und Dangly-Bytes, die er zusammen mit Erik Dean entwickelte [2, 42]. Die folgende Abbildung 2-10 zeigt einen Screenshot aus einem der beiden Tools.

Neben der einfach Darstellung der Bytes über einen Farbton von schwarz für `0x00` bis leuchtend grün `0xFF` in dem mit c markierten Fenster kommen in BinVis noch weitere Visualisierungstechniken zur Anwendung [42]. So wird in der Ansicht unter b gezeigt, welche Werte von `0x00` bis `0xFF` jeweils in der gleichen Zeile unter c vorkommen [42]. So lassen sich beispielsweise Zeilen mit ASCII-Werten identifizieren, da diese nur in den Spalten 32 bis 127 der Darstellung vorkommen [42]. Ist eine Zeile fast vollständig grün eingefärbt, so ist die Wahrscheinlichkeit sehr hoch, dass in diesem Bereich der Datei verschlüsselte oder komprimierte Informationen vorliegen [42].

2.4.1.2 nach Aldo Cortesi

Auch Cortesi beschäftigte sich bereits mit der Visualisierung von Binärdaten. Sein Hauptziel lag dabei darin, auf möglichst einfache Weise einen schnellen Überblick über den Aufbau einer Datei zu gewinnen [44]. Dazu repräsentiert er die Bytes einer Datei in Pixeln, die er entsprechend dem Wert eines jeden Bytes einfärbt. Eine einfache, von ihm vorgeschlagene Farbskala unterscheidet in die vier Kategorien `0x00` (schwarz), `0xFF` (weiß), druckbare Zeichen (blau) und Rest (rot). Dies hebt Füllzeichen und vor allem Strings hervor [44], welche für den Mensch einfach zu interpretieren sind und so zum Verständnis des Dateiinhalts beitragen können. Für die Anordnung der Pixel schlug er die in Abbildung 2-11 gezeigten Kurven vor [44].

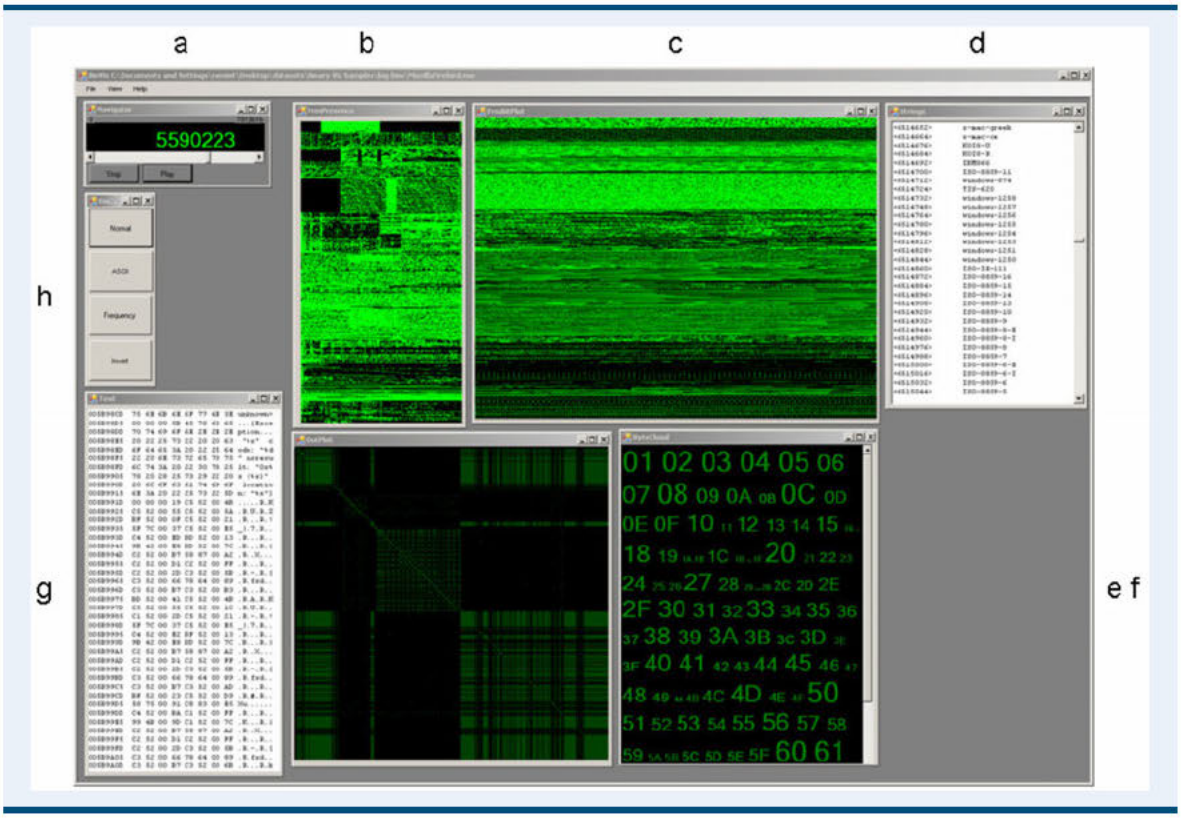
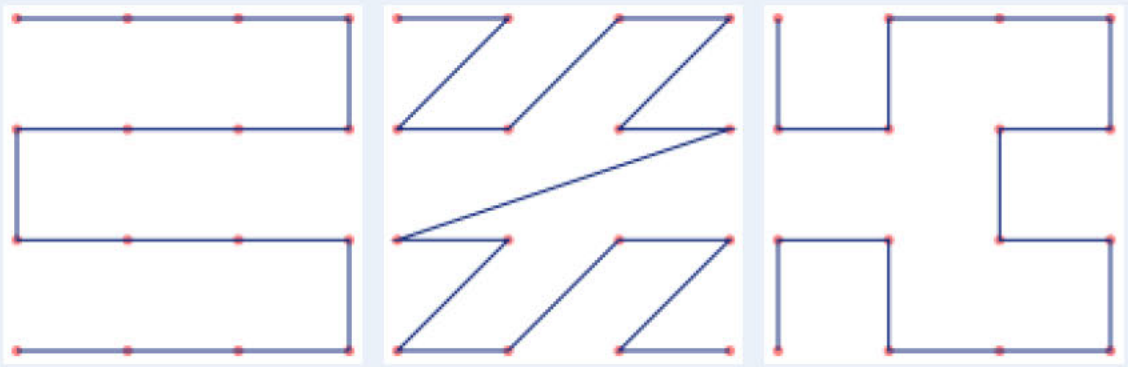


Abbildung 2-10: BinVis Tool, aus [42]

Wie gut zu sehen ist, sind kleinere Regionen gleicher Farbe in den beiden rechten Darstellungen besser zu erkennen als in der linken [3, 44]. Eine gutes Beispiel ist die schmale schwarze Linie am oberen Rand von Abbildung 2-12a, die in den Abbildungen 2-12b und 2-12c in der oberen linken Ecke gebündelt wird und dadurch größer erscheint. Ebenso sind im unteren Drittel der Zickzack-Darstellung schmale blaue und rote Linien zu erkennen, die durch die anderen beiden Kurven besser erkennbar dargestellt werden. Bei genauerer Betrachtung fällt zudem auf, dass bei der Visualisierung auf Basis der Lebesgue-Kurve einzelne Regionen nicht zusammenhängend dargestellt werden, obwohl es sich um aufeinanderfolgende Bytes handelt [44]. Beispielhaft ist hier der relativ mittig dargestellte blaue Bereich. In der rechten Abbildung zeigt sich dieser dagegen ohne sichtbare Unterbrechung. Würde dies bei einer sehr kleinen Anzahl zusammenhängender Bytes gleicher Kategorie passieren, würden diese unter Umständen gar nicht wahrgenommen werden. Von den vorgestellten Kurven ist die Hilbertkurve daher am Besten geeignet, um zu analysierende Daten zu visualieren [44]. Trotzdem bringt die Verwendung der Hilbertkurve nicht nur Vorteile. So lässt sich aus der Visualisierung die etwaige Position einer genauer zu untersuchenden Stelle nicht einfach in der Hexadezimal-Ansicht wiederfinden [44]

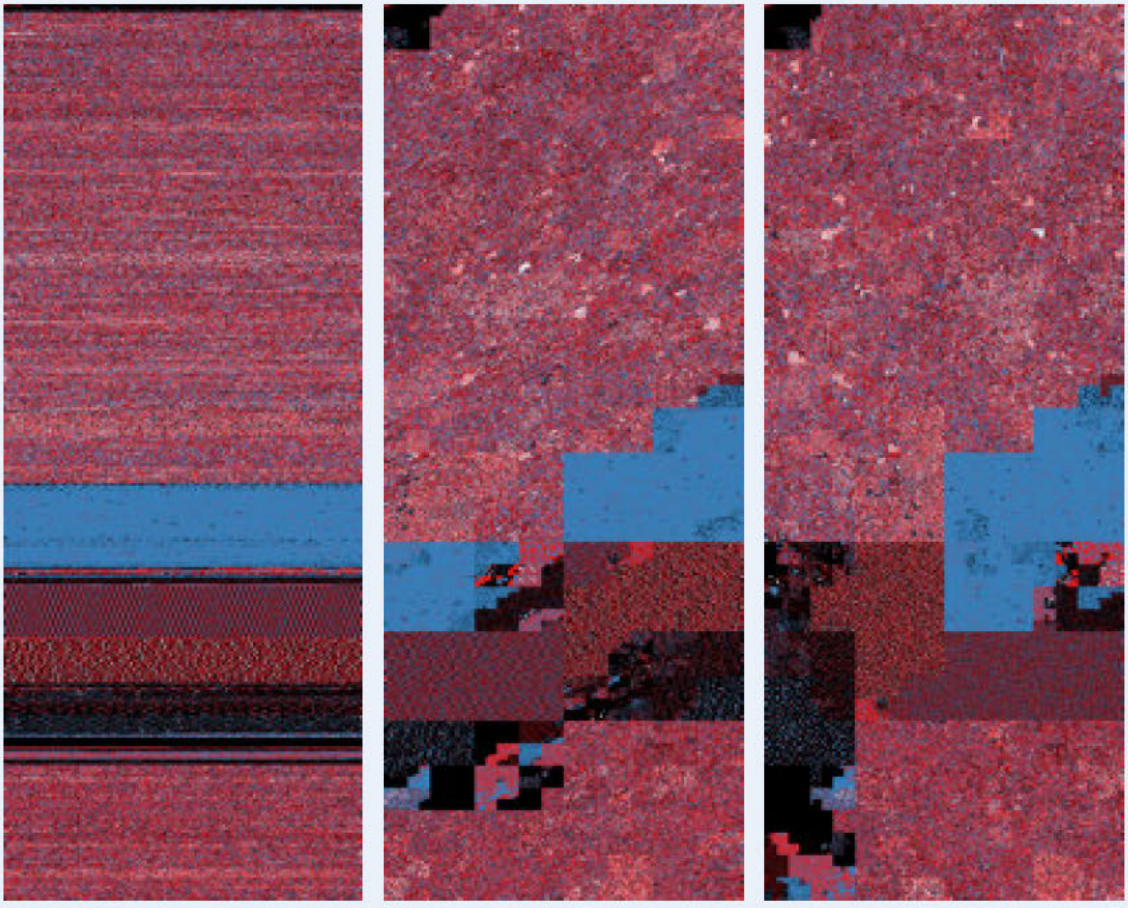
Dieses Problem löste Cortesi mit der Entwicklung der interaktiven Webanwendung binvis.io, die das eben vorgestellte Konzept der Visualisierung von Binärdaten umsetzt [45]. Das Tool ist unter binvis.io zu erreichen und lässt den Nutzer eigene Dateien hochladen und visualisieren. Abbildung 2-13 zeigt die Oberfläche der Anwendung, nachdem eine Datei hochgeladen wurde. Rechts neben der Visualisierung ist die Datei im Hexadezimal-Format dargestellt. Bewegt der Nutzer die Maus über die Visualisierung, wird die entsprechende Passage im Bild und in der Hexadezimal-Ansicht farblich hervorgehoben [45, 46]. Dies funktioniert umgekehrt genauso.



(a) Zickzack **(b)** Lebesgue-Kurve **(c)** Hilbertkurve

Abbildung 2-11: Zum Darstellen der Pixel verwendete Kurven, aus [44]

Zudem fällt auf, dass zu der zuvor vorgestellten Farbskala ein Farbton hinzugekommen ist. Gab es vorher nur vier Farben für `0x00`, `0xFF`, druckbare Zeichen und alle restlich, so wurde die letzte Kategorie nun weiter aufgeteilt. Die druckbaren Zeichen und Füllzeichen behalten ihre Farbe. Der Rest wurde dagegen aufgeteilt. Die zwischen `0x01` und `0x1F` liegenden Werte werden mit Ausnahme der Textzeichen `0x09`, `0x0A` und `0x0D` nun grün dargestellt [47]. Werte zwischen `0x7F` und `0xFE` bleiben auch weiterhin rot [47].



(a) Zickzack **(b)** Lebesgue-Kurve **(c)** Hilbertkurve

Abbildung 2-12: Beispiel der Visualisierung nach Cortesi, aus [44, bearbeitet]

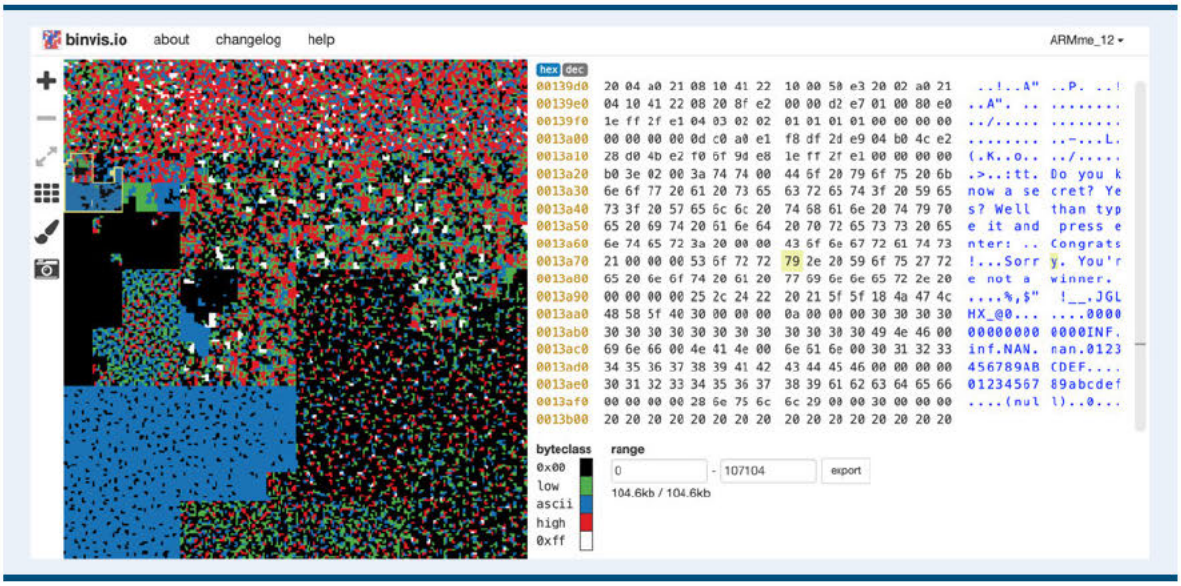


Abbildung 2-13: Screenshot des Tools binvis.io [45]

Darüber hinaus kann die Ansicht in binvis.io zwischen der Darstellung mittels Hilbertkurve oder der einfacheren Zickzack-Kurve umgeschalten werden [46, 47]. Auch verschiedene weitere Farbschemata, zum Beispiel auf Basis der Entropie zur Erkennung von kryptographischem oder komprimiertem Material in der Datei, werden angeboten [46, 47].

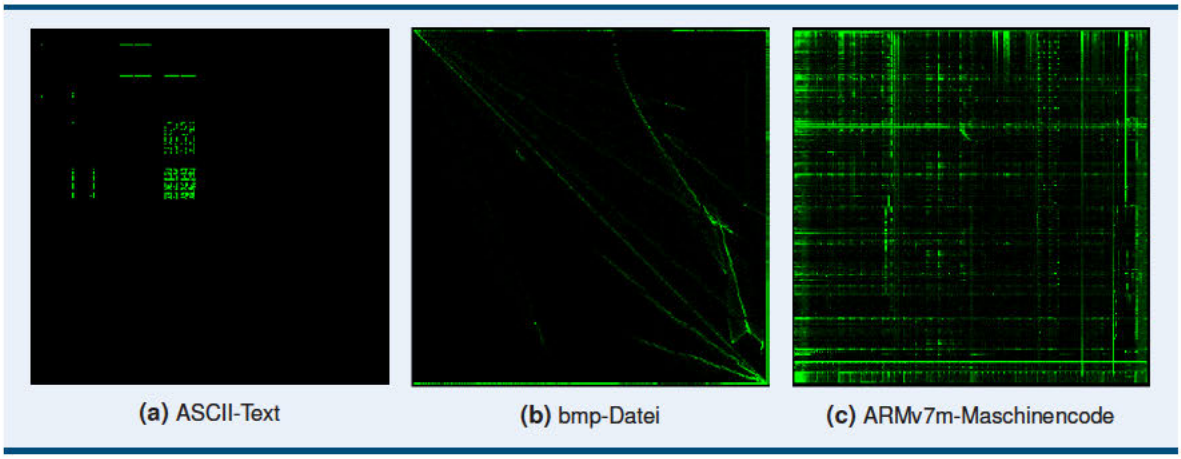
2.4.1.3 nach Christopher Domas

Christopher Domas baute mit seiner Forschung auf den Erkenntnissen von Conti und Cortesi auf. Ziel seiner Arbeit war es, eine neue Methodik fürs Reverse Engineeringg zu finden. Dies begründete er vor allem mit der stetigen und immer schneller werdenden Weiterentwicklung von Informationen. Domas [3] argumentiert, dass dies problematisch ist, da die eingesetzten RE-Tools nicht zur Analyse unbekannter Datenstrukturen geeignet sind. Zudem ist die Analyse von Daten schwierig, wenn diese z.B. ohne oder mit obfuskiertem (=verschleiertem) Header auftreten, Steganographie oder ein aus mehreren Elementen bestehendes Binary Large Object (Blob) beinhalten oder in einem unbekannten Format abgelegt wurden. Die von ihm zu entwickelnde Methodik sollte also in der Lage sein, diese Probleme zu umgehen, indem sie unabhängig vom Format der auszuwertenden Daten zu deren Verständnis beiträgt. Darüber hinaus sollte sie die Analysen beschleunigen, da die Menge aller auf der Welt gespeicherten Daten [48] und damit auch die Menge der zu analysierenden Daten immer weiter wächst [3].

Auf Basis dieser Überlegungen entwickelte Domas [3] die Methode der „Dynamic Binary Visualization“ (Dynamische Visualisierung von Binärdaten). Diese wird vor allem durch das von ihm entwickelte Tool „..cantor.dust..“ verwirklicht. Mit diesem Programm führte er verschiedene statistische Analysen über ein ausgewähltes Code- oder Datenfragment aus. Entsprechend der oben vorgestellten Ansätze von Conti und Cortesi verwendet er unter anderem die Hilbertkurve für seine Darstellungen, Histogramme zur Bewertung der Byte-Häufigkeit und die Entropie zur Erkennung von Teilbereichen in Datensätzen [3]. Darüber hinaus erweiterte er das Konzept der Digramme auf dreidimensionale Trigramme beziehungsweise allgemein auf n-Gramme.

Mit diesen Verfahren gelang es Domas [3], schnell und unkompliziert relevante Abschnitte in einer größeren Datei zu identifizieren. Darüber hinaus nutzte er den Naive-Bayes-Algorithmus zur Klassifizierung einzelner Regionen in einer Datei [3]. Als Vorlagen für die einzelnen Klassen dienten ihm u.a. Beispieldateien für einzelne Dateitypen. So erreichte Domas [3] bereits eine erste inhaltsbasierte Klassifizierung von Daten nach ihrem wahrscheinlichsten Datenformat.

Als Weiterentwicklung von Domas' ..cantor.dust.., veröffentlichte das Battelle-Institut das Plugin CantorDust für Ghidra, einen Open-Source-Disassembler [4]. Mit diesem können Nutzer zuerst ein grundlegendes Verständnis der zu betrachtenden Datei gewinnen, bevor sie mit Ghidras traditioneller statischer Analyse beginnen. Das Plugin bietet verschiedene Visualisierungsmöglichkeiten: neben einer Digramm-Darstellung zeigt das Plugin auch die Entropie oder die Bitmap-basierte Visualisierung eines Datenfragments [4]. Wie schon bei Domas' alleinstehendem Programm lässt sich das betrachtete Fragment dynamisch einschränken und so an die jeweiligen Daten anpassen [3, 4].



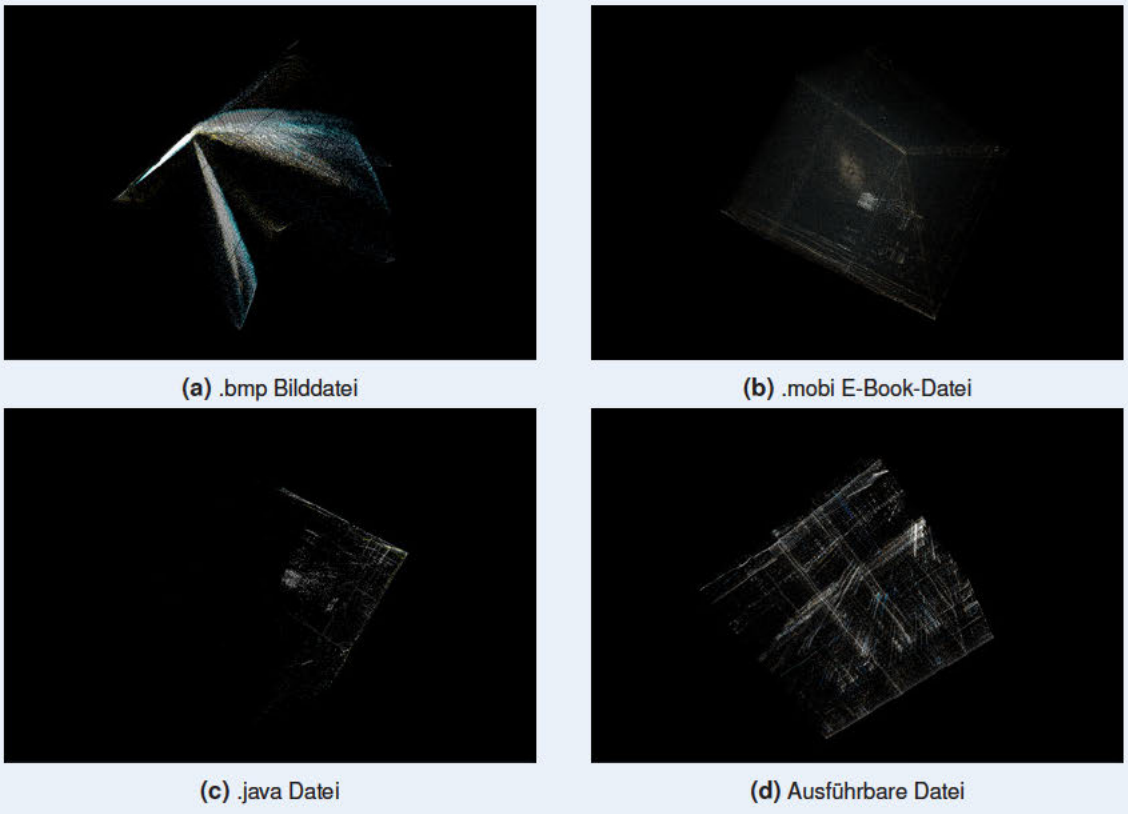
(a) ASCII-Text **(b)** bmp-Datei **(c)** ARMv7m-Maschinencode

Abbildung 2-14: Zum Darstellen der Pixel verwendete Kurven, aus [44]

In der Digramm-Darstellung konnte Domas bereits einige charakteristische Muster identifizieren, die verschiedene Dateistrukturen repräsentieren [3]. Abbildung 2-14 zeigt beispielhaft die Visualisierung von einem Ausschnitt ASCII-Text, einer Bilddatei im bmp-Format sowie einem Datenausschnitt in Maschinensprache [4]. Während Abbildung 2-14a die typischen ASCII-Quadrate beinhaltet, zeigt Abbildung 2-14b verschiedene diagonale Linien und eine leichte Umrandung des Diagramms [4]. Abbildung 2-14c zeigt dagegen ein für Maschinensprache typisches Muster aus senkrechten und waagerechten Linien, wobei die Anzahl und Position dieser Linien von der Art des Prozessors abhängt, auf den die Sprache ausgelegt ist [3, 4].

Noch bevor das Plugin von Battelle veröffentlicht wurde, wurde bereits ein Open-Source Tool nach dem von Domas vorgestellten Ansatz entwickelt [49]. Das Tool Veles erlaubt beispielsweise die Visualisierung von Digrammen und Trigrammen auf dynamisch auswählbaren Ausschnitten einer geladenen Datei [49]. Abbildung 2-15 zeigt beispielhaft einige Dateien, die mittels Trigramm-Analyse visualisiert wurden [50].

In den Visualisierungen des E-Books (2-15b) und der Java-Datei (2-15c) ist jeweils mittig eine würfelförmige Häufung von Werten erkennbar. Hierbei handelt es sich analog zu den quadratischen Punktansammlungen in Abbildung 2-14a um den Bereich der ASCII-Zeichen. Genauer gesagt scheinen hier vermehrt Kombinationen von drei aufeinanderfolgenden lateinischen Kleinbuchstaben aufzutreten. Diese Zeichen treten bei beiden Dateitypen recht häufig auf, sofern das E-Book in einer Sprache



(a) .bmp Bilddatei **(b)** .mobi E-Book-Datei

(c) .java Datei **(d)** Ausführbare Datei

Abbildung 2-15: Verschiedene Dateien visualisiert mit Veles, aus [50]

geschrieben wurde, welche lateinische Buchstaben nutzt. Die in Abbildung 2-15a gezeigt bmp-Datei weißt ähnlich zu der Visualisierung in Abbildung 2-14b diagonal von einer Ecke ausgehende Linien, jedoch sind diese in der dreidimensionalen Darstellungen zu Flächen breitgezogen. Auch die Darstellung der ausführbaren Datei in Abbildung 2-15d weist Ähnlichkeiten mit ihrem zweidimensionalen Pendant aus. Beide Dateien zeigen ein Gitter aus parallel zu den Kanten der Diagramme verlaufenden Linien.

2.4.2 in der IT-Forensik

Zum Zeitpunkt der Entstehung der vorliegenden Arbeit ist nur ein Tool aus dem Bereich der IT-Forensik bekannt, welches Ansätze zur Visualisierung von Binärdaten berücksichtigt. Der von Björn Kerler entwickelte Mobile Revelator ist ein vielseitiges Tool zur Analyse von Datensicherungen aus Android-Smartphones [51]. Neben dem Carven nach jpg-Dateien und einer Ansicht zum Durchsuchen von SQLite-Datenbanken, bietet es auch eine Hexeditor-Ansicht [51, 52]. Diese zeigt links neben den Hexadezimal-Werten einen senkrechten Balken, der in Abhängigkeit der Werte farblich eingefärbt wird und so bei der Erkennung einfacher Strukturen im Binärcode helfen kann [52]. Die Aufteilung der Farben ist dabei ähnlich zu Cortesis Darstellung in binvis.io gewählt, mit dem Unterschied, dass `0x00` und `0xFF` beide schwarz, ASCII-Zeichen in grün, die Werte darunter blau und die Werte über dem ASCII-Bereich gelb dargestellt sind [52]. Links daneben ist zusätzlich noch eine kleine Visualisierung enthalten, die Byte-Trigramme sowie die Entropie darstellt und bei der Erkennung von Dateitypen helfen soll [52]. Abbildung 2-16 zeigt einen Screenshot des Programms.

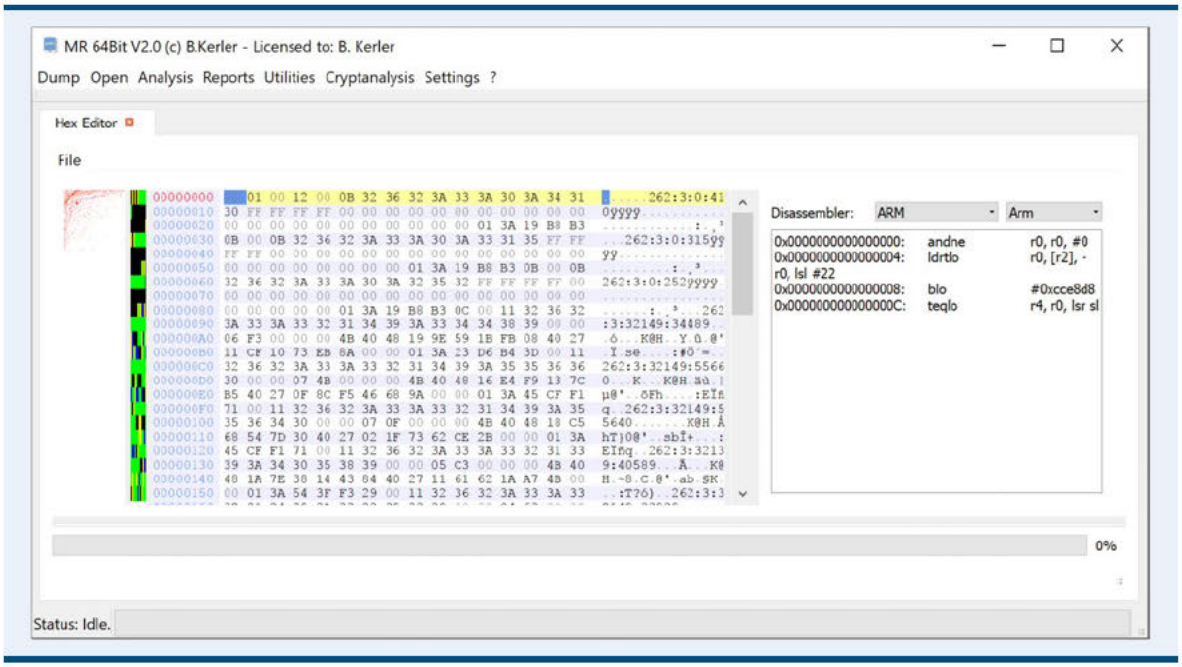


Abbildung 2-16: Screenshot des Tools Mobile Revelator, aus [52]

2.4.3 in der Malware-Analyse

Der Ansatz der Visualisierung von Binärdaten zur Klassifizierung von Datenstrukturen wird auch im Bereich der Malware-Forschung verwendet. In mehreren Studien wurde der Quellcode verschiedener Malware-Dateien in eine visuelle Darstellung überführt und anhand der darin enthaltenen Muster in Kategorien sortiert. Diese Klassifizierung erfolgte meist auf Basis von Machine-Learning-Verfahren zur Mustererkennung und -korrelation. Im Folgenden werden zwei Ansätze näher vorgestellt.

2.4.3.1 nach Ajay Singh et al.

Einen Ansatz zur Malware-Klassifizierung wurde 2020 von Singh et al. [53] vorgestellt. Dieser basiert auf den in einer visuellen Repräsentation der ausführbaren Datei sichtbaren Mustern. Die in [53] beschriebene Methodik, welche im Folgenden genauer beschrieben wird, erzeugt ein RGB-Bild. Laut den Autoren waren die Ergebnisse vorheriger Versuche mit Graustufen-Bildern nicht genau genug.

Zur Erstellung der visuellen Repräsentationen nach [53] werden alle betrachteten Executables zunächst auf Binärebene betrachtet und in 8-Bit-Einheiten aufgeteilt. Mit einer zweidimensionalen Color Map wird jedem der 8-Bit-Einheiten anhand der zwei Gruppen aus 4 Bits, aus denen sie bestehen, ein Farbwert zugeordnet. Die 8-Bit-Einheiten werden jetzt nur noch als Pixel betrachtet. Die Sequenz aus Pixeln, welche entsteht, wenn die gesamte ausführbare Datei entsprechend verarbeitet wird, wird abschließend in eine zweidimensionale Matrix übertragen und angezeigt. Um gleichbleibende Bildformate zu erhalten, haben Singh et al. die Breite dieser Matrix auf 384 Bytes festgesetzt. Die Länge des Bildes unterscheidet sich jedoch entsprechend der Größe der analysierten ausführbaren Datei. Abbildung 2-17 zeigt beispielhaft Ausschnitte von drei von Singh et al. [53] erzeugten Bildern. Darin sind deutliche Unterschiede zwischen den verschiedenen Malwaretypen erkennbar.

(a) Trojaner (b) Virus (c) Backdoor

Abbildung 2-17: Beispiele der Visualisierung von Windows-Malware von Singh et al., aus [53]

Die so erstellten Bilder werden von den Autoren [53] anschließend mit einer Kombination aus neuralen Netzwerken klassifiziert, nachdem diese einem Supervised-Learning-Prozess unterzogen wurden. Die Klassifizierung in verschiedene Malware-Kategorien, wie beispielsweise Trojaner, Virus, Backdoor oder Botnet, konnte so sehr erfolgreich umgesetzt werden. Bei ungepackter Malware erreichte die vorgestellte Methodik in den durchgeführten Tests einen F1-Score von 95% oder höher [53].

2.4.3.2 nach Aryan Marwaha et al.

Auch Marwaha et al. beschäftigen sich in ihrer Forschungsarbeit [54] mit Ansätzen zur Klassifizierung von Android-Malware auf Basis von Visualisierungen. Dabei stellen die Autoren zunächst verschiedene vorhergegangene Arbeiten zu dem Thema kurz vor. Diese haben alle gemeinsam, dass sie aus Malware ein oder mehrere Bilder generieren und diese dann mit verschiedenen Machine-Learning-Methoden kategorisieren. Einige dieser Ansätze konnten dabei eine Genauigkeit von 93% oder mehr erreichen [54].

Die von Marwaha et al. selbst angewandten Methoden werden auch in [54] vorgestellt, allerdings liegt hier der Fokus nicht auf der Erstellung der Visualiserung. Stattdessen beschreiben die Autoren ausführlich die angewendeten Machine-Learning-Methoden sowie die verschiedenen Kombinationen von Malware-Bestandteilen, welche für den Gesamtansatz als Input verwendet werden können.

Da die vorliegende Arbeit sich nicht mit Android-Malware im Speziellen beschäftigt und auch kein komplexer Machine-Learning-Ansatz entwickelt werden soll, wird im Folgenden nur kurz die von Marwaha et al. [54] verwendete Methodik zur Erstellung von Bildern beschrieben.

Analog zu Singh et al. [53] betrachten auch Marwaha et al. [54] die zu analysierende Datei auf Binärebene und bilden 8-Bit-Blöcke. Dabei werden immer zwei aufeinander folgende Blöcke zu einem Paket aus zweimal acht Bits zusammengefügt. Jedes dieser Pakete stellt die Informationen für ein Pixel im fertigen Bild. Die Autoren nutzen dabei die ersten 4 Bits um die Position des Pixels im fertigen Bild zu bestimmen, während die restlichen Bits den Farbwert des Pixels im RGB-Spektrum bestimmen [54]. Ein Beispielbild für eine visualisierte Malware-Datei wird von [54] nicht bereitgestellt und kann daher hier auch nicht gezeigt werden.

3 Vorüberlegungen

Die in Kapitel 2.4 Visualisierung von Binärdaten vorgestellten Methoden zur Klassifizierung oder Zuordnung von Datenbereichen zu einzelnen Datenstrukturen basiert immer auf der persönlichen Erfahrung des Auswertenden oder auf der Verwendung von neuronalen Netzwerken und maschinellem Lernen. Letztere fallen beide in den Bereich der Künstlichen Intelligenz (KI), deren Verwendung in der digitalen Forensik ungern gesehen wird. Wie bereits in Abschnitt 2.3 erwähnt, liegt eines der größten Gebote der IT-Forensik in der Nachvollziehbarkeit und Wiederholbarkeit von durchgeführten Untersuchungen und deren Ergebnissen. Dies wurde zunächst vom Bundesamt für Sicherheit in der Informationstechnik (BSI) im Leitfaden „IT-Forensik“ [35] beschrieben und auch in darauffolgenden Publikationen, zum Beispiel von Geschonneck [33] und Reibold [37], aufgenommen. Diese Nachvollziehbarkeit ist bei KI-Systemen in der Regel aber nicht gegeben [55]. Auch Solanke [56] geht darauf ein, dass es weiterhin große Bedenken bezüglich der Verwendbarkeit von durch KI-Systemen getroffenen Entscheidungen in der digitalen Forensik und Strafverfolgung im Allgemeinen gibt. Er gibt jedoch auch konkrete Ziele vor, die eine in der digitalen Forensik verwendete KI erfüllen sollte, um als xDFAI zu gelten und somit auch zukünftig akzeptiert zu werden. Der Begriff xDFAI wird dabei vom Englischen „Explainable Digital Forensics Artificial Intelligence“ abgeleitet und steht für eine „erklärbare Künstliche Intelligenz (KI) in der Digitalen Forensik“ [56]. Auch wenn aktuelle Verfahren zur Mustererkennung in Bildern bereits sehr genaue Ergebnisse liefern (vgl. [54]), fordert mit dem Europäischen Parlament auch der Gesetzgeber, dass KI in der Strafverfolgung nur unter bestimmten Bedingungen verwendet werden darf. Zu diesen Bedingungen zählen unter anderem auch die Nachvollziehbarkeit, Transparenz, Verifikation und Erklärbarkeit des Algorithmus [55, 57].

Da zudem davon auszugehen ist, dass nur vereinzelte Menschen ausreichend Expertise aufweisen, um Datenstrukturen direkt anhand der Visualisierung zu erkennen, muss also ein neuer Ansatz zur Klassifizierung der Daten gefunden werden. Logisch erscheint hier ein mathematischer Ansatz, wie der bereits von Domas verwendete Naive-Bayes-Klassifikator (siehe Abschnitt 2.4.1.3). Da die verschiedenen Visualisierungen auf Binärwerten, also auf Zahlen innerhalb einer statistischen Verteilung, basieren, sollte auch eine mathematische Bewertung dieser statistischen Verteilung möglich sein, ohne das Algorithmen zur Bildmustererkennung nötig sind. Auch wenn die tatsächliche Visualisierung bei diesem Ansatz nicht zwingend nötig ist, soll sie trotzdem erzeugt und angezeigt werden, um einen ersten Überblick über die einzelnen Methoden zu gewinnen. Zudem dient sie der Anschaulichkeit und Nachvollziehbarkeit der dazu durchgeführten Berechnungen.

Der Demonstrator, mit dem diese Überlegungen umgesetzt werden, sollte möglichst einfach zu installieren und zu bedienen sein. Ebenso sollte er auf den gängigen Computer-Betriebssystemen ausführbar sein, da in der IT-Forensik kein einheitliches Betriebssystem vorgegeben ist. Die Programmiersprache Python ist eine der meist verwendeten Sprachen für Datenanalyse, da sie bereits viele Bibliotheken mit passenden Funktionen bereithält [58]. Dies vereinfacht die Implementierung des zu entwickelnden Demonstrators stark. Darüber hinaus sind Python-Programme auf Windows, macOS und Linux ausführbar [59]. Sie erfüllen somit das Kriterium der Plattformunabhängigkeit. Zur Bedienung kommen sowohl eine Graphische Benutzeroberfläche (engl. Graphical User Interface, kurz GUI) als auch ein einfacheres Kommandozeilentool in Frage.

Wie in Abschnitt 2.3.1 dargelegt, werden Daten im Speicher als Blöcke oder Cluster geschrieben. Da Cluster in ihrer Größe Vielfache von Blöcken sind, welche der Länge eines Sektors entsprechen, erscheint eine sektorweise Analyse von Daten für die vorliegende Arbeit geeignet. Wird ein clusterbasiert geschriebener Datenausschnitt bewertet, wäre grundsätzlich eine Analyse der einzelnen Cluster sinnvoller, als die der einzelnen Sektoren. Es soll jedoch zunächst eine möglichst allgemeine Methode erarbeitet werden, welche auch dann verwendet werden kann, wenn noch unklar ist, ob die zu verarbeitenden Daten cluster- oder sektorbasiert entstanden sind. Die Größe eines Sektors von 512 Byte bildet dabei den größten gemeinsamen Teiler der Clustergröße von 4096 Byte und der Blockgröße, welche entsprechend der Sektorgröße auch 512 Byte beträgt. Darüber hinaus besteht die Möglichkeit, dass Binärdaten als reiner Stream vorliegen, zum Beispiel bei einer Satellitenübertragung oder einem Netzwerkmitschnitt. Hier gäbe es gar keine Cluster oder Blöcke, sodass auch hier eine möglichst kleinteilige Betrachtung unter Berücksichtigung des verwendeten Protokolls sinnvoll ist.

In der Theorie ist die Menge der verschiedenen Dateiformate unendlich groß. Dies liegt daran, dass einzelne Datenformate in unterschiedlichen Formen auftreten, abhängig von den jeweils gewählten Codierungs-, Kompressions- und Verschlüsselungsalgorithmen [36]. FILExt [60], das nach eigenen Angaben weltweit größte Verzeichnis für Dateiformate, führt derzeit über 26 000 verschiedene Dateiformate. Eine Erkennung dieser Menge an Dateitypen erscheint jedoch nicht umsetzbar. Tatsächlich wird in der Praxis meist nur ein Bruchteil dieser 26 000 Dateitypen regelmäßig verwendet. Daher erscheint es sinnvoll, dass sich die vorliegende Arbeit zunächst auf einige, tendenziell weiter verbreitete Dateitypen fokussiert. Die Auswahl dieser sollte dabei insbesondere beachten, dass es sich um Dateiformate handelt, welche von populären Programmen erstellt oder gelesen werden können [36]. Als Kandidaten fallen daher sofort die aktuellen Standard-Dateitypen[1] docx, pptx und xlsx für Dokumente der Microsoft-Office-Programme Word, Powerpoint und Excel sowie das Dokumenten-Format pdf auf. Da die vorliegende Arbeit auf den Vorteilen von visuellen Darstellungen aufbaut, sollen zudem ein paar Bildformate betrachtet werden. Hier wurden die Formate bmp, gif, jpg, png und svg sowie das Videoformat mp4 ausgewählt. Darüber hinaus werden die Dateiformate exe, elf und apk ausgewertet, da diese für ausführbare Dateien auf verschiedenen Betriebssystemen verwendet werden und einige der oben vorgestellten Ansätze speziell für die Auswertung ausführbarer Dateien entwickelt wurden. Zusätzlich wurden mit epub, txt und xml noch drei Dateitypen ausgewählt, die tendenziell sehr viele ASCII-Zeichen beinhalten. Eine Klassifikation nach Datenstrukturen wäre ungenauer als die nach Dateitypen und ließe sich durch das Zusammenfassen mehrere Dateitypen zu einer Kategorie erreichen. Anhand der eben ausgewählten Dateitypen würden sich beispielsweise die Kategorien Dokument, Bild, ausführbare Datei und Text eignen.

Der zu entwickelnde Demonstrator sollte also in der Lage sein, einem Nutzer die ausgewählten Visualisierungsmethoden graphisch darzustellen. Zudem ist eine Funktion für die automatisierte Erkennung und Klassifizierung von Dateien auf Basis der enthaltenen Sektoren vorzusehen. Zur Überprüfung der Qualität der Klassifizierung sollte der Demonstrator selbstständig Performancemaße berechnen, dem Nutzer anzeigen und für die spätere Verwendung abspeichern.

[1] Bei den folgenden Dateiformaten handelt es sich um Standards, welche typischerweise groß geschrieben werden. In der vorliegenden Arbeit wird aufgrund der besseren Lesbarkeit jedoch die kleingeschriebene Variante bevorzugt, wie sie auch für Dateiendungen verwendet wird.

4 Versuchsaufbau

Um eine Nachvollziehbarkeit und gegebenenfalls Wiederholbarkeit der in den folgenden Abschnitten beschriebenen Methoden und Ergebnisse zu gewährleisten, werden in diesem Kapitel die Entwicklungs- und Versuchsumgebung sowie das verwendete Datenset vorgestellt.

4.1 Versuchsumgebung

Alle im Folgenden benötigten Berechnungen und Funktionen wurden mit der Programmiersprache Python in der zum Bearbeitungszeitraum der vorliegenden Arbeit aktuellen Version 3.11 (Stable Release) implementiert und ausgeführt. Die Erstellung der Skripte erfolgte in der Entwicklungsumgebung PyCharm 2023.1.1 (Community Edition) auf einem MacBook Pro (Retina, 13”, Mitte 2014) mit Betriebssystem macOS Big Sur, Version 11.7.6.

Aufgrund der besseren Performance wurde die Entwicklung im Verlauf der Arbeit auf einen PC mit Intel i9 3.20 GHz-Prozessor und 128 Gigabyte (GB) RAM migriert. Auf diesem sind das Betriebssystem Windows 10 Pro sowie PyCharm 2023.2 (Community Edition) installiert. Auf diesem Rechner wurden nach Möglichkeit dieselben Python-Pakete genutzt, wie zuvor auf dem Mac. Die verwendeten Pakete werden in Tabelle 4-1 mit ihrer Versionsnummer und Funktion gelistet. Sollte eine der verwendeten Funktionen in Zukunft aus den Paketen entfernt werden, so dient die Tabelle als Referenz für Paketversionen, mit denen die im weiteren Verlauf der Arbeit beschriebenen Methoden ausführbar sind. Mit * markierte Pakete sind in der Python-Standard-Installation enthalten, weshalb die Paketversion nicht separat aufgeführt wird. Zusätzlich sind in der Spalte „Dok“ die zugehörigen Dokumentationen, die zur Implementierung konsultiert wurden, verlinkt.

Tabelle 4-1: Verwendete Python-Pakete und -Module

Paket-/Modulname	Version	Dok	Funktion
colorsys	*	[61]	Umrechnung HSL-Werte ins RGB-Format
matplotlib	3.8.0	[62]	Bereitstellung von Colormaps
matplotlib.pyplot	3.8.0	[62]	Graphische Darstellung der vorgestellten Konzepte
numpy	1.24.3	[63]	Verwendung von numpy-Arrays
numpy-hilbert-curve	1.0.1	[11]	Umformungen entsprechend der Hilbertkurve
os	*	[61]	Ermittlung und Überprüfung von Dateipfaden
Pillow.Image	9.5.0	[64]	Erstellung und Anzeige von HSL-Bildern
scikit-learn	1.2.2	[65]	Erstellung und Nutzung Naive-Bayes-Klassifikator
scipy.ndimage	1.10.1	[66]	Erkennung von Regionen

4.2 Versuchsdaten

Um Forschungsarbeiten möglichst wiederholbar und nachvollziehbar zu gestalten, sollten die verwendeten Datensätze öffentlich zugänglich sein. Zusätzlich ermöglicht dies Anderen, die Ergebnisse der Forschungsarbeit zu verifizieren. Die vorliegende Arbeit verwendet daher Dateien der Sammlung

NapierOne [67] zum Bearbeiten der Fragestellung. Diese Sammlung wurde ausgewählt, da sie eine breite Menge verschiedener Dateitypen abdeckt [67]. Zudem handelt es sich bei den enthaltenen Dateien um Echt-Dateien, das heißt diese wurden nicht erst zu Forschungszwecken synthetisch erzeugt [67].

Die NapierOne-Sammlung bietet Datei-Zusammenstellungen verschiedener Größen an [67]. Die gesamte Sammlung umfasst knapp 2 Terabyte (TB) Daten, was zwar eine große Menge unterschiedlicher Dateien verspricht, für die vorliegende Arbeit aber nicht gut geeignet ist, da die Erprobung neuer Methoden zunächst nur mit geringerer Datenmenge erfolgen soll. Daher wurde nur das Teilset NapierOne-Tiny aus der Sammlung ausgewählt und heruntergeladen. Dieses besteht aus 100 Dateiklassen zu je 100 Dateien und bietet damit immer noch eine ausreichend große Grundlage für die Entwicklung erster Methoden.

Werden im weiteren Textverlauf Dateien aus der NapierOne-Sammlung verwendet, so sind sie entsprechend ihrem Dateinamen gekennzeichnet. Dies erlaubt eine spätere Zuordnung zur jeweilig verwendeten Datei, da diese je Dateityp durchnummeriert und vom Autor [67] eindeutig benannt wurden. Dabei wurde sich auf die in Kapitel 3 festgelegten Dateitypen beschränkt. Einige dieser Typen sind in mehrere Klassen mit unterschiedlicher Qualität oder Verschlüsselungsart aufgeteilt. Bei den Bildformaten wurden die Sammlungen bmp, gif, jpg-q050, png-from-web und svg-from-web auagewählt. Bei allen anderen Formaten wurde jeweils die nur mit dem Dateityp benannte Sammlung ausgewählt, um möglichst unverschlüsselte Dateien zu erhalten.

5 Methoden

Zur praktischen Umsetzung der vorliegenden Arbeit mussten die in Kapitel 2.4 vorgestellten Ideen auf Sektoren angepasst werden. Dafür wurden verschiedene Funktionen in Python geschrieben, welche im Folgenden näher beschrieben werden. Die in diesem Kapitel enthaltenen Abschnitte unterteilen sich in die Arbeitsschritte Einlesen der Daten (5.1), Aufbereitung der Binärdaten (5.2), Visuelle Bewertung (5.3) und Statistische Bewertung (5.4).

5.1 Einlesen der Daten

Da die Länge einer einzulesenden Datei nicht vorhersehbar ist, ist es sinnvoll, die Funktion zum Einlesen und ersten Verarbeiten der Datei in Form einer Generatorfunktion zu programmieren. Diese hat den Vorteil, dass die eingelesenen Blöcke nicht direkt in den Speicher des ausführenden Computers geladen werden [68]. So wird verhindert, dass der Arbeitsspeicher des ausführenden Computers überläuft.

Quelltext 5.1 zeigt die Funktion `read_blocks()`, mit der eine Datei blockweise eingelesen werden kann. Um die Datei nicht versehentlich zu verändern, wird sie nur lesend geöffnet, erkennbar am `r` in den Argumenten der Funktion `open()`. Das `b` sorgt dafür, dass die Datei im Binär-Modus geöffnet wird. Dies ist essenziell, da alle folgenden Methoden die Dateien auf Byte-Ebene verarbeiten.

Wie oben bereits erwähnt, ist eine blockweise Verarbeitung der Dateien sinnvoll. Daher wird der Funktion `read()` der Wert `0x200` als Parameter übergeben, was mit 512_d Bytes der Länge eines Blockes, analog der Sektorgröße, entspricht. Je nach weiterer Verwendung, kann der eingelesene Block gleich innerhalb der Funktion verarbeitet werden. Dafür müssen einfach die entsprechenden Befehle vor dem Aufruf von `yield` eingefügt und ggf. die zurückgegebene Variable angepasst werden.

```
1  def read_blocks(filename):
2      with open(filename, "rb") as rf:          # open file as read-only,
3                                                # binary-mode
4          maxpos = os.stat(filename).st_size    # get file size
5          pos = 0                               # marks position in rf
6          while True:
7              if pos >= maxpos:                 # break when reaching end of file
8                  break
9
10             fdata = rf.read(0x200)            # read one block
11             pos += len(fdata)
12             yield fdata
```

Quelltext 5.1: Einlesen einer Datei mittels Generator

Um die von `read_blocks()` zurückgegebenen Werte nun zu verarbeiten, muss über den Generator iteriert werden [68]. Dies geht zum Beispiel durch die Verwendung einer `for`-Schleife, wie in Quelltext 5.2 gezeigt. Die innerhalb der Schleife aufgerufenen Befehle und Funktionen werden dann für jedes Element des Generators ausgeführt.

```
1 blockGenerator = read_blocks(filepath)
2 for block in blockGenerator:
3     # do something with each block
```

Quelltext 5.2: Verwendung des Generators

5.2 Aufbereitung der Binärdaten

Die Anwendung bereits bewährter Methoden kann bei der Entwicklung neuer Tools von Vorteil sein, da nicht alles neu entwickelt werden muss [42]. Um die eingelesenen Sektoren nun zu analysieren, soll zunächst die Aufbereitung in Form eines Bildes und eines Digrammes entsprechend der in den Abschnitten 2.4.3.1 und 2.4.1.1 vorgestellten Ideen und Methoden erfolgen. Zudem wird auf Basis der im Abschnitt 2.4.1.3 vorgestellten Arbeiten ein Trigramm erstellt. Eine Umsetzung von n-Grammen mit $n > 3$ erfolgte vorerst nicht. Anschließend werden in diesem Abschnitt weitere Funktionen und Ideen besprochen, die im weiteren Verlauf erarbeitet wurden.

5.2.1 Bild

Zunächst soll entsprechend der unter 2.4.3 vorgestellten Methoden zur Malware-Klassifizierung ein Bild erstellt werden. Dabei wird je ein Byte der Datei als ein Pixel dargestellt.

Für die Umsetzung dieser Methodik im Rahmen der vorliegenden Arbeit wurde mit leichten Abweichungen entsprechend der von Singh et al. [53] beschriebenen Vorgehensweise gearbeitet. Die Abweichungen entstehen durch die Verwendung der Programmiersprache Python und die zuvor getroffene Entscheidung, Dateien blockweise einzulesen. Im Folgenden wird der Verlauf vom Einlesen der Datei bis zur Erzeugung eines Bildes genauer beschrieben.

Wie oben bereits erwähnt, erfordert die Vorgehensweise von Singh et al. [53] eine byteweise Verarbeitung der Daten. Daher erfolgt die Berechnung von Farbwerten für das zu erzeugende Bild direkt in der Funktion `read_hsl_pixels()`, welche in ihrer Form an die `read_blocks()`-Funktion aus 5.1 angelehnt ist. Hier erfolgt die Bearbeitung der gelesenen Daten also vor Aufruf des `yield`-Befehls.

Im ersten Schritt wird jedes Byte einzeln in zwei Teile aus je 4 Bits, sogenannte Nibbles [69], geteilt. Diese werden von Singh et al. [53] zur Bestimmung eines Farbwertes anhand einer zweidimensionalen Colormap verwendet. Mit diesem Farbwert wird ein Pixel eingefärbt, welches dem eingelesenen Byte entspricht. In der beschriebenen Vorgehensweise werden die nacheinander bestimmten Pixel dann sequentiell in ein Bild mit einer festgelegten Breite von 384 Pixeln eingefügt und das Bild schließlich angezeigt.

Im Rahmen der vorliegenden Arbeit konnte nicht auf eine bestehende Colormap zurückgegriffen werden, da die für die vorliegende Arbeit verwendete Programmiersprache Python nur eindimensionale Colormaps unterstützt. Um trotzdem eine zweidimensionale Zuordnung von Farbwerten zu erreichen, wurde auf das HSL-Schema zurückgegriffen. Wird die Sättigung mit $S = const. = 1$ festgelegt, so lassen sich über die übrigen beiden Werten einem Byte eine Vielzahl von Farben zuordnen. Die Verwendung des RGB-Farbmodells ist hier weniger gut geeignet, da bei einem konstanten Wert

mit den beiden verbleibenden Farben deutlich weniger verschiedene Farbtöne dargestellt werden können. Würde beispielsweise B als konstanter Wert $B = const. = 255$ gewählt werden, so liessen sich mit R und G nur die Farben auf der oberen Seite des Würfels aus Abbildung 2-6 darstellen.

Alternativ könnte der Ansatz von Marwaha et al. [54] verwendet werden. Dabei stehen jeweils drei Werte aus je 4 Bits zur Bestimmung eines Farbtons zur Verfügung. In der Quelle ist die Bestimmung der Position des Pixels über die ersten vier Bit in einem 16-Bit-Paket jedoch nicht eindeutig beschrieben, weshalb dieser Ansatz in der vorliegenden Arbeit nicht weiter verfolgt wird.

```
1 floats = np.linspace(0, 1, 18)[1:-1]
```

Quelltext 5.3: Float-Array zur Definition der HSL-Werte

Jedes Nibble kann einen Wert zwischen 0 und 15 annehmen. Zur Darstellung der Farben in Python wird jedoch eine Skala von 0 bis 1 verwendet. Um die Werte der Nibbles an diese Skala anzupassen, wurde mit `floats` ein Array erzeugt. Dieses wurde mit der Funktion `np.linspace()` mit 18 Werten von 0 bis 1 gefüllt. Zusätzlich wurden das erste und letzte Element mit den Werten 0 respektive 1 entfernt, wie in Quelltext 5.3 gezeigt. Dies soll verhindern, dass alle Pixel mit einem Helligkeitswert von 0 beziehungsweise 1 unabhängig vom Farbwert gleichmäßig schwarz beziehungsweise weiß dargestellt werden. Anhand des Indexes lässt sich dann für jeden Nibble der korrespondierende Gleitkommawert abfragen.

Um dies zu verdeutlichen wurde auf Basis der Werte in `floats` eine 16x16 Pixel große Colormap erzeugt (siehe Abbildung 5-1). Diese zeigt alle 256 Farben, durch die ein Byte repräsentiert werden kann. Dabei ist oben links das Pixel für Bytes mit dem Wert `00`, während unten rechts die Repräsentation für `FF` zu erkennen ist.

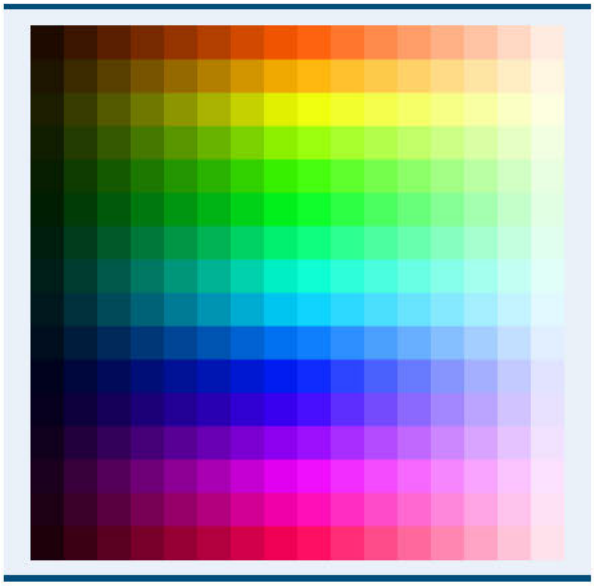


Abbildung 5-1: Verwendete Colormap, eigene Darstellung

Eine weitere Anpassung erfolgte bei der Breite des zu erzeugenden Bildes. Da bei hiesiger blockweisen Vorgehensweise immer Abschnitte von 512 Bytes eingelesen werden, erschien eine Bildbreite von 384 Pixeln bzw. Bytes unpraktisch. Stattdessen wurde die Bildbreite auf 256 Pixel gesetzt, sodass ein eingelesener Block zwei Zeilen im Gesamtbild ausmacht. Diese Breite ist vorteilhaft, da sie

nicht nur ein Teiler von 512 ist, sondern auch eine komprimiertere Ansicht erlaubt als eine einfach Zeile von 512 Pixel/Byte Länge. Die fertige Funktion `read_hsl_pixels()` (siehe Quelltext 5.4) gibt also Arrays der Größe `[2,256,3]` zurück, aus welchen in der Folge Bilder erzeugt werden können.

```
1  def read_hsl_pixels(filename):
2   with open(filename, "rb") as rf:
3     maxpos = os.stat(filename).st_size
4     pos = 0
5     while True:
6       if pos >= maxpos:          # break when reaching end of file
7       break
8
9       fdata = rf.read(0x200)     # read one block
10      pos += len(fdata)
11
12      sat = 1                    # saturation = const = 1
13
14      blockpixels = np.zeros([2, 256, 3])   # create array for pixels of
15                                 # current block; 2 & 256 so each block
16                                 # makes up 2 rows of final pic, each of
17                                 # them being 256 pixels wide
18      bytecounter = 0
19
20      for currentByte in fdata: # for each byte (cast to int)
21        hue, light = currentByte >> 4, currentByte & 0x0F
22                                 # split currentbyte
23                                 # into nibbles
24        hue = floats[hue]        # convert to float between 0 and 1
25        light = floats[light]    # using floats array
26
27        # add hsl-pixel-values to blockpixels
28        if bytecounter <= 255:
29        blockpixels[0][bytecounter] = [hue, light, sat]
30        else:
31        blockpixels[1][bytecounter-256] = [hue, light, sat]
32
33        bytecounter += 1
34
35      yield blockpixels
```

Quelltext 5.4: Erzeugung Pixelwerte im HSL-Format

5.2.2 Digramm

Zur Durchführung der Digramm-Analyse wurde zunächst die Funktion `create_digram_freq_map()` erstellt, welche eine Datei einlesen und für jeden enthaltenen Sektor eine Häufigkeitsmatrix für die enthaltenen Digramme füllt. Dafür wird wie unter Abschnitt 5.1 beschrieben, über die mittels `read_blocks()` eingelesenen Sektoren iteriert. Für jeden Sektor, wird eine neues Numpy-Array `freq_map` der Größe `[256, 256]` erstellt, welche als Häufigkeitsmatrix dient. Beginnend mit Zeile 8 in Quelltext 5.5, werden die Digramme Stück für Stück ausgelesen. Entsprechend der in Kapitel 2.4.1.1 beschriebenen Vorgehensweise, werden die beiden Werte des Digrammes als Index für die Referenzierung eines Wertes im Array genutzt, welcher um den Wert eins erhöht wird.

```
1  def create_digram_freq_map(filename):
2      for block in (read_blocks(filename)):
3          freqMap = np.zeros([256,256])        # create variables needed
4          size = len(block)
5          min = 0xffffffff
6          max = 0
7
8          for i in range(0, size - 1):         # read tuples from block
9              x = block[i]
10             y = block[i + 1]
11             freqMap[y][x] += 1               # add one at coordinates of tuple
12             if freqMap[y][x] > max:          # update max if needed
13                 max = freqMap[y][x]
14
15         for j in range(0, 256):              # determine min
16             for i in range(0, 256):
17                 if freqMap[j][i] < min:
18                     min = freqMap[j][i]
19
20         yield [freqMap, min, max, size]
```

Quelltext 5.5: Erstellung Häufigkeitsmatrix aus Digrammen

Ist der eingelesene Sektor kleiner als 512 Byte, weil er zum Beispiel vom Ende einer Datei stammt, so ist hier keine zusätzliche Fehlerbehandlung erforderlich. In diesem Fall werden einfach weniger Werte in die `freqMap` eingetragen. Die Variable `size` wird verwendet, um die Länge des jeweils eingelesenen Sektors zu bestimmen und so zu gewährleisten, dass die Bestimmung der Digramme rechtzeitig abbricht.

Innerhalb der Funktion werden zudem der kleinste und der größte Wert jeder Häufigkeitsmatrix bestimmt. Zu Beginn wird die Variable `max = 0` gesetzt. Während der Erstellung der Häufigkeitsmatrix, wird fortlaufend überprüft, ob der neu erhöhte Wert nun höher ist, als der bisherige Wert von `max`. Ist dies der Fall, so wird `max` dieser Wert zugewiesen.

Die Variable `min` wird zu Beginn auf `0xffffffff` gesetzt. Nachdem der Sektor komplett verarbeitet wurde und die Häufigkeitsmatrix fertiggestellt ist, wird noch einmal über das gesamte Array `freqMap` iteriert (siehe Zeilen 15-18, Quelltext 5.5). Dabei wird überprüft, ob der jeweils betrachtete Wert im Array kleiner ist als der aktuelle Wert von `min`. Bei Bedarf wird `min` entsprechend aktualisiert.

Anschließend werden die Häufigkeitsmatrix `freqMap` sowie die Variablen `min`, `max` und `size` an einen Generator übergeben. Eine graphische Darstellung der Matrix kann nun anschließend über die Funktion `imshow()` der Bibliothek `matplotlib.pyplot` darstellen.

5.2.3 Trigramm

Die Erstellung einer Häufigkeitsmatrix auf Basis von Trigrammen erfolgte analog zur Digramm-Analyse. Wie in Quelltext 5.6 erkennbar, wurde dafür dem Array `freqMap` eine weitere Dimension hinzugefügt, sodass es nun die Größe `[256,256,256]` hat.

```
1  def create_trigram_freq_map(filename, hilbert):
2   for block in (test.read_blocks(filename)):
3     freqMap = np.zeros([256,256,256])      # create variables needed
4     size = len(block)
5     min = 0xffffffff
6     max = 0
7
8     if hilbert:
9       block = data_manipulation.block_to_hilbert_order(block)
10
11    for i in range(0, size - 2):           # read tuples from block
12      x = block[i]
13      y = block[i + 1]
14      z = block[i + 2]
15      freqMap[z][y][x] += 1                # add one at coordinates of tuple
16      if freqMap[z][y][x] > max:           # update max if needed
17        max = freqMap[z][y][x]
18
19    for j in range(0, 256):                # determine min
20      for i in range(0, 256):
21        for h in range(0, 256):
22          debug = freqMap[j][i][h]
23          if debug < min:
24            min = freqMap[j][i][h]
25
26    yield [freqMap, min, max, size]
27
```

Quelltext 5.6: Erstellung Häufigkeitsmatrix aus Trigrammen

5.2.4 Zusammenhängende Regionen

Betrachtet man die Häufigkeitsmatrizen für Digramme und Trigramme rein visuell anhand einzelner Beispiele, so fällt auf, dass an einigen Stellen im Array deutlich höhere Werte auftreten, als im Rest des Arrays. Ein Ansatz diese Bereiche zu identifizieren und gleichzeitig eine Streuung von Werten, die nicht zu den primär relevanten Bereichen zählen, zu ignorieren, liegt in der Bestimmung des Mittelwerts für das gesamte Array. Im Folgenden werden nur noch diejenigen Werte betrachten, die oberhalb des Mittelwerts liegen. Ziel dieses Vorgehens ist es, den potentiellen Rechenaufwand für die Erstellung und den Vergleich der Häufigkeitsmatrizen zu reduzieren sowie besonders markante Muster für die einzelnen Dateitypen herauszufiltern.

Da einzelne Stellen mit besonders hohen Werten im Array durchaus relevant sind, wird für die Berechnung des Mittelwerts auf das arithmetische Mittel zurückgegriffen. Dieses ist in Python durch die Funktion `mean()` implementiert. Die Digramm- und Trigramm-Verteilungsmatrizen enthalten jedoch meist auch viele Nullen. Das heißt, wenn die Funktion auf das gesamte Array angewendet wird, bleibt der Mittelwert meist im Bereich zwischen null und eins. Um dies zu umgehen, werden alle Nullen im Array `freqMap` auf `nan`, kurz für „Not a Number (nan)“ gesetzt. Zusätzlich wird die Funktion `mean()` durch `nanmean()` ersetzt, da diese alle `nan`-Werte im Array ignoriert, sodass die Nullen den Mittelwert nicht länger beeinflussen. Anschließend müssen nur noch alle Werte, die kleiner als der berechnete Mittelwert sind, aus dem Array entfernt werden. Dieses Vorgehen ist in Quelltext 5.7 in den Zeilen 2 bis 8 umgesetzt.

```
1  def find_regions(freqMap):
2    # calculate mean ignoring zeros
3    nanmap = np.copy(freqMap)
4    nanmap[nanmap == 0] = np.nan
5    av = np.nanmean(nanmap)
6    # remove values that are below average
7    avmap = np.copy(freqMap)
8    avmap = avmap > av
9
10   # label connected regions in map
11   s = [[1, 1, 1],
12        [1, 1, 1],
13        [1, 1, 1]]
14   labels, num_labels = ndimage.label(freqMap, structure=s)
15
16   # plot
17   fig, ax = plt.subplots(nrows=1, ncols=2, sharex='all', sharey='all',
18                          figsize=(10, 5))
19                          # create figure with two axes (row 1,
20                          # columns 2), all share x and y axis
21
22   ax[0].imshow(nanmap, cmap=mpl.colormaps['viridis'])
23                          # ax[0] plots values in freqMap
24   ax[1].imshow(np.ma.masked_array(labels, ~avmap),
25                cmap=mpl.colormaps['rainbow'])
26                          # ax[1] plots labeled regions
27
28   fig.tight_layout()     # minimize white border around figure
29   plt.show()
30
```

Quelltext 5.7: Identifikation von Regionen in einer Häufigkeitsmatrix

Python bietet darüber hinaus in der Bibliothek `scipy.ndimage` die Funktion `label()` [70], welche Regionen zusammenhängender Felder in einem Array identifizieren kann. Damit könnte ein Programm gegebenenfalls in der Lage sein, charakteristische Muster automisiert zu identifizieren. Diese wird im Anschluss auf die reduzierte Häufigkeitsmatrix angewendet. Wie in den Zeilen 10-14 von Quelltext 5.7 erkennbar, muss dafür zunächst die Struktur angepasst werden, anhand derer die Regionen gelabelt werden. Standardmäßig markiert `label()` einzelne Felder in einem Array nur dann als zusammenhängend, wenn diese in der graphischen Darstellung eine Kante teilen. Felder, die ausschließlich diagonal über eine Ecke verbundenen sind, werden dabei nicht als zusammenhängende Regionen erkannt. Um eine allzu kleinteilige Aufteilung in Regionen zu verhindern, wurde daher über die Variable `s` der Parameter `structure` angepasst, um auch diagonal verbundene Felder zu einer Gruppe zusammenzufassen.

5.2.5 Umformung mit Hilbertkurve

Cortesis Arbeit konzentrierte sich vor allem auf die Hilbertkurve und die Vorteile, die diese der visuellen Binäranalyse bringt (siehe Abschnitt 2.4.1.2). Um diese nun mit Contis n-Gramm-Analyse zu verbinden, sollen die Funktionen `create_digram_freq_map(filename)` und `create_trigram_freq_map(filename)` so angepasst werden, dass die eingelesenen Blöcke nicht zeilenweise sondern der Hilbertkurve folgend eingelesen werden. Dafür erhalten die beiden genannten Funktionen

zunächst noch den Eingabeparameter `hilbert` (Dateityp `Boolean`) und es wird die in Quelltext 5.8 gezeigte Funktion `block_to_hilbert_order(block)` implementiert, welche in Abhängigkeit des Wertes von `hilbert` vor der Erstellung der Häufigkeitsmatrix ausgeführt wird.

```
1  def block_to_hilbert_order(block):
2      array = np.zeros([32,16])       # 32 rows, 16 columns -> like in a
       Hexeditor
3    for i in range(32):
4      for j in range(16):
5        index = i*16+j
6        array[i][j] = block[index]
7    hilbertlist = hilbert_flatten(array).astype(int)
8    return hilbertlist
9
10 def hilbert_flatten(array):
11   D = array.ndim
12   S = np.arange(np.array(array.shape).prod())
13   L = hc.decode(S, D, 8).T.tolist()
14   return array[tuple(L)]
```

Quelltext 5.8: Umformung eines Blocks auf Basis der Hilbertkurve, `hilbert_flatten()` aus [71]

Da die Funktion `block_to_hilbert_order(block)` den eingelesenen Block als eindimensionales `Bytes`-Objekt erhält, werden die eingelesenen Werte zunächst in ein zweidimensionales Numpy-Array überführt. Analog zu bestehenden Hex-Editoren und auch Cortesis binvis.io, wird für das Array eine Breite von 16 Spalten gewählt. Bei einer Blocklänge von 512 Bytes ergibt sich daher eine Array-Größe von `[32,16]`. Im Anschluss wird die Funktion `hilbert_flatten(array)` [71] verwendet, um das Array entlang der Hilberkurve wieder in ein eindimendionales Array der Größe `[512]` zu überführen.

Die aktualisierte Funktion `create_digram_freq_map(filename)` ist im folgenden Quelltext 5.9 dargestellt. In der Funktion `create_trigram_freq_map(filename)` wurden der Eingabeparameter in Zeile 1 und die Verzweigung in den Zeilen 8-9 analog eingefügt.

```
1  def create_digram_freq_map(filename, hilbert):
2    for block in (test.read_blocks(filename)):
3        freqMap = np.zeros([256,256])    # create variables needed
4        size = len(block)
5        min = 0xffffffff
6        max = 0
7
8        if hilbert:
9            block = data_manipulation.block_to_hilbert_order(block)
10
11       for i in range(0, size - 1):     # read tuples from block
12           x = block[i]
13           y = block[i + 1]
14           freqMap[y][x] += 1           # add one at coordinates of tuple
15           if freqMap[y][x] > max:      # update max if needed
16               max = freqMap[y][x]
17
```

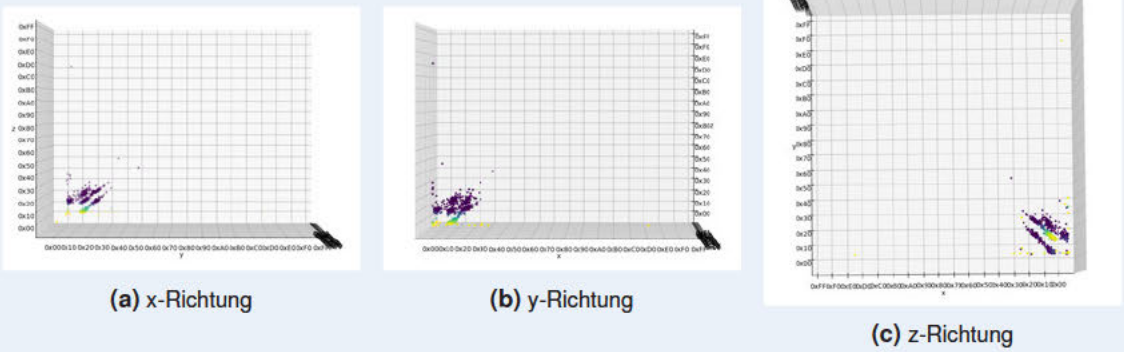
```
18
19          for j in range(0, 256):          # determine min
20              for i in range(0, 256):
21                  if freqMap[j][i] < min:
22                      min = freqMap[j][i]
23
24          (*@\textcolor{orange}{yield}@*) [freqMap, min, max, size]
```

Quelltext 5.9: Erstellung Häufigkeitsmatrix aus Digrammen mit Option zur Umformung

Auch für die Bild-Methode wurde ein Aufruf der oben vorgestellten Funktion `block_to_hilbert_order(block)` in die entsprechende Funktion eingesetzt. Durch die Umformung mit der Hilbertkurve, entspricht diese Methode nun nicht mehr dem Vorgehen von Singh et al. [53], sondern wirkt mehr an Cortesis Ideen [44] angelehnt

5.2.6 Überführung Trigramm-Analyse ins Zweidimensionale

Die Visualisierung von Trigrammen erlaubt es dem Betrachter, das Datenmodell von allen Seiten zu betrachten. Durch die verschiedenen Blickwinkel können unterschiedliche Muster erkannt und verglichen werden, wie in Abbildung 5-2 am Beispiel einer bmp-Datei dargestellt. Während in den Abbildungen 5-2a und 5-2c relativ gut zu erkennen ist, dass es sich in der unteren Ecke um drei längliche Punkteansammlungen handelt, wirken diese Punkte in Abbildung 5-2b mehr wie eine einzelne, wenn auch eher diffuse Gruppe. Auch die separat auftretenden Punkte sind bei allen drei Perspektiven unterschiedlich angeordnet und fallen dadurch teilweise mehr und teilweise weniger ins Auge.



(a) x-Richtung **(b)** y-Richtung **(c)** z-Richtung

Abbildung 5-2: Trigramm-Visualisierung einer bmp-Datei, eigene Darstellung

Um diese Betrachtung aus verschiedenen Perspektiven auch bei der statistischen Bewertung zu ermöglichen, wurde versucht, die verschiedenen Betrachtungsmöglichkeiten der dreidimensionalen Häufigkeitsmatrix aus Abschnitt 5.2.3 in einem zweidimensionalen Array darzustellen. Dies sollte möglichst so umgesetzt werden, als würde ein Betrachter der Trigramm-Visualisierung frontal auf den Würfel schauen. Dies erlaubt sechs verschiedene Perspektiven, wobei die jeweils gegenüber liegenden Blickwinkel die selben Ergebnisse bringen sollten, wenn auch gespiegelt. Daher wird die Funktion aus drei Richtungen implementiert.

```
def sum_slices(cube, dir='z'):
  sum = np.zeros([256,256])
  match dir:
  case 'x':
    for x in range(256):
      for y in range(256):
        for z in range(256):
          sum[x][y] += cube[z][y][x]
  case 'y':
    for x in range(256):
      for y in range(256):
        for z in range(256):
          sum[x][y] += cube[z][x][y]
  case 'z':
    for x in range(256):
      for y in range(256):
        for z in range(256):
          sum[x][y] += cube[x][y][z]
  case _:
    for x in range(256):
      for y in range(256):
        for z in range(256):
          sum[x][y] += cube[x][y][z]
  return sum
```

Quelltext 5.10: Erstellung einer zweidimensionalen Darstellung der Trigramm-Analyse

Quelltext 5.10 zeigt die fertige Funktion `sum_slices`. Darin wird über den Parameter `dir` die Richtung festgelegt, aus der ein Betrachter auf die übergebene Häufigkeitsmatrix gucken würde. Als Standardwert wurde `dir = 'z'` festgelegt, da dies der Perspektive „von vorne“ entsprechen würde.

Nach einigen Tests fiel auf, dass die Anwendung der Funktion auf eine dreidimensionale Häufigkeitsmatrix mit den Parametern `'x'` und `'z'` ein zweidimensionales Array ergibt, welches keine signifikanten Unterschiede zu der Häufigkeitsmatrix aus einer Digramm-Analyse aufweist. Damit geht die Dreidimensionalität des Trigramm-Ansatzes verloren. Wird die Funktion mit `dir = 'y'` aufgerufen, so lässt sich das Ergebnis mit einer Digramm-Analyse vergleichen, bei dem ein Byte nicht mit dem unmittelbar darauffolgenden Byte, sondern mit dem übernächsten in Relation gesetzt wird. Dies entspricht nicht mehr Contis These, dass aufeinanderfolgende Bytes zueinander in Relation stehen (siehe Abschnitt 2.4.1.1). Der Ansatz wird daher nicht weiter verfolgt.

5.3 Visuelle Bewertung

Um den Nutzen der einzelnen oben vorgestellten Methoden zu überprüfen, müssen diese nun auf verschiedene Dateien angewendet und bewertet werden. So soll festgestellt werden, ob eine gewählte Methode tatsächlich geeignet ist, Unterschiede zwischen einzelnen Dateitypen zu erkennen. Da alle aufgeführten Methoden auf Visualisierungstechniken basieren, soll diese erste Bewertung auch rein visuell erfolgen.

5.3.1 Vorbereitung

Jede der Methoden wird für sich einzeln bewertet. Dafür werden einzelne Dateien aus dem unter Kapitel 4.2 Versuchsdaten vorgestellten Datenset mit der jeweiligen Methode verarbeitet und als Bilddatei gespeichert. Diese konnten anschließend nebeneinander beziehungsweise nacheinander angesehen und miteinander verglichen werden.

Da bereits zur Bewertung einer einzelnen Methode zahlreiche Bilder erstellt werden müssten, wurde die Datenmenge für die visuelle Bewertung stark eingeschränkt. So wurde pro Klasse nur eine Datei betrachtet. Dabei fiel die Wahl auf die jeweilige Datei mit dem Titel `0003-<Dateityp>.<Dateityp>`. Ausnahmen davon bilden die Dateien `0001-xml.xml` und `0002-svg-from-web.svg`, da die jeweils dritte Datei der Sammlung bei den beiden Dateitypen xml und svg zu klein ist, um ausreichend Sektoren zu visualisieren. Um den Aufwand bei der Erstellung und Betrachtung der Bilder weiter zu begrenzen, wurden je Datei nur die ersten sechs Sektoren verarbeitet und betrachtet.

Die entstandenen Visualisierungen für alle Methoden ohne Umformung entlang der Hilbertkurve befinden sich im Anhang der vorliegenden Arbeit und können dort parallel zum Text betrachtet werden. Zusätzlich befinden sich auf dem beigelegten Datenträger Bilddateien zur großformatigen Ansicht. Diese sind für alle bewerteten Methoden in entsprechend gekennzeichneten Ordnern im Ordner „Visuelle_Bewertung“ abgelegt worden.

Bei den Bildern der Digramm- und Trigramm-Analyse ist es möglich, dass die gelb dargestellten Punkte in der gedruckten Ansicht nur schwer zu erkennen sind. Dies liegt daran, dass die Bilder im Anhang der vorliegenden Arbeit zunächst nur einen schnellen Überblick geben sollen. Für eine genauere Betrachtung wird die digitale Version der Bilder empfohlen. Zudem ist die dreidimensionale Ansicht bei der Visualisierung der Trigramm-Analyse im Bildformat nur begrenzt möglich. Hier wird empfohlen, nach Möglichkeit, mit dem in Kapitel 6 beschriebenen Demonstrator dynamische Visualisierungen zu erzeugen und dort zu betrachten. Die zur Erzeugung der im Anhang befindlichen Bilder genutzten Dateien sind auf dem beiliegenden Datenträger im Ordner „Visualisierungsdateien“ innerhalb des Ordners „Visuelle_Bewertung“ abgelegt.

5.3.2 Umsetzung

Aufgrund der Verschiedenheit der Visualisierungsmethoden, wurde jede Methode einzeln betrachtet. Dabei wurde nach erkennbaren Mustern in den einzelnen Darstellungen gesucht, welche charakteristisch für einen Dateityp scheinen. Dabei wurde nicht nur darauf Wert gelegt, dass die Sektoren einer Datei gleichartige Muster ergeben, sondern auch, dass diese sich von denen anderer Dateien unterscheiden. Zudem wurde bei der Bewertung davon ausgegangen, dass kein Vorwissen über den Header oder andere Informationen zum Aufbau einer Datei eines bestimmten Dateityps besteht. So soll der Fokus ganz auf die Visualisierung gelenkt werden.

Während der Auswertung fiel schnell auf, dass die aktuelle Umsetzung der Bild-Methode zum Erreichen der gesetzten Zielstellung eher unvorteilhaft ist. Die zweizeilige Darstellung erlaubt keinen guten Überblick über den gesamten Sektor, da die Bilder zu breit sind, um alle Pixel gleichzeitig zu betrachten. Daher wurde die Funktion `create_hsl_pixels()` noch einmal angepasst. In der neuen Form werden die Bilder mit einer Breite von 16 Pixeln in 32 Zeilen angeordnet.

In diesem Zug wurde auch die in Abschnitt 5.2.5 implementierte Hilbertumformung noch einmal überarbeitet. In der bisherigen Version wurde die Umformung benutzt, um die Bytes im Sektor hilbertförmig einzulesen und dann zeilenweise in das Bild einzufügen. Dadurch werden direkt aufeinanderfolgende Bytes jedoch auseinandergerissen, was der These von Conti wiederspricht, wonach aufeinanderfolgende Bytes zueinander in Relation stehen (siehe Abschnitt 2.4.1.1). In der neuen Version werden die Bytes daher so verarbeitet, wie sie eingelesen werden. Die Hilbertkurve wird erst beim Einfügen der berechneten Pixel in das Bild angewendet. Um dies zu erreichen, werden zunächst zwei 16x16-Arrays entlang der Hilbertkurve mit Pixeln gefüllt und anschließend mittels `np.vstack()` zu einem 32x16 großem Array kombiniert. Die aktualisierte Funktion ist in Quelltext 5.11 zu sehen.

```
1  def read_hsl_pixels(filename):
2   sat = 1                                  # saturation = const = 1
3   blockpixels = np.zeros([32, 16, 3])   # create array for pixels of current
4                                 # block; 2 & 256 so each block makes 2 rows of
5                                 #  final pic, each of them being 256 pixels
      wide
6
7   if hilbert:
8     array = np.arange(0, stop=256)
9     dims = 2
10    y_locs, x_locs = hc.decode(array, dims, 8).T.tolist()
11
12    array1 = np.zeros([16, 16, 3])
13    array2 = np.zeros([16, 16, 3])
14    counter = 0
15    locs_counter = 0
16
17  else:
18    bytecounter = 0
19    rowcounter = 0
20
21  for currentByte in block:              # for each byte (cast to int)
22    hue, light = currentByte >> 4, currentByte & 0x0F   # split currentbyte
23                                  # into nibbles
24    hue = floats[hue]                      # convert to float between 0 and 1
25    light = floats[light]                  # using floats array
26
27    # add hsl-pixel-values to blockpixels
28    if hilbert:
29      if counter <= 255:
30         array1[x_locs[counter]][y_locs[counter]] = [hue, light, sat]
31      else:
32         locs_counter = counter - 256
33         array2[x_locs[locs_counter]][y_locs[locs_counter]] = [hue, light,
     sat]
34      counter += 1
35    else:
36      if bytecounter <= 15:
37         blockpixels[rowcounter][bytecounter] = [hue, light, sat]
38      else:
39         rowcounter += 1
40         bytecounter = 0
41         blockpixels[rowcounter][bytecounter] = [hue, light, sat]
42      bytecounter += 1
```

```
43
44  if hilbert:
45    array1 = np.transpose(array1, axes=[1,0,2])
46    array2 = np.transpose(array2, axes=[1,0,2])
47    blockpixels = np.vstack([array1,array2])
48
49  return blockpixels
```

Quelltext 5.11: Aktualisierte Funktion zur Erstellung von HSL-Bildern

Darüber hinaus wurde die Funktion so verändert, dass sie jetzt statt einem Dateinamen einen einzelnen Sektor (im Quellcode als `block` bezeichnet) als Parameter annimmt. Dazu wurde der Aufruf von `read_blocks()` aus der Funktion entfernt und erfolgt nun vor Aufruf von `read_hsl_pixels()`.

5.3.3 Ergebnis

Die betrachteten Methoden sind alle unterschiedlich gut geeignet, um Muster zur Unterscheidung von Dateitypen zu finden. Während die mittels Digramm- und Trigramm-Analyse erzeugten Visualisierungen verschiedene Muster bilden, die teilweise charakteristisch für einzelne Dateitypen zu sein scheinen, konnten in den Bild-Darstellungen nicht genügend Gemeinsamkeiten innerhalb einer Datei festgestellt werden. Auch die Regionen-Darstellung wirkt eher ungeeignet zur Analyse von Sektoren. Die genauen Ergebnisse der visuellen Bewertung werden im Folgenden für die jeweils angewendete Methode beschrieben.

5.3.3.1 Bild

Bei Betrachtung der erstellten HSL-Bilder lassen sich mögliche Relationen zwischen benachbarten Pixeln nur begrenzt erkennen. Ausnahme hiervon bilden einzelne Sektoren, in denen ganze Bytefolgen enthalten sind, wie zum Beispiel bei der exe- und der elf-Datei. Spezifische Muster für die jeweiligen Dateiformate fallen eher nicht auf.

Beispielsweise ist gut zu erkennen, dass die einzelnen Sektoren der bmp-Datei zusammen zu gehören scheinen, da sich über die Sektoren hinweg ein Farbverlauf zieht. Würden die einzelnen Sektoren jedoch jeder für sich betrachtet und versucht einen Zusammenhang zu finden, würden sie wohl nicht als eine Gruppe erkannt werden, da jeder Sektor in einem anderen Farbton dargestellt wird.

Die Sektoren der apk-Datei sind dagegen alle mit ähnlich leuchtenden Farben dargestellt. Diese gilt jedoch auch für andere Dateien, wie die jpg-, png-, mp4- oder pdf-Datei. Im Gegensatz zu den anderen vier Dateiformaten, kommen in der Visualisierung der pdf-Datei jedoch auch größere Abschnitte vor, die eher grün als bunt gefärbt sind. Auch die docx-, pptx- und xlsx-Datei bewegen sich im selben bunten Farbspektrum. Statt der grünen Bereiche enthalten diese jedoch größere schwarze Abschnitte. Da diese schwarzen Bereiche keine Regelmäßigkeit aufweisen und nicht in jedem Sektor der betreffenden Datei vorkommen, eignen auch sie sich nur begrenzt als Erkennungsmerkmal.

Die elf-Datei zeigt in den meisten Sektoren ein recht deutliches Muster. Die Visualisierungen sind hier größtenteils schwarz, haben aber senkrecht angeordnete bunte Pixel. In dieser Darstellung wird die Struktur der Datei erkennbar, wodurch ein Wiedererkennungswert gegeben ist. Da es so markant ist, fallen auch die teilweisen Unterbrechungen des Musters nicht sonderlich ins Gewicht. Ausnahme bildet hier der letzte Sektor, welcher sich komplett im grünen Farbbereich befindet.

Auch die Visualisierungen der txt-, xml- und svg-Dateien bestehen hauptsächlich aus Grüntönen. Während die txt-Datei etwas weniger schwarze Pixel zu enthalten scheint, ist zwischen der xml- und der svg-Datei kein deutlicher Unterschied zu erkennen.

Es lassen sich also keine charakteristischen Muster feststellen, die sich über alle Sektoren einer Datei erstrecken und diese damit deutlich von den anderen abgrenzt. Die Verwendung dieser Methode scheint daher für die Analyse ganzer Dateien oder Speicherabbilder, wie in den Abschnitten 2.4.1.2, 2.4.3.1 und 2.4.3.2 beschrieben, besser geeignet zu sein als für den hier gewünschten Anwendungsfall.

5.3.3.2 Digramm

Bei diesem Ansatz fallen recht schnell einzelne Muster auf, die eher dateitypspezifisch wirken. Besonders markant ist zum Beispiel das Muster aus drei diagonalen Punktewolken, welches in jedem Sektor der bmp-Datei in der oberen linken Ecke auftritt. Auch bei der epub-Datei sind dünne diagonale Linen erkennbar, welche bei anderen Dateitypen so nicht vorkommen. Diese Datei zeigt aber auch sehr deutlich, dass das scheinbar typische Muster nicht über alle Sektoren hinweg auftritt. Der Unterschied zwischen den ersten beiden und den restlichen betrachteten Sektoren lässt sich aber vermutlich dadurch erklären, dass die ersten Sektoren den Dateiheader enthalten und der Datenstream erst später beginnt.

Darüber hinaus fallen bei mehreren Dateien Ansammlungen von Punkten im Bereich links oben auf, die zusammengenommen vier Quadrate ergeben. Bei genauerer Betrachtung fällt auf, dass es sich hierbei um aufeinanderfolgende, druckbare Zeichen aus dem ASCII-Bereich handelt. Auch wenn diese bei mehreren Dateien vorkommen, sind sie nicht immer gleich. In der Visualisierung der txt-Datei fällt zum Beispiel vor allem das Quadrat rechts unten auf, mit einigen Strichen am Rand der anderen Quadrate, während bei der svg-Datei auch im Quadrat links oben eine starke Häufung von Punkten erkennbar ist. Auch bei der xml-Datei scheint der Fokus auf dem Quadrat rechts unten zu liegen, allerdings wirken die Punkte hier gleichmäßiger auf aller vier Quadrate verteilt. Die Visualisierung der pdf-Datei enthält ebenfalls die Quadrate links oben und rechts unten, allerdings wirken sie hier schwächer und sind von sehr diffus verteilten einzelnen Punkten umgeben. Eine solche Punkteverteilung (mit und ohne ASCII-Quadrate) tritt auch bei verschiedenen anderen Dateitypen auf, ohne dass genauere Muster erkennbar sind.

Weiterhin lässt sich in der gif-Datei ein Muster erkennen, da die Punkte hier nicht ganz so gleichmäßig verteil sind. Während im ersten Sektor eine leichte Konzentration entlang der Diagonalen auftritt, entstehen ab dem dritten Sektor tendenziell kurze Linien, die vom oberen Bildrand relativ senkrecht nach unten verlaufen. Der zweite Sektor bildet eine Mischung aus beiden Muster und markiert damit vermutlich den Übergang vom Dateiheader zum Datenstream.

Auch die ausführbaren Dateien unterscheiden sich von den anderen Dateitypen in ihrer Visualisierung. So lassen sich bei der exe-Datei ab dem dritten dargestellten Sektor schwache senkrechte und waagerechte Linien in den Punkten ausmachen. Auch hier fallen die ersten Sektoren jedoch aus der Reihe, was die Vermutung nahe legt, dass der Header sich über die ersten beiden Sektoren erstreckt. Auch im dritten Sektor ist über den geraden Linien eine Diagonale sehr deutlich zu erkennen, sodass dieser vermutlich auch noch Teile des Headers enthält.

Die elf-Datei (0003-so.so) scheint auf den ersten Blick kaum Gemeinsamkeiten in ihren Sektoren aufzuweisen. Teilweise sind einzelne Punkte im Bild verstreut, teilweise ist eine Diagonale zu erkennen, während in zwei Sektoren wieder das ASCII-Quadrat auftaucht. Die Visualisierung eines anderen Sektors erscheint gar leer. Bei genauerer Betrachtung fällt jedoch eine Gemeinsamkeit über alle Sektoren hinweg auf, welche als charakteristisches Muster gewertet werden kann. So weisen alle Sektoren Punkte entlang der oberen und linken Kante der Darstellung auf, die in dieser Form bei anderen Dateitypen eher selten auftreten.

Die nicht explizit aufgeführten Dateiformate ähneln sich in ihrer Visualisierung stark. Bei allen sind die Punkte sehr gleichmäßig über das Diagramm verteilt, wodurch keine Muster erkennbar sind, die bei Erkennung des jeweiligen Dateityps hilfreich sein könnten.

Insgesamt gesehen, lässt sich keine allgemeingültige Aussage über die Nutzbarkeit der Methode treffen. Während einige Dateiformate sehr deutliche Muster aufzuweisen scheinen, sind diese bei anderen Typen eher schwach ausgeprägt oder sogar gar nicht erkennbar. Gegebenenfalls ist eine Verwendung der Methode auch nur für ausgewählte Dateitypen möglich.

5.3.3.3 Trigramm

Auch wenn die Sektor-Visualisierungen jetzt dreidimensional dargestellt sind, lassen sich Ähnlichkeiten in der Darstellung einer Datei mit der Digramm-Methode erkennen. So treten insgesamt ähnliche Muster auf, wirken in der Trigramm-Darstellung zum Teil jedoch etwas ausgeprägter.

Ein Beispiel dafür ist exe-Datei. Hier fallen die kastenförmig angeordneten waagerechten und senkrechten Linien erst richtig auf, wenn die Visualisierung aus mehreren Winkeln betrachtet wird. Das Muster tritt dann aber stärker hervor als in der zweidimensionalen Darstellung. Auch in der elf-Datei ist die Konzentration der Punkte hin zu den Außenkanten des Diagramms bei der Trigramm-Analyse besser zu erkennen. Darüber hinaus zeigt sich auch in dieser Darstellung sehr deutlich die Diagonale, die bei beiden Dateien jeweils über zwei Sektoren hinweg vom Koordinatenursprung zur gegenüberliegenden Seite verläuft. Auch bei der bmp-Dateien sind die Parallelen zur zweidimensionalen Variante deutlich erkennbar. Wie beim Digramm-Ansatz, sind hier ebenfalls in jedem Sektor drei längliche, diagonale Punktansammlungen zu sehen.

Im Gegensatz dazu konnte das nach der Digramm-Analyse bei der epub-Datei sichtbare Muster nicht wiedererkannt werden. Hier wirkten die Punkte im Diagramm sehr gleichmäßig verteilt, ohne dass ein charakteristisches Muster erkannt werden konnte. Gleiches gilt für die apk-, jpg- und png-Dateien, die bei Verwendung der Trigramm-Analyse keine Auffälligkeiten aufweisen und nur aus diffusen Punkten zu bestehen scheinen.

Die dreidimensionale Visualisierung der gif-Datei besteht aus einer eher gleichmäßigen Punkteverteilung. Bei genauerer Betrachtung der Anordnung der Punkte konnte allerdings eine leichte Tendenz der Punkte zum Koordinatenursprung festgestellt werden. Dieses Verhalten zeigt sich auch in der zweidimensionalen Darstellung vor allem im ersten Sektor der Datei. Ein ähnliches Verhalten zeigt sich bei den Dateiformaten docx, xlsx und pptx. Diese drei Formate weisen darüber hinaus untereinander sehr viele Ähnlichkeiten auf. So enthält bei diesen Formaten insbesondere die Visualisierung des ersten und dritten Sektors nur relativ wenige Punkte. Die Muster in den anderen Sektoren sind dagegen vergleichbar mit denen der apk-, jpg- und png-Dateien. Sie sind voller Punkte, deren Streuung kein charakteristisches Muster aufweist.

Die mp4-Datei zeigt in den ersten beiden Sektoren einige Punkte im Bereich des ASCII-Bereichs, jedoch ist der Würfel hier nicht gut als solcher zu erkennen. Die anderen Sektoren enthalten im Gegensatz dazu kein erkennbares Muster.

Bei den anderen Dateiformaten, bei denen einen Fokus auf dem ASCII-Bereich festgestellt werden konnte, lassen sich hier konkrete Formen feststellen, die sich voneinander unterscheiden. So besteht die Visualisierung der txt-Datei fast ausschließlich aus Punkten im ASCII-Bereich mit wenigen einzeln verteilten Werten. Bei der pdf-Datei sind dagegen trotz des deutlichen Hervortretens der Werte im ASCII-Bereich auch im restlichen Diagramm noch reichlich Punkte verteilt. Die svg-Datei nutzt dagegen in den meisten Sektoren nur einen kleinen Teil des ASCII-Bereichs, in dem die Punkte sehr komprimiert auftreten. In den Sektoren, in denen sich die Punkte im gesamten ASCII-Bereich befinden, scheint trotzdem eine der Ecken des ASCII-Würfel stärker mit Punkten befüllt. Dieser Bereich tritt dadurch hervor. Darüber hinaus lässt sich auch bei der xml-Datei eine Konzentration der Punkte im ASCII-Bereich feststellen. Im Unterschied zu den vorgenannten Dateitypensind hier nicht nur die Ecken des ASCII-Würfels gut befüllt, sondern auch die Bereiche zwischen diesen.

Wie aus den obigen Ausführungen ersichtlich wird, gibt es auch bei der Trigramm-Methode einige Dateitypen, die einen höheren Wiedererkennungswert aufweisen als andere. Somit ist die Methode für die Erkennung einiger Dateiformate geeignet, aber tendenziell nicht für alle.

5.3.3.4 Regionen

Die Visualisierung der zusammenhängenden Regionen sollte dabei helfen, Muster in den zugehörigen Digramm- und Trigramm-Darstellungen zu erkennen. Durch das Entfernen aller Werte unter dem Mittelwert der jeweiligen Häufigkeitsmatrix sollten eher unbedeutende Punkte vor der Auswertung herausgefiltert werden. Im Vergleich scheint diese Methode jedoch nicht besonders gut geeignet zu sein, da die Mittelwert-Filterung bei den einzelnen Dateitypen unterschiedlich gut funktioniert.

Die Muster, die in der Digramm- und Trigramm-Darstellung bereits deutlich wahrnehmbar waren, wurden durch die Regionen-Darstellung noch einmal hervorgehoben. Beispiele heirfür sind die Sektoren der bmp-Datei, aber auch die Sektoren, bei denen die Quadrate im ASCII-Bereich festgestellt werden konnten. Auch in der Regionen-Darstellung der epub-Datei lassen sich die Linien der Digramm-Darstellung wiederfinden, wenn auch deutlich weniger ausgeprägt.

Insbesondere bei den Dateitypen, deren Digramm- und Trigramm-Visualisierungen eher diffus waren, scheint die Regionen-Darstellung nicht hilfreich. Hier bleiben nur vereinzelte Punkte übrig, die für sich nicht aussagekräftig sind. Beispiele dafür sind unter anderem die Dateien in den Microsoft Office-Dateitypen docx, pptx und xlsx sowie die jpg- und mp4-Dateien.

Darüber hinaus erscheint auch das Labeln der Punkte als zusammengehörige Regionen als nicht sehr zielführend. Viele Punkte, die für den menschlichen Betrachter eine Gruppe bilden, werden vom Algorithmus nicht als solche erkannt. Dies liegt daran, dass die Erkennung von Regionen immer nur mit einem kleinen Ausschnitt der Gesamtdarstellung arbeitet und Pixel nicht als zusammhängend wertet, sobald ein Nullwert in der gefilterten Häufigkeitsmatrix zwischen den jeweiligen Pixeln liegt. Dies ist in der Theorie richtig, für den hier gewünschten Effekt aber nicht hilfreich. Betrachtet ein Mensch die Darstellungen, so nimmt er sie zunächst im Ganzen wahr und ist dadurch geneigter, auch dann Muster zu erkennen, wenn sie unterbrochen sind oder aus mehreren kleineren Gruppen bestehen.

Das Ziel, die Muster über eine Filterung der Werte in der Häufigkeitsmatrix besser hervortreten zu lassen, funktioniert bei Anwendung des Mittelwerts also nur begrenzt. Auch das Labeln zusammenhängender Regionen bringt nicht die gewünschten Ergebnisse.

5.3.3.5 Hilbertkurve

In der bisherigen Forschung wurde die Hilbertkurve nur im Zusammenhang mit der Visualisierung von Bytes in Form von Pixeln, wie bei den HSL-Bildern, eingesetzt. Durch das Umformen der Daten nach der Hilbertkurve werden Bytes miteinander in Relation gesetzt, die sonst eher nicht direkt nebeneinander liegen würden. Um zu erforschen, welche Auswirkungen dies auf die n-Gramm-Analyse hat, wurden daher auch die mittels Digramm- und Trigramm-Methode verarbeiteten Daten entlang der Hilbertkurve umsortiert. Im Ergebnis war die Anwendung der Hilbertkurve in allen Fällen nur mäßig erfolgreich.

Auf die HSL-Bilder angewendet ist die Hilbertkurve nicht sehr hilfreich. Da die Bilder schon in Normalform als unvorteilhaft für die gewünschte Zielstellung empfunden werden, bringt auch eine Neusortierung der Pixel keine nennenswerte Verbesserung. Auch wenn die schwarzen Bereiche in den vorwiegend grünen Sektoren nun eher auffallen, reicht dies immer noch nicht aus, um Unterscheidungsmerkmale zwischen den Sektordarstellungen verschiedener Dateitypen zu erkennen. Selbst bei der elf-Datei, welche vorher als einzige Datei bei dieser Methode ein charakteristisches Muster aufwies, tritt dagegen eine Verschlechterung auf. Das Muster der einzelnen bunten Punkte auf schwarzem Hintergrund ist zwar immer noch erkennbar, die klare senkrechte Struktur geht jedoch verloren.

In der Digramm-Darstellung sorgt die Umsortierung der verarbeiteten Bytes in einem Sektor dafür, dass die Punkte in der Darstellung verteilter auftreten. Um die bisherigen Muster können weitere Punkte festgestellt werden. Dadurch werden einige der zuvor erkannten Muster noch markanter, da sie durch die zusätzlichen Punkte besser hervortreten. Dies ist beispielsweise bei der xml- und der svg-Datei der Fall.

Bei anderen Dateien verschwimmt das Muster durch die zusätzlichen Punkte. Als Beispiel sei hier die bmp-Datei benannt. Wo ohne Hilbertkurve drei separate, längliche Punktgruppen klar zu erkennen waren, lässt sich die Punktverteilung nach Anwendung der Hilbertkurve je nach Sektor eher als ein oder mehrere rundliche Flecken beschreiben. Auch das Muster der epub-Datei ist in dieser Darstellung nicht mehr zu erkennen.

Bei den Dateitypen, für die aufgrund der gleichmäßigen Streuung der Punkte im Diagramm keine Muster festgestellt werden konnten, hat die Umformung mit der Hilbertkurve keinen großen Effekt. Die Punkte befinden sich zwar nicht mehr an den exakt selben Positionen, gut erkennbare Muster entstehen jedoch auch nicht.

Die Veränderungen in der Trigramm-Darstellung sind mit denen der Digramm-Darstellung vergleichbar. Auch hier sind die Muster nun weniger klar umrissen, sodass die Kanten der Muster undeutlicher werden. Während dies teilweise vorteilhaft ist, da die Muster mehr Fläche einnehmen und somit besser erkannt werden können, lässt die Anwendung der Hilbertkurve teilweise auch das Muster verschwinden.

Im Ergebnis lässt sich nicht genau einschätzen, welche Auswirkungen die Anwendung der Hilbertkurve auf den Klassifizierungsprozess hat, da einige Dateitypen so schwerer erkannt werden, andere dagegen besser. Abhängig vom Dateityp ist somit sowohl eine Verbesserung als auch eine Verschlechterung der Klassifizierungsergebnisse im Vergleich zur Digramm- und Trigramm-Methode ohne Hilbertkurve denkbar. Bei der Methode der HSL-Bilder verringert die Umformung der Daten mit der Hilbertkurve nochmals die Chance, Muster zu erkennen. Hierfür ist sie also nicht geeignet.

5.4 Statistische Bewertung

Um die Ergebnisse der visuellen Bewertung zu verifizieren und um den tatsächlichen Nutzen der entwickelten Methoden festzustellen, wurden diese anschließend auch statistisch ausgewertet. Das dabei angewendete Vorgehen und die Ergebnisse sind im Folgenden beschrieben.

5.4.1 Vorbereitung

Aufgrund der schlechten Ergebnisse in der visuellen Bewertung werden die Ansätze der Bild- und der Regionen-Darstellungen in der statistischen Bewertung nicht weiter verfolgt. Der Fokus soll hier auf der Digramm- und Trigramm-Analyse liegen.

Der Vergleich der Methoden erfolgt mit einem Naive-Bayes-Klassifikator. Diese Art Klassifikator wird vor allem in der Textverarbeitung als effektiv angesehen, wobei z.B. betrachtet wird, wie oft einzelne Worte auftreten [16]. Da in der vorliegenden Arbeit im Rahmen der Digramm- und Trigramm-Analysen ebenfalls Häufigkeitsanalysen als Datengrundlage vorliegen, erscheint eine gute Performance eines Naive-Bayes-Klassifikators auch hier als wahrscheinlich. Zudem basieren die Ergebnisse eines solchen Klassifikators auf nachvollziehbaren statistischen Berechnungen, sodass er auch im Bereich der IT-Forensik angewendet werden könnte.

Wie bereits in Abschnitt 2.1.4 beschrieben, müssen bei diesem Verfahren zunächst alle Klassen- und Merkmalswahrscheinlichkeiten berechnet werden. Dies geschieht in der Trainingsphase, für welche Trainingsdaten erforderlich sind. Aus diesen kann der Klassifikator die Merkmale der einzelnen Klassen „lernen“. Hierfür werden mehr Daten benötigt als für die visuelle Bewertung, allerdings wurde auch hier nur eine kleine Auswahl an Daten verwendet, um einen ersten Überblick über die Methoden zu gewinnen. In einem ersten Ansatz wurden von den zu bearbeitenden Dateitypen jeweils die ersten beiden Dateien im NapierOne-Tiny-Datenset ausgewählt und in nach dem Dateityp benannten Unterordnern sortiert in einem Verzeichnis abgelegt.

```
1  def generate_test_training_data(in_path, out_path, use_hilbertcurve, mode='2
     d'):
2    [...]
3    for class_name in classesArray:
4      path = os.path.join(in_path,class_name)
5      file_list = os.listdir(path)
6      label_as_int = int(np.where(classesArray == class_name)[0])
7      data_list = []
8      label_list = []
9      for file_name in file_list:      # iterate over all files in directory
10       [...]
11       current_file = os.path.join(path, file_name)
12       if os.path.isfile(current_file):
13         match mode:
14           case '2d':
15             x = ngram_analysis.create_digram_freq_map(current_file,
     use_hilbertcurve)
16           case '3d':
17             x = ngram_analysis.create_trigram_freq_map(current_file,
     use_hilbertcurve)
18           case _:
19             print(f"'{mode}' is not a valid mode. Please choose from '2d', '3
     d'.")
20             return 0
21         try:
22           while True:
23             freqmap, min, max, size = next(x)
24             data_list.append(data_manipulation.hilbert_flatten(freqmap).
     astype('int'))
25             label_list.append(label_as_int)
26         except StopIteration:
27           pass
28
29     # splitting data into test and training data
30     x_training_list, x_test_list, y_training_list, y_test_list =
     train_test_split(data_list, label_list, test_size=0.2, random_state=42)
31     # saving data to files
32     tr_maps = np.vstack(x_training_list)
33     name = f'training_data_class_{class_name}.npy'
34     np.save(os.path.join(out_path,name), tr_maps)
35     te_maps = np.vstack(x_test_list)
36     name = f'test_data_class_{class_name}.npy'
37     np.save(os.path.join(out_path,name), te_maps)
38   return 1
```

Quelltext 5.12: Erzeugung von Test- und Trainingsdaten

Wie in Quelltext 5.12 zu sehen, werden die Dateien zunächst mit der Digramm- oder der Trigramm-Methode verarbeitet. Zusätzlich werden diese mit der Funktion `hilbert_flatten()` in eine eindimensionale Liste überführt. Dies sollte die Erstellung des Klassifikators erleichtern, da so bei beiden Analysemethoden jeweils Daten in gleicher Form erzeugt werden, wenn auch mit unterschiedlicher Länge. Im nächsten Schritt werden die Listen in Test- und Trainingsdaten aufgeteilt. So wird sichergestellt, dass der Klassifikator nicht mit den selben Daten getestet wird, mit denen er trainiert wurde. Zu beachten ist, dass der Quelltext der Funktion `generate_test_training_data()` hier nur ausschnittsweise eingefügt wurde, um die wesentlichen Befehle nachzuvollziehen. Die Auslassungen sind mit [...] gekennzeichnet.

Um die Generierung von Trainingsdaten vom eigentlichen Trainings- und Klassifikationsprozess abzugrenzen, werden die Test- und Trainingsdaten zunächst in Dateien abgespeichert. Soll nun ein Klassifikator erzeugt und trainiert werden, müssen die entsprechenden Trainingsdaten nur wieder eingelesen und nicht jedes Mal neu erzeugt werden. So wird es möglich, einmal erstellte Trainingsdaten immer wieder zu verwenden.

5.4.2 Umsetzung

Nachdem die Trainings- und Testdaten im vorherigen Schritt erzeugt wurden, ist die Erstellung und Verwendung des Klassifikators nun vergleichsweise simpel, da in Python auf fertige Funktionen des `scikit-learn`-Packages zurückgegriffen werden kann. Quelltext 5.13 zeigt die wichtigsten Ausschnitte der Funktion `create_classifier()`, in der ein Klassifikator auf Basis von vorher erzeugten und abgespeicherten Test- und Trainingsdaten erstellt, trainiert und getestet wird.

```
1  def create_classifier(data_path):
2    [...]
3    # load training data from file, abort function if load_data() is
       unsuccessful
4    x_train, y_train = load_data(data_path,'training')
5    if x_train == 0:
6      return 0
7    sizes = [arr.shape[0] for arr in x_train]
8    number_of_samples = np.sum(sizes)
9    x_train_flat = np.concatenate(x_train, axis=0)
10   y_train_flat = np.zeros([number_of_samples])
11   index = 0
12   class_no = 0
13   for j in sizes:
14     for i in range(j):
15       y_train_flat[index] = class_no
16       index += 1
17     class_no += 1
18
19   # create a Gaussian Naive Bayes classifier and fit it on the training data
20   gnb = GaussianNB()
21   gnb.fit(x_train_flat, y_train_flat)
22
23   # load test data from file, abort function if load_data() is unsuccessful
24   x_test, y_test = load_data(data_path, 'test')
25   if x_test == 0:
26     return
```

```
27  # flatten test data
28  x_test_flat = np.concatenate(y_train, axis=0)
29  y_test_flat = np.zeros([number_of_samples])
30  index = 0
31  class_no = 0
32  for j in sizes:
33    for i in range(j):
34      y_test_flat[index] = class_no
35      index += 1
36    class_no += 1
37
38  # let gnb make predictions on the test data
39  y_pred = gnb.predict(x_test_flat)
40
41  # compare predicted labels in y_pred to actual labels in y_test
42  accuracy = accuracy_score(y_test_flat, y_pred)
43  [...]
44  micro_precision = precision_score(y_test_flat, y_pred, average='micro')
45  micro_recall = recall_score(y_test_flat, y_pred, average='micro')
46  micro_f1 = f1_score(y_test_flat, y_pred, average='micro')
47  macro_precision = precision_score(y_test_flat, y_pred, average='macro')
48  macro_recall = recall_score(y_test_flat, y_pred, average='macro')
49  macro_f1 = f1_score(y_test_flat, y_pred, average='macro')
50  conf_mat = confusion_matrix(y_test_flat, y_pred)
51  [...]
52  return
```

Quelltext 5.13: Erzeugung und Testen eines Naive-Bayes-Klassifikators

Bei ersten Tests fiel auf, dass für die Dateitypen png und txt im Vergleich zu den anderen Klassen nicht ausreichend Testdaten vorlagen. Daher wurde der Datensatz zur Erstellung der Test- und Trainingsdaten angepasst. Im aktualisierten Datensatz sind für den Dateityp txt die ersten zwölf Dateien aus dem NapierOne-Tiny-Datenset enthalten. Zudem wurden die zuvor verwendeten Dateien aus dem Ordner png-from-web durch die ersten beiden Dateien des png-c9-Ordners ersetzt. Somit stehen nun von beiden Typen mehr Sektoren zum Training zur Verfügung.

Es bestand die Überlegung, dem Klassifikator zusätzlich zu den erstellten Häufigkeitsmatrizen auch den minimalen und maximalen Wert der Matrix sowie einen Entropiewert zu Verfügung zu stellen. Damit stünden dem Klassifikator mehr Features zur Verfügung, was die Klassifikationsergebnisse positiv beeinflussen könnte. Der hier implementierte Klassifikator kann jedoch nur eine Gruppe Features gleichen Dateityps verarbeiten. Daher wurde der Klassifikationsprozess auf die zur Liste umgeformten Häufigkeitsmatrizen konzentriert und die einzelnen Werte werden vorerst ignoriert.

Während dieser Ansatz für die Digramm-Analyse sehr gut funktionierte, kam es bei der Anpassung der Klassifikator-Funktionen an die Trigramm-Methoden zu Problemen. Zum Einen ist die Bearbeitungsdauer mehrere Tage lang, zum Anderen treten Speicherprobleme auf. Die Bearbeitungsdauer ließ sich verkürzen, indem Teile der Trainingsdatengenerierung über das Python-Package `multiprocessing` parallel verarbeitet werden. Dafür wird die Trainingsdatengenerierung, wie in Quelltext 5.14 gezeigt, pro Dateityp in eine neue Funktion `read_class_directory()` kopiert und anschließend über die `map()`-Funktion gestartet. Um möglichst viele CPU-Kerne auszunutzen, wur-

de zunächst die Anzahl der CPU-Kerne des Rechners bestimmt und davon die Hälfte verwendet. Da dies jedoch zu Problemen mit dem Arbeitsspeicher führte, wurde später `num_cores = 4` gesetzt, sofern mindestens 5 Kerne vorhanden sind.

```
# number of cpu cores to use
num_cores = multiprocessing.cpu_count()//2
# create a pool of processes that are processed in parallel
with multiprocessing.Pool(num_cores) as pool:
 other_params = [in_path, mode, use_hilbertcurve, out_path]
 process_params = ([name, other_params] for name in classesArray)
 pool.map(read_class_directory,process_params)
```

Quelltext 5.14: Parallelisierung der Trainingsdatenerstellung

Weiterhin war einer erneute Anpassung der Datenbasis nötig, da die Generierung von Traingsdaten auf Basis der Trigramm-Analyse Arrays in einer Größe erzeugt, für die Python keinen Speicherplatz allokieren kann. Daher wurden alle Dateien mit einer Dateigröße von mehr als 1 MB aus der Datenbasis entfernt und möglichst kleine Dateien des entsprechenden Dateityps aus dem NapierOne-Tiny-Set eingefügt. Dabei wurde darauf geachtet, dass von jedem Dateityp mindestens 60 KB und maximal 2 MB an Dateien vorliegen.

Trotz wiederholter Anpassung der Datenbasis konnte der Klassifikator für die Trigramm-Analyse nicht erfolgreich durchlaufen werden. Zunächst traten bei der Erstellung der Trainingsdaten wiederholt Fehler der Kategorie `ArrayMemoryError` auf, welche darauf schließen lassen, dass der Python-Prozess zu viel Arbeitsspeicher in Anspruch nimmt. Auch wurden die Trainings- und Testdaten von bereits bearbeiteten Dateitypen nicht immer erstellt. Dies könnte daran liegen, dass die Arrays zu groß sind, um sie mit der Funktion `numpy.save()` abzulegen. Zum Vergleich: Aus zwei Dateien des Dateityps epub mit einer Gesamtgröße von 459 KB entstehen 46 GB Trainings- und 11,5 GB Testdaten. Da diese zur Erstellung und zum Testen des Klassifikators wieder eingelesen werden müssen, ist der Bedarf an Arbeitsspeicher also auch bei einer reduzierten Datenbasis von insgesamt nur 13,4 MB Speichergröße enorm.

Doch das Erstellen der Trainingsdaten ist nicht die einzige Schwierigkeit. Nachdem in einem Versuch Test- und Trainingsdaten für die Dateitypen epub, exe, jpg, txt und xml erstellt und gespeichert werden konnten, sollte der Klassifikationsprozess zumindest mit diesen fünf Klassen gestartet werden. Dafür wurde der Quellcode des Klassifikators temporär angepasst, sodass das Programm auch mit diesen wenigen Klassen ausgeführt werden kann. Doch die Funktionen `fit()` und `predict()` des Klassifikators brachen auch mit diesen vermeintlich wenigen Daten mit Fehlermeldungen ab. Daraufhin wurde der Klassifikator erneut angepasst, sodass dieser nur noch die Dateitypen exe, jpg und txt unterscheiden muss, da die Trainingsdaten dieser Dateitypen die geringste Speichergröße hatten. Mit diesen Einschränkungen konnte nun eine Klassifikation als Minimalbeispiel durchgeführt werden.

Da mit der Trigramm-Methode kein Klassifikationsprozess mit allen Dateitypen erfolgreich durchlaufen werden konnte, ist es nicht sinnvoll die beiden Ansätze auf derselben Datenbasis zu vergleichen. Daher wurde die Datenbasis für die Klassifikation auf Basis der zweidimensionalen Methode wieder erhöht, diesmal mit dem Ziel insbesondere die mit nur wenigen Sektoren vertretenen Dateitypen stärker aufzufüllen. Dafür wurde eine Mindestmenge von 1 MB Daten pro Dateityp festgelegt. Dieses

Tabelle 5-1: Zur Erzeugung von Trainingsdaten verwendete Dateien

Typ	Dateinamen
apk	0001-apk.apk, 0002-apk.apk
bmp	0001-bmp.bmp, 0002-bmp.bmp
docx	0001-docx.docx, 0002-docx.docx
elf	0001-so.so, 0002-so.so
epub	0001-epub.epub, ..., 0006-epub.epub
exe	0001-exe.exe, 0002-exe.exe, 0005-exe.exe, 0010-exe.exe
gif	0001-gif.gif, 0002-gif.gif
jpg	0001-jpg-q050.jpg, ..., 0010-jpg-g050.jpg
mp4	0001-mp4.mp4, 0002-mp4.mp4
pdf	0001-pdf.pdf, 0002-pdf.pdf
png	0001-png-c9.png, 0002-png-c9.png
pptx	0001-pptx.pptx, 0002-pptx.pptx
svg	0001-svg-from-web.svg, 0002-svg-from-web.svg, 0012-svg-from-web.svg, 0025-svg-from-web.svg, 0026-svg-from-web.svg, 0059-svg-from-web.svg, 0065-svg-from-web.svg, 0099-svg-from-web.svg
txt	0001-txt.txt, ..., 0012-txt.txt, 0031-txt.txt, 0036-txt.txt, 0047-txt.txt, 0048-txt.txt, 0052-txt.txt, ..., 0054-txt.txt, 0064-txt.txt, 0071-txt.txt, 0079-txt.txt, 0091-txt.txt, 0099-txt.txt
xlsx	0001-xlsx.xlsx, 0002-xlsx.xlsx
xml	0001-xml.xml, ..., 0100-xml.xml
exe	0001-exe.exe, 0002-exe.exe
jpg	0001-jpg-q050.jpg, 0002-jpg-g050.jpg
txt	0001-txt.txt, ..., 0012-txt.txt

Kriterium konnte für fast alle Dateitypen erreicht werden. Einzige Ausnahme waren die xml-Dateien, da hier alle 100 im NapierOne-Tiny-Set enthaltenen Dateien zusammen nur 857 KB ergeben. Tabelle 5-1 gibt einen Überblick über die verwendeten Dateien.

Im oberen Teil sind dabei die Dateien gelistet, die für die Klassifizierung mit der Digramm-Methode verwendet wurden. Um die Tabelle nicht unnötig zu vergrößern, wurde bei mehr als zwei numerisch aufeinanderfolgenden Dateien auf die einzelne Auflistung verzichtet. Diese sind stattdessen teilweise als Intervall angegeben. Im unteren Teil sind die Dateien, die für eine erfolgreiche Klassifizierung mit dem dreidimensionalen Ansatz verwendet wurden.

5.4.3 Ergebnis

Die soeben implementierten Klassifikatorfunktionen sollen nun anhand der in Abschnitt 2.1.5 vorgestellten Performancemaße ausgewertet werden. Dafür wurden die in `sklearn.metrics` bereitgestellten Funktionen `accuracy_score()`, `precision_score()`, `recall_score()` und `f1_score()` verwendet. Um einen Überblick zu erhalten, wurden die Precision, der Recall und das F1-Maß sowohl mit der Methode des Micro-Averaging als auch mit der des Macro-Averaging berechnet. Die gerundeten Ergebnisse sind in Tabelle 5-2 dargestellt. Die Abkürzung HK im Tabellenkopf steht dabei für Hilbertkurve und soll ausdrücken, dass die Methode mit vorheriger Sortierung entlang der Hilbertkurve ausgeführt wurde. Die Abkürzungen 2D und 3D werden hier stellvertretend für die Digramm- und Trigramm-Analyse verwendet.

Tabelle 5-2: Performancemaße der verschiedenen Methoden

	2D	2D & HK	3D	3D & HK
Accuracy	0.69214	0.62909	0.99342	0.98026
TP + TN	8903	8092	151	149
FP + FN	3960	4771	1	3
Micro-Precision	0.69214	0.62909	0.99342	0.98026
Micro-Recall	0.69214	0.62909	0.99342	0.98026
Micro-F1-Maß	0.69214	0.62909	0.99342	0.98026
Macro-Precision	0.73963	0.68262	0.99567	0.98148
Macro-Recall	0.66442	0.57065	0.98667	0.96895
Macro-F1-Maß	0.67007	0.58427	0.99102	0.97437

Bei Betrachtung der errechneten Werte fällt auf, dass alle Werte, die über das Micro-Averaging berechnet wurden, identisch sind mit dem jeweiligem Wert für die Accuracy. Dies sollte bei korrekter Berechnung nicht der Fall sein. Ein Blick in die Dokumentation der verwendeten Python-Bibliothek scikit-learn [22, 65] zeigt, dass die Funktionen für das Micro-Averaging tatsächlich gleich berechnet werden. Eine Bewertung anhand der mit Micro gekennzeichneten Performancemaße ist also nicht aussagekräftig, sodass nur die im Macro-Averaging berechneten Maße betrachtet werden.

Sowohl bei der Digramm- als auch bei der Trigramm-Analyse scheint der Klassifikator insgesamt bessere Ergebnisse zu erzielen, wenn die Hilbertkurve nicht beachtet wird. In der Digramm-Darstellung wurden hier gut zwei Drittel der Sektoren korrekt zugeordnet. Dies ist kein sehr gutes Ergebnis, zeigt aber auch, dass die verwendeten Ansätze trotzdem teilweise funktionieren.

Da die Datenbasis für die Erstellung der Test- und Trainingsdaten mehrfach angepasst wurde, um eine Klassifikation mit der Trigramm-Analyse zu ermöglichen, wurde die Digramm-Methode mit unterschiedlich großen Datenbasen durchgeführt. Dabei wurde festgestellt, dass eine größere Menge an Dateien in der Datenbasis tendenziell zu besseren Ergebnissen führt. Beispielsweise wurden bei einer Datenbasis von 13,4 MB nur eine Genauigkeit von gerundet 0.56 erreicht. Dies ist deutlich geringer, als die in Tabelle 5-2 gezeigten Werte, welche mittels Digramm-Analyse auf einer Datenbasis von 31,3 MB erreicht wurden. Es ist somit anzunehmen, dass der Klassifikator mit einer größeren Menge Trainingsdaten noch bessere Ergebnisse erzielen würde. Eine Garantie besteht jedoch nicht.

Die Trigramm-Methode schneidet in Tabelle 5-2 dagegen sehr gut ab. Dies ist auch auf die vergleichsweise geringe Datenmenge zurückzuführen und somit nicht mit den Ergebnissen der Digramm-Analyse vergleichbar. Bei nur drei zur Verfügung stehenden Klassen ist die Wahrscheinlichkeit höher, dass der Klassifikator die richtige Klasse vorhersagt als bei 16 zu erkennenden Dateitypen. Wie diese Methode bei Verwendung einer größeren Datenbasis tatsächlich abschneiden würde, lässt sich jedoch nicht prognostizieren.

Für die weitere Auswertung der Ergebnisse wurde mit der Funktion `confusion_matrix()` aus dem Package `sklearn.metrics` für jede Methode eine Konfusionsmatrix erstellt. Die Matrizen für die Digramm-Analyse sind aufgrund ihrer Größe nicht im Fließtext enthalten, sondern wurden in Anhang D eingefügt. Die Matrizen für die Trigramm-Analyse sind in den Tabellen 5-3 und 5-4 dargestellt.

Tabelle 5-3: Konfusionsmatrix für die Trigramm-Analyse ohne Anwendung der Hilbertkurve

cc \ pc	exe	jpg	txt
exe	24	1	0
jpg	0	76	0
txt	0	0	51

Tabelle 5-4: Konfusionsmatrix für die Trigramm-Analyse mit Anwendung der Hilbertkurve

cc \ pc	exe	jpg	txt
exe	23	0	2
jpg	0	75	1
txt	0	0	51

Anhand der Konfusionsmatrizen für die Digramm-Methode (Tabellen D-1 und D-2) lässt sich erkennen, dass einige Dateitypen genauer klassifiziert wurden als andere. Dies ist vergleichbar mit den Ergebnissen der visuellen Bewertung und zeigt, dass einige Dateitypen eher für diese Methode geeignet sind als andere. Beispielsweise wurden mp4-, svg- und xml-Sektoren deutlich seltener falsch klassifiziert als andere. Im Gegensatz dazu wurden apk- und elf-Sektoren relativ häufig nicht als solche erkannt. Zudem fällt auf, dass generell sehr viele Sektoren den Dateitypen apk, mp4 und pdf zugeordnet wurden. Hier besteht also die Chance durch gezielte Auswahl einiger Dateitypen die Klassifikationsergebnisse zu verbessern.

Auf diese Art ließen sich auch die guten Ergebnisse bei der Klassifizierung mittels Trigramm-Analyse erklären. Die xml- und auch jpg-Sektoren werden unter Beachtung aller 16 Dateitypen bereits sehr gut erkannt. Auch exe fällt hier nicht negativ auf. Zudem konnten bei der visuellen Bewertung in Abschnitt 5.3.3 bei diesen drei Typen bereits Unterschiede in den Visualisierungen Digramm- und Trigramm-Analyse festgestellt werden.

Zusätzlich ist anzumerken, dass die Durchführung der Klassifikation auf Basis der Trigramm-Analyse sehr lang dauert, wenn vorher noch Test- und Trainingsdaten erzeugt werden müssen. Auch wenn mit der Trigramm-Analyse im Rahmen der vorliegenden Arbeit keine Klassifikation über alle Dateitypen vorgenommen werden konnte und somit keine Angaben über die tatsächliche benötigte Zeit für die Generierung von Test- und Trainingsdaten gemacht werden kann, ist eine kleine Hochrechnung möglich. Die Durchführung der Trigramm-Analyse ohne Hilbert-Umformung dauerte bei hiesigen Versuchen für einen Block circa 20 Sekunden. Mit einer einfachen Hochrechnung lässt sich feststellen, dass die Analysezeit für ein Speichermedium mit 20 GB bei Anwendung dieser Methode über 25 Jahre betragen würde. Eine Ausführungszeit, wie sie aktuell gegeben ist, ist also praktisch nicht nutzbar, selbst wenn für die Ausführung ausreichender Speicherplatz vorhanden ist. Bei der Digramm-Analyse konnten 25 MB unter Verwendung von vier CPU-Kernen in circa 15 Minuten, bei nur einem Kern in circa 75 Minuten zu Test- und Trainingsdaten verarbeitet werden. Die Methode ist somit praktisch eher einsetzbar.

6 Demonstrator

Die implementierten Funktionen wurden zu einem Demonstrator zusammengeführt. Dieser soll helfen die Ergebnisse nachzuvollziehen und gegebenenfalls zu verifizieren. Zudem kann er als Grundlage für weitere Forschungen zu dem Thema verwendet werden. Im finalen Kapitel 7 werden Ideen und Fragestellungen vorgestellt, wie der Demonstrator verwendet und weiterentwickelt werden kann. Zunächst erfolgt jedoch eine kurze Einführung in die Benutzung des Demonstrators.

Um die Funktionalität des Demonstrators vollumfänglich zu gewährleisten, sollte sichergestellt werden, dass die in Tabelle 4-1 gelisteten Packages installiert wurden. Zu diesem Zweck wurde auch eine Datei requirements.txt angelegt, anhand welcher mit dem Python-Befehl `pip` alle nötigen Module installiert werden können.

Der Demonstrator umfasst im wesentlich drei Funktionen, welche mittels verschiedener Parameter-Konfigurationen aufgerufen werden können. Dafür kann ein beliebiges, auf dem Host-PC vorhandenes, Kommandozeilentool verwendet werden. Aufzurufen ist dabei immer die Datei `demo.py` mit `python3`. Die einzelnen Funktionen und die jeweils benötigten Parameter werden im Folgenden genauer vorgestellt.

Mit dem Parameter `-v` lassen sich Visualisierungen erzeugen, wie in Quelltext 6.1 angegeben. Dafür werden alle Dateien, die im als `inputPath` übergebenen Ordner liegen, verarbeitet und die Visualisierungen als png-Datei im Ordner `outputPath` abgelegt. Der Parameter `-m` kann dabei genutzt werden, um zwischen der Digramm- (`2d`) und der Trigramm-Methode (`3d`) oder der Methode zum Erzeugen eines HSL-Bildes (`im`) zu wählen. Wird `-m` nicht angegeben, wird standardmäßig die Digramm-Methode ausgeführt. Mit `-hc` lässt sich zusätzlich angeben, ob die ausgewählte Methode zusammen mit der Hilbertkurve angewendet werden soll. Sollen nur Teile einer Datei visualisiert werden, kann mit `-n` die Anzahl der maximal zu bearbeitenden Sektoren pro Datei festgelegt werden. Über `-s` kann mit den Werten `True` oder `False` angegeben werden, ob die Visualisierungen zusätzlich zum Speichern auch angezeigt werden sollen. Dabei ist der Standardwert `True`, wenn mit `-n` eine maximale Sektoranzahl festgelegt wurde und `False`, wenn `-n` nicht angegeben wird.

```
1 python3 demo.py -v [-n numberOfSectors] [-s true|false] [-m {2d,3d,im}]
    [-hc] inputPath outputPath
```

Quelltext 6.1: Aufruf des Demonstrators zur Erstellung von Visualisierungen

Werden mit diesem Befehl entweder Digramm- oder Trigramm-Analysen visualisiert, so öffnet sich bei `-s True` eine dynamische Ansicht. In dieser kann mittels des Lupensymbols in der unteren linken Ecke zum Beispiel in das Diagramm hereingezoomt werden. Bei der Trigramm-Darstellung lässt sich das Diagramm rotieren und erlaubt so einen Blick aus verschiedenen Perspektiven. Über das Haus-Symbol gelangt der Nutzer jederzeit zurück auf die Kombination aus Zoom und Ausrichtung, in der das Diagramm ursprünglich geöffnet wurde.

Diese Darstellungen werden standardmäßig im Vollbildmodus geöffnet, damit sich die Auflösung an den jeweils verwendeten Monitor anpassen kann. Wird der Vollbildmodus nicht genutzt, kann es passieren, dass die Diagramme auf manchen Bildschirmen leer erscheinen. Die Punkte werden hier

erst beim Hereinzoomen sichtbar. Um den Vollbildmodus wieder zu verlassen oder auch wieder zu aktivieren, kann jederzeit die Taste `f` auf der Tastatur gedrückt werden. Über die Taste `s` lässt sich die aktuelle Ansicht des Diagramms als Bilddatei speichern. Dies kann praktisch sein, wenn ein Muster erst nach Vergrößerung oder/und Rotation des angezeigten Diagrammausschnitts sichtbar wird und dokumentiert werden soll. Mit der Taste `q` lässt sich ein Diagramm, ebenso wie mit dem klassischen Kreuz in der - je nach Betriebssystem - oberen linken oder rechten Ecke, schließen. Hier ist zudem anzumerken, dass das nächste Digramm oder Trigramm erst geladen wird, wenn die Darstellung des vorherigen geschlossen wurde. Wurde dagegen die Methode der HSL-Bilder gewählt, so wird der Algorithmus fortlaufend abgearbeitet und die Bilder öffnen sich übereinander, solange `-s True` gesetzt ist.

Eine weitere Funktion des Demonstrators liegt in der Erzeugung von Test- und Trainingsdaten auf Basis der Digramm- und Trigramm-Methode. Diese Funktion wird über den Parameter `-t` gestartet. Dabei muss die in Quelltext 6.2 gezeigte Syntax eingehalten werden.

```
1 python3 demo.py -t [-m {2d,3d}] [-hc] inputPath outputPath
```

Quelltext 6.2: Aufruf des Demonstrators zum Generieren von Test- und Trainingsdaten

Zur Erzeugung von Trainingsdaten sind wenige Parameter nötig. Die Parameter `-m` und `-hc` können analog verwendet werden, mit dem Unterschied, dass die Option `-m im` für HSL-Bilder nicht mehr zur Auswahl steht. Dafür benötigt der `inputPath` ein wenig mehr Vorbereitung als zuvor, da eine feste Struktur eingehalten werden muss. Konkret sind im angegebenen Ordner 16 Unterordner zu erstellen, die entsprechend der zu klassifizierenden Dateitypen mit `apk`, `bmp`, ..., `xml` zu benennen sind. In jedem der Ordner muss mindestens eine Datei des jeweiligen Dateityps vorliegen. Zusätzlich muss der als `outputPath` angegebene Ordner leer oder nicht nicht vorhanden sein, wobei er neu erzeugt wird.

Die Generierung der Trainings- und Testdaten kann einige Zeit in Anspruch nehmen. Um anzuzeigen, dass der Prozess noch läuft, wird jedes Mal, wenn eine neue Datei eingelesen wird, eine entsprechende Meldung an den Nutzer ausgegeben. Sollte der Computer über ausreichend CPU-Kerne verfügen, werden bis zu vier Dateien gleichzeitig verarbeitet um die Rechenzeit zu verkürzen. Bei Verwendung des Modus `-m 3d` ist es wahrscheinlich, dass das Programm nicht korrekt ausgeführt werden kann, da der Arbeitsspeicherbedarf zu groß wird.

Wurden die Trainings- und Testdaten für alle Dateitypen korrekt erzeugt, wird ein Klassifikator erstellt und mit den soeben erzeugten Trainingsdaten trainiert. Anschließend werden die erzeugten Testdaten probeweise klassifiziert und die in Abschitt 5.4.3 bereits verwendeten Performancemaße berechnet und dem Nutzer angezeigt. Zusätzlich werden die errechneten Werte zusammen mit einer Konfusionsmatrix der Probe-Klassifikation in die Datei `performance.txt` geschrieben, welche zusammen mit den Test- und Trainingsdaten in dem als `outputPath` angegebenen Ordner abgelegt wird. So erhält der Nutzer einen ersten Überblick über die Güte der ausgewählten Dateien und kann diese bei Bedarf noch einmal anpassen, um gegebenenfalls bessere Klassifikationsergebnisse zu erhalten.

Sollen diese Trainingsdaten nun zur Klassifikation anderer Dateiinhalte verwendet werden, ist der Parameter `-c` zu verwenden, wie in Quelltext 6.3 gezeigt. Auch hier kann über `-m` und `-hc` der gewünschte Algorithmus ausgewählt werden. Wichtig ist dabei, dass die Parameter genauso gesetzt

werden, wie zuvor bei der Erzeugung der Trainingsdaten, deren Ordnerpfad als `inputPath` anzugeben ist.Der Parameter `-d` erwartet einen Ordnerpfad, an welchem die zu klassifizierenden Dateien liegen. Als Ergebnis der Klassifikation wird im Ordner `outputPath` für jede klassifizierte Datei eine neue Textdatei erstellt. Diese enthält in jeder Zeile zunächst die Nummer des klassifizierten Sektors, gefolgt vom Klassifizierungsergebnis in Zahlenform und übersetzt in einen String mit dem Namen des Dateityps. Diese drei Werte sind jeweils mit Komma getrennt gelistet, wodurch eine automatisierte Weiterverarbeitung ermöglicht wird.

```
1 python3 demo.py -c [-m {2d,3d}] [-hc] -d dataPath inputPath outputPath
```

Quelltext 6.3: Aufruf des Demonstrators zum Klassifizieren

Zudem lässt sich mit den Parametern `-h` und `-help` eine Hilfeseite aufrufen, auf der alle Parameter gelistet und kurz erklärt sind. Außerdem werden dem Nutzer die zu verwendende Syntax sowie einige Hinweise zur Funktionsweise des Programms angezeigt. Die Paramter `inputPath`, `outputPath` und eine der Optionen `-v`, `-t` oder `-c` müssen immer angegeben werden, `-m` und `-hc` sind dagegen optional, funktionieren aber mit allen drei Methoden. Die anderen drei optionalen Parameter `-n`, `-s` und `-d` werden dagegen nur beim Visualisieren bzw. Klassifizieren vom Programm verwendet. Wird einer dieser Parameter außerhalb der vorgesehenen Funktion aufgerufen, so wird er vom Programm ignoriert.

Auch wenn Trainingsdaten gewählt wurden, mit denen der entwickelte Klassifikator recht genau klassifiziert, sind die Ergebnisse nicht immer zutreffend. Es ist daher unerlässlich, dass die Klassifikationsergebnisse nur als Vorschlag angesehen werden und vor der Weiterverwendung immer auch eine Fehlklassifikation einzelner Sektoren in Betracht gezogen wird. Nachdem die automatisierte Bewertung erfolgt ist, sollten die Ergebnisse daher noch von einem Menschen, zumindest stichprobenweise, betrachtet und ausgewertet werden [72].

7 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden die im Kapitel 2.4 vorgestellten Methoden zur visuellen Analyse von Dateien aufgegriffen und an die Analyse von Sektoren angepasst. Damit sollte erforscht werden, ob diese Methoden auch zur Bestimmung von Dateitypen geeignet sind, wenn nur kleine Ausschnitte des Datenstreams betrachtet werden.

Dafür wurden diese und weitere Ansätze in Python implementiert, wie in Kapitel 5 beschrieben. Der in Kapitel 6 vorgestellte Demonstrator kann die bearbeiteten Methoden graphisch darstellen und abspeichern. Die graphischen Darstellungen wurden visuell bewertet, um so einzelne Methoden für eine statistische Bewertung auszuwählen. Anschließend wurde ein Naive-Bayes-Klassifikator implementiert, welcher anhand der ausgewählten Methoden die Dateitypen einzelner Sektoren erkennt. Dieser betrachtet die Sektoren jeweils als Ganzes, sodass charakteristische Elemente, wie beispielsweise spezifische Signaturen, nicht explizit ausgewertet werden. Somit wurde die Zielstellung dieser Arbeit grundsätzlich erreicht.

Trotzdem ist die im Klassifikationsprozess erzielte Genauigkeit bei der Digramm-Methode noch ausbaufähig. Die Klassifikation mittels Trigramm-Methode konnte wegen Performanceproblemen nicht ausreichend getestet werden. In einem Minimalbeispiel konnte die Funktionsweise des Klassifikators für Trigramme bestätigt werden, jedoch ist dieser nur experimentell nutzbar. Die in Abschnitt 5.4.3 Ergebnis vorgestellten Ergebnisse für die Digramm- und Trigrammanalyse sind aufgrund der unterschiedlichen Datenbasen nicht vergleichbar. Zur weiteren Erforschung der Eignung und Vorzüge der beiden Methoden sind weitere Tests in einer leistungsfähigen Testumgebung erforderlich. Die Methoden Bild (siehe Abschnitte 5.2.1 und 5.3.3.1) und zusammenhängende Regionen (siehe Abschnitte 5.2.4 und 5.3.3.4) haben sich als ungeeignet herausgestellt.

Sollte keine leistungsfähigere Testumgebung zur Verfügung stehen, so kann alternativ die Performance des Demonstrators verbessert werden. Dafür sollte insbesondere der Rechen- und Speicheraufwand der verwendeten Methoden stark verbessert werden. Dies würde unter anderem die Skalierbarkeit des Programms verbessern, sodass größere Mengen an Trainingsdaten verarbeitet werden können. Zudem würde dies das Hinzufügen weiterer Dateitypen als Klassen erlauben. Ebenso könnte umgekehrt auch eine gröbere Aufteilung, zum Beispiel in die Klassen „Text, Bild, Ausführbare Dateien, Kryptographisches Material, Sonstige“ erfolgen. Während weitere Kategorien bei der Klassifikation die Gesamtmenge der Trainingsdaten und damit die Rechenzeit erhöhen würden, würden allgemeinere Klassen mehr verschiedene Trainingsdaten pro Klassen benötigen, was die Verwendung von Arrays zum Ablegen der Trainingsdaten unmöglich macht. Eine Möglichkeit zur Effizienzsteigerung liegt in der Verwendung einer anderen Programmiersprache. So könnte die Implementierung mit einer Low-Level-Sprache wie C++ die Performance des Programmes stark verbessern [58].

Um den Fokus noch mehr auf den Datenstream, also den Inhalt einer Datei, zu lenken, besteht zudem die Überlegung, keine vollständigen Dateien als Trainingsdaten zu nutzen. Würden die ersten und letzten Sektoren einer Datei vor der Verarbeitung entfernt, so könnte sich der Klassifikator mehr

auf Charakteristika des Datenstreams fokussieren, da Header und Footer tatsächlich keinen Einfluss mehr hätten. Es wäre dann zu erforschen, ob dieser Ansatz bessere Klassifikationsergebnisse produzieren würde.

Eine andere Möglichkeit, die Klassifikationsergebnisse zu verbessern liegt darin, die Anzahl der zu erkennenden Dateitypen zu reduzieren. Dabei sollten anhand der in Abschnitt 5.4.3 vorgestellten Ergebnisse jene Dateitypen ausgewählt werden, welche tendenziell gut erkannt wurden. Andere Dateitypen, die eher nicht erkannt wurden, könnten aus der Konfiguration des Klassifikators entfernt werden. Realistisch wäre eine Verringerung der Dateitypen allerdings nur dann, wenn bereits Vorwissen über die im zu analysierenden Datenbereich vorhandenen Dateien und Dateitypen besteht.

Zur Vereinfachung der Nutzung wäre die Implementierung einer Graphischen Benutzeroberfläche (GUI) von Vorteil. Diese könnte es dem Nutzer ermöglichen, das Klassifikationsergebnis für einen Sektor gleichzeitig in der visualisierten und in der Hexadezimal-Ansicht zu betrachten. Idealerweise erlaubt die Oberfläche auch das dynamische Umschalten zwischen den verschiedenen Analysemethoden und bietet so ein umfassendes Bild der zu analysierenden Daten. Eine solche Graphische Benutzeroberfläche (engl. Graphical User Interface) (GUI) könnte es dem Nutzer auch erlauben, die durch den automatisierten Klassifizierungsprozess bestimmten Klassenzuweisungen zu korrigieren, falls bei der händischen Auswertung Klassifikationsfehler auffallen sollten. Eine solche GUI könnte von einem Menschen darüber hinaus dazu verwendet werden, die für die verschiedenen Dateitypen charakteristischen Muster in der Visualisierung zu erkennen und zu erlernen. So wären mehr Menschen in der Lage, Datenausschnitte aufgrund von Visualisierungstechniken einem Dateityp zuzuordnen, falls eine automatisierte Klassifikation mal nicht möglich sein sollte.

Ein weiterer vielversprechender Ansatz liegt in der Auswertung von größeren Sektoren, da nicht alle Datenträger und Dateisysteme Daten mit einer Blockgröße von 512 Bytes ablegen. Die Option, zwischen der in der vorliegenden Arbeit angenommenen Sektorgröße von 512 Bytes und 4096 Bytes zu wechseln, würde den Klassifikationsprozess dynamischer gestalten. Es könnte auch angebracht sein, im Zuge dieser Erweiterung flexible Clustergrößen vorzusehen, falls die vorgestellten Methoden auf Datenstreams bekannter Clustergröße angewendet werden sollen. Eine Abschätzung, wie sich die Verarbeitung von größeren Sektoren auf die Rechenzeit des Klassifikators auswirkt, ist hier nicht möglich.

Denkbar wäre auch eine Anpassung der bei der Klassifikation zu betrachtenden Features. Während eine Extraktion weniger relevanter Punkte aus der Häufigkeitsmatrix in der Digramm- oder Trigramm-Analyse zur Verbesserung der Effizienz beitragen dürfte, wäre auch eine Erweiterung der Features um bisher nicht betrachtete Punkte denkbar, um die Klassifikationsgenauigkeit zu erhöhen. Neben einer Kombination der hier bereits vorgestellten Methoden wäre auch eine zusätzliche Auswertung der Entropie oder der Minimal- und Maximalwerte in einer Häufigkeitsmatrix möglich. Obwohl eine Beachtung dieser Werte im Rahmen der vorliegenden Arbeit angedacht war, ließ sich eine Umsetzung mit dem verwendeten Klassifikator nicht realisieren.

Darüber hinaus könnte eine Vorverarbeitung der Sektoren vor der Klassifikation zu einer verbesserten Effizienz oder auch Genauigkeit führen. Beispielsweise sind auch hier die Minimal- und Maximalwerte der Häufigkeitsmatrizen anzuführen. So wären diese geeignet um „leere“ , also nur mit `0x00` gefüllte, Sektoren herauszufiltern, bevor die Klassifikation beginnt. Dies ist ebenso für alle anderen Arten von Padding in einem Sektor möglich. Eine weitere häufig vorkommende Art von Sektoren, die vor

der Klassifizierung abgefangen werden könnte, sind die ausschließlich mit `0xFF` beschriebenen Sektoren. Beide Arten von Sektoren sollten anhand des Maximalwertes in der Häufigkeitsmatrix einfach zu identifizieren sein.

Sollten sich die im Demonstrator implementierten Methoden bei Verwendung von mehr Trainingsdaten als zuverlässiger erweisen, so können sie beispielsweise verwendet werden, um den Carving-Prozess zum Auffinden von gelöschten Daten zu verbessern. Klassischerweise wird beim Carving ein Datenträgerabbild nach dateitypspezifischen Magic Bytes gesucht und anhand dieser der Anfang und das Ende eine Datei bestimmt. Mit den in dieser Arbeit erarbeiteten Methoden besteht nun auch die Möglichkeit, Teile einer Datei zu identifizieren, zu denen keine Header und Footer (mehr) auffindbar sind. Zudem können gegebenenfalls verschleierte Daten gefunden werden, bei denen der Datenstream einer Datei mit einem Header und Footer eines anderen Dateityps präpariert wird, um Unwissende am Öffnen des Dateiinhalts zu hindern. In beiden Fällen könnte zu den analysierten Daten ein Header bzw. Footer des korrekten Dateityps rekonstruiert werden, um so die Daten schließlich wieder lesbar zu machen.

Außerdem kann die Klassifizierung von Sektoren in Dateitypen bei der Rekonstruktion von fragmentierten Dateien von Vorteil sein. In diesem Szenario gilt es, verschiedene Datenfragmente einer oder mehreren Dateien zuzuordnen. Dabei kann es von Vorteil sein, eine zusätzliche Methode zur Bestimmung von Dateitypen zur Hand zu haben. Dabei ist jedoch zu beachten, dass mehrere aufeinanderfolgende Sektoren gleichen Dateityps nicht zwingend zu ein und derselben Datei gehören müssen. Es ist ebenso möglich, dass an einer Stelle im Speicher Fragmente verschiedener Dateien gleichen Typs hintereinander abgelegt wurden [36]. Dies verdeutlicht erneut die Notwendigkeit, dass die vom Demonstrator oder einem anderen, auf Erkenntnissen dieser Arbeit aufbauenden, Klassifikator gegengeprüft werden sollten.

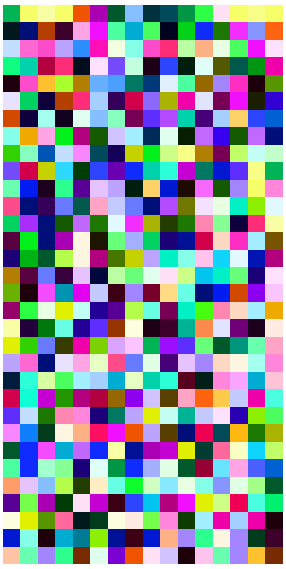
Anhang A: Bild-Darstellungen

Gemäß dem in Abschnitt 5.2.1 Bild beschriebenen Vorgehen wurden die ausgewählten Dateien zunächst als HSL-Bilder visualisiert. Diese sind in den Abbildungen A-1 bis A-16 dargestellt, wobei der jeweilige Dateiname der Original-Datei angegeben ist. Die Bilder sind begleitend zur visuellen Bewertung in Abschnitt 5.3.3.1 zu betrachten.

Datei 0003-apk.apk

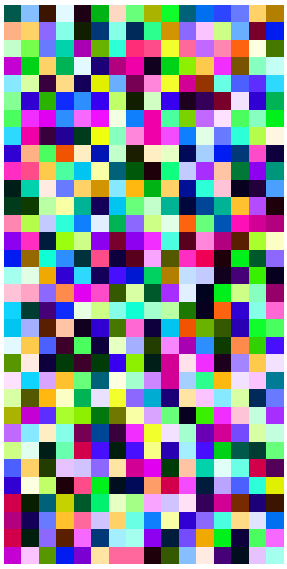
(a) Sektor 1

(b) Sektor 2



(c) Sektor 3

(d) Sektor 4



(e) Sektor 5

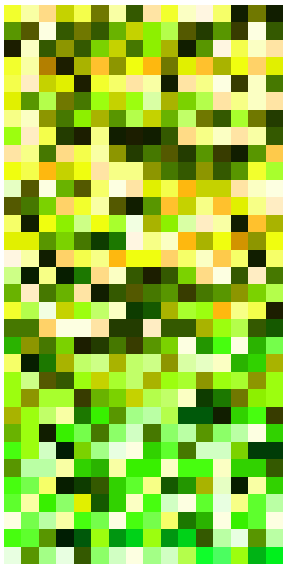
(f) Sektor 6

Abbildung A-1: Bild-Darstellung der Datei 0003-apk.apk

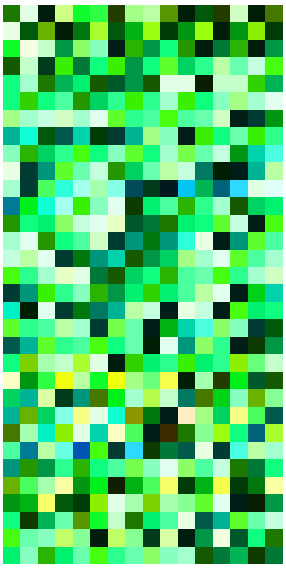
Datei 0003-bmp.bmp

(a) Sektor 1

(b) Sektor 2



(c) Sektor 3



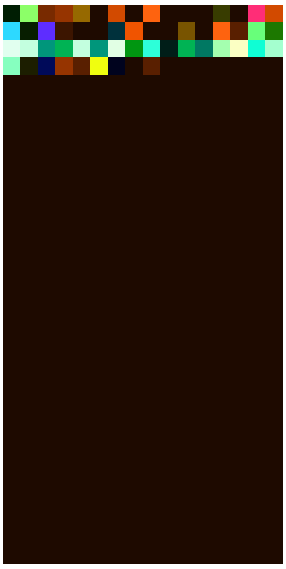
(d) Sektor 4

(e) Sektor 5

(f) Sektor 6

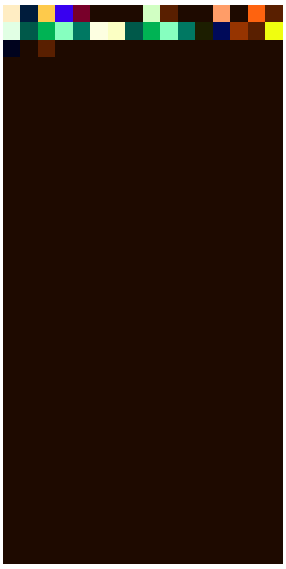
Abbildung A-2: Bild-Darstellung der Datei 0003-bmp.bmp

Datei 0003-docx.docx

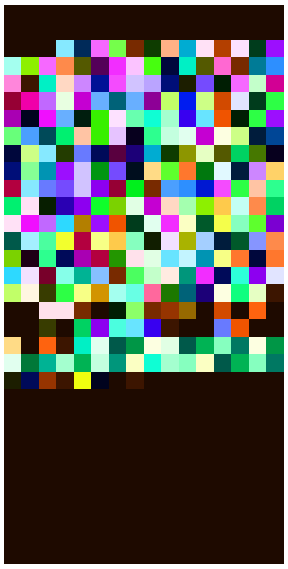


(a) Sektor 1

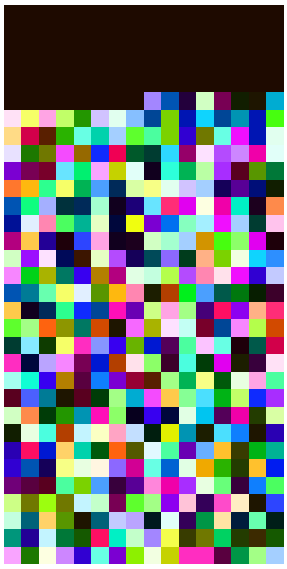
(b) Sektor 2



(c) Sektor 3



(d) Sektor 4

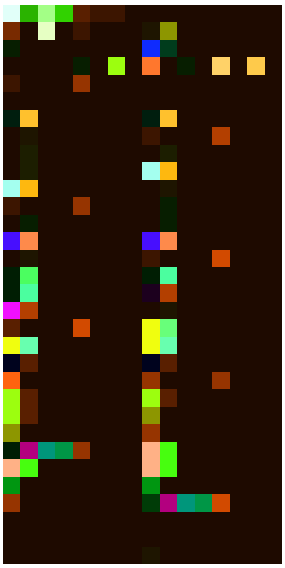


(e) Sektor 5

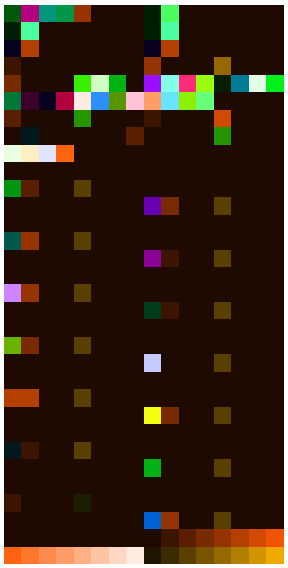
(f) Sektor 6

Abbildung A-3: Bild-Darstellung der Datei 0003-docx.docx

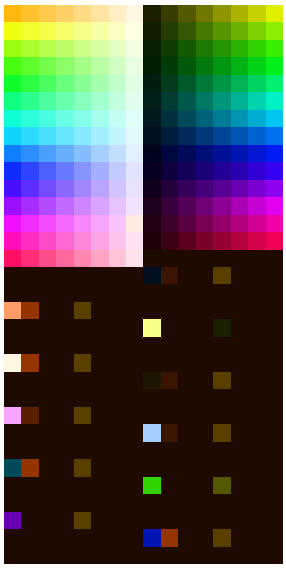
Datei 0003-so.so



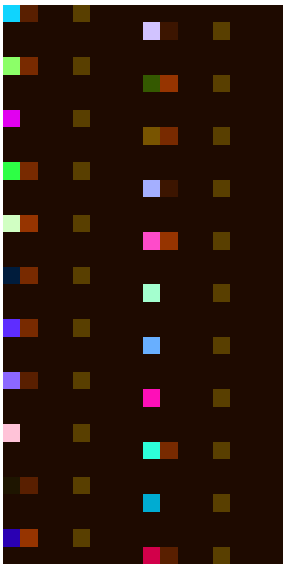
(a) Sektor 1



(b) Sektor 2

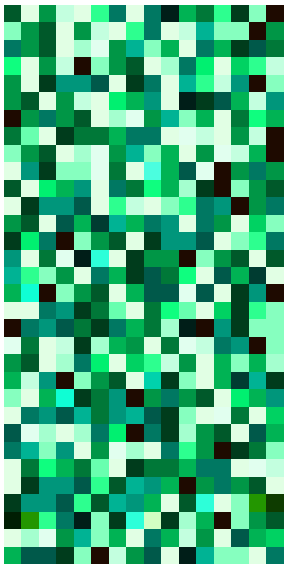


(c) Sektor 3



(d) Sektor 4

(e) Sektor 5

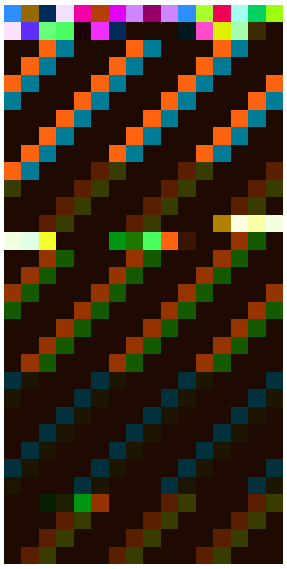


(f) Sektor 6

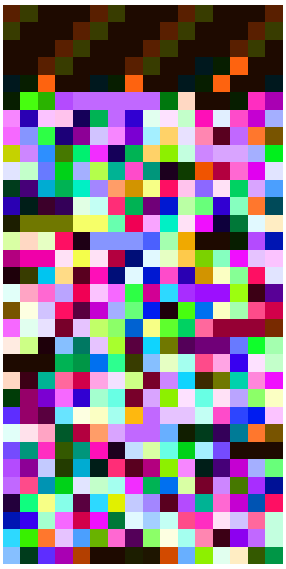
Abbildung A-4: Bild-Darstellung der Datei 0003-so.so

Datei 0003-epub.epub

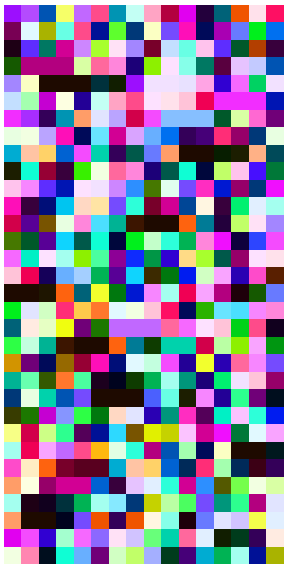
(a) Sektor 1



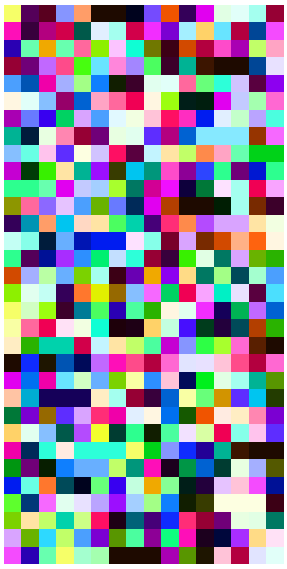
(b) Sektor 2



(c) Sektor 3



(d) Sektor 4

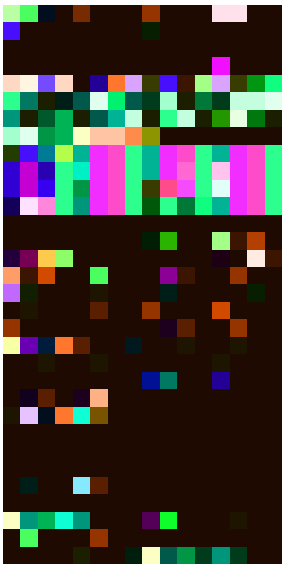


(e) Sektor 5

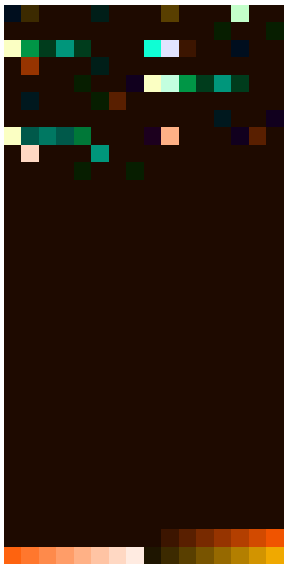
(f) Sektor 6

Abbildung A-5: Bild-Darstellung der Datei 0003-epub.epub

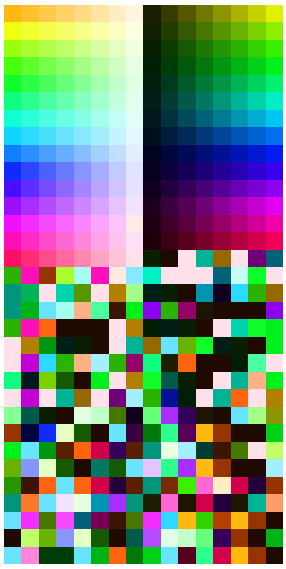
Datei 0003-exe.exe



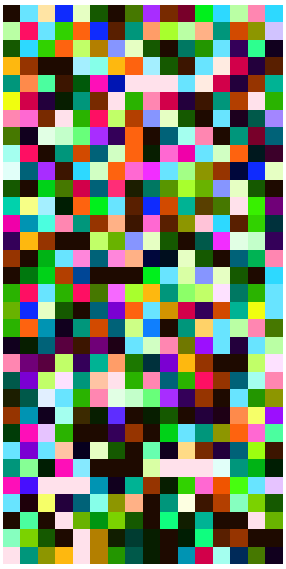
(a) Sektor 1



(b) Sektor 2



(c) Sektor 3



(d) Sektor 4

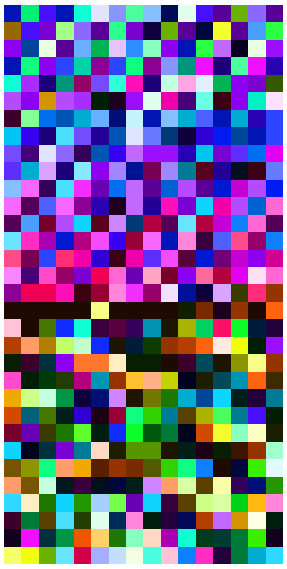
(e) Sektor 5

(f) Sektor 6

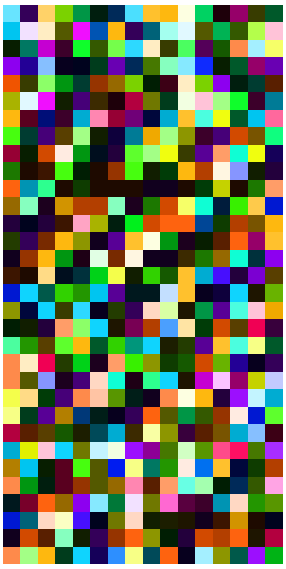
Abbildung A-6: Bild-Darstellung der Datei 0003-exe.exe

Datei 0003-gif.gif

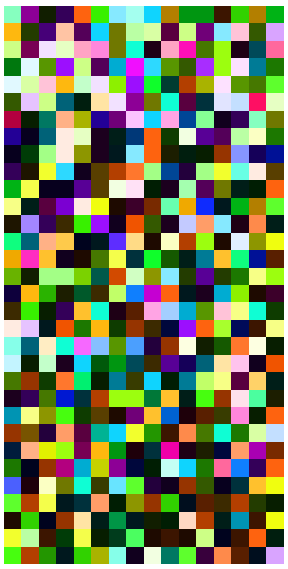
(a) Sektor 1



(b) Sektor 2



(c) Sektor 3



(d) Sektor 4

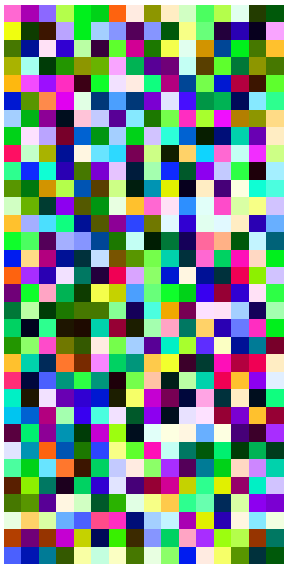
(e) Sektor 5

(f) Sektor 6

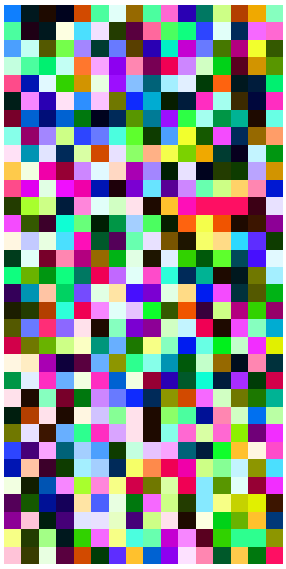
Abbildung A-7: Bild-Darstellung der Datei 0003-gif.gif

Datei 0003-jpg-q50.jpg

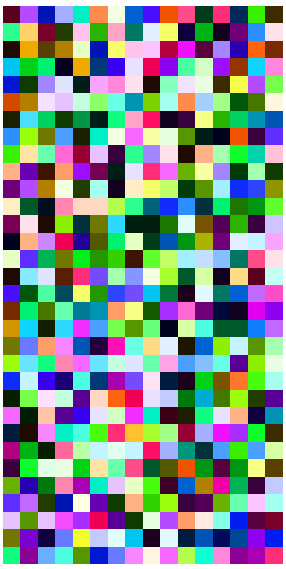
(a) Sektor 1



(b) Sektor 2

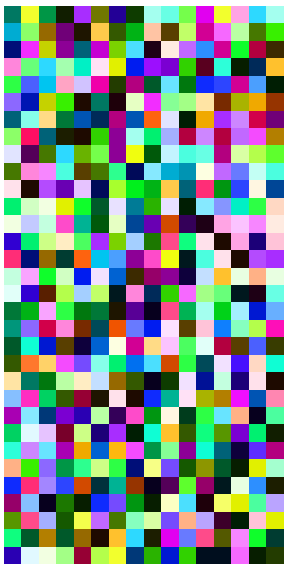


(c) Sektor 3



(d) Sektor 4

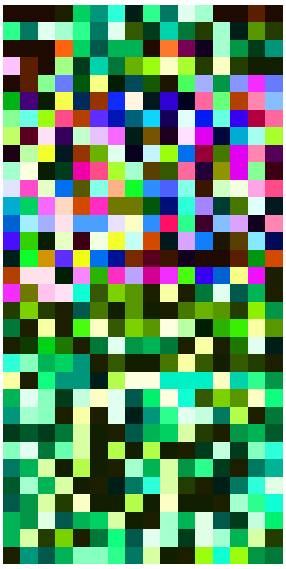
(e) Sektor 5



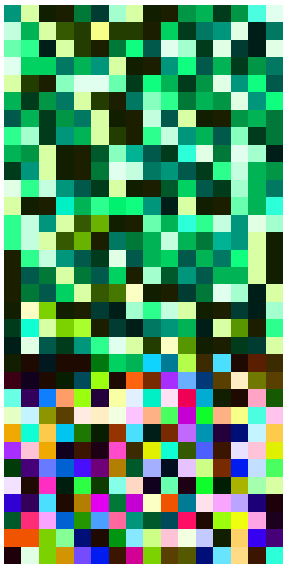
(f) Sektor 6

Abbildung A-8: Bild-Darstellung der Datei 0003-jpg-q50.jpg

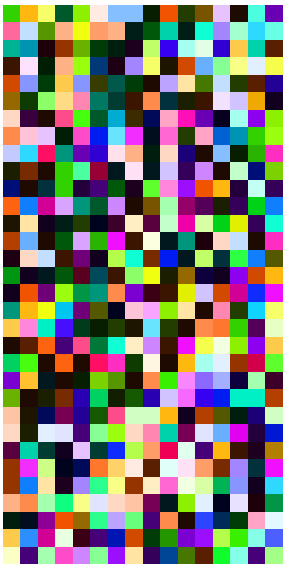
Datei 0003-mp4.mp4



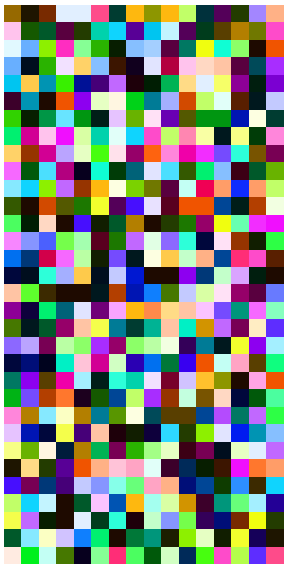
(a) Sektor 1



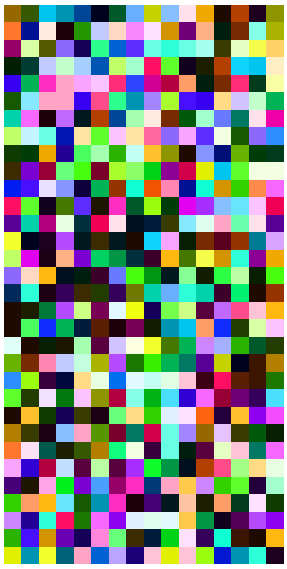
(b) Sektor 2



(c) Sektor 3



(d) Sektor 4



(e) Sektor 5

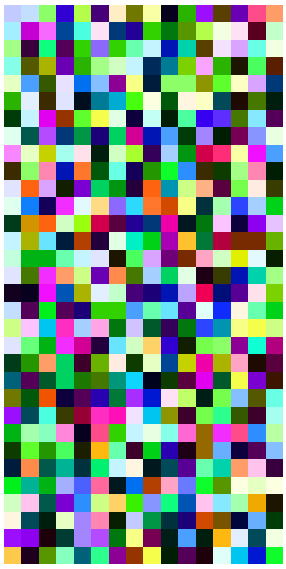
(f) Sektor 6

Abbildung A-9: Bild-Darstellung der Datei 0003-mp4.mp4

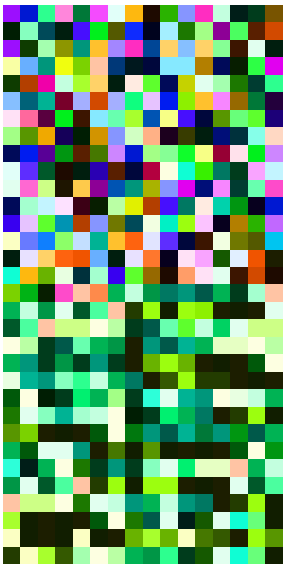
Datei 0003-pdf.pdf

(a) Sektor 1

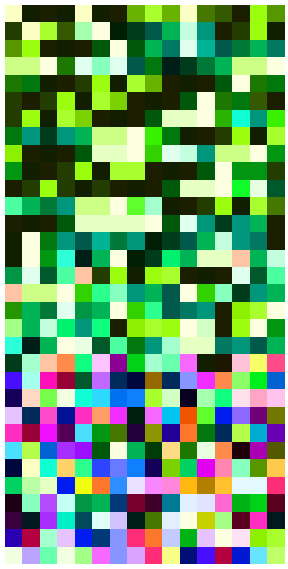
(b) Sektor 2



(c) Sektor 3



(d) Sektor 4



(e) Sektor 5

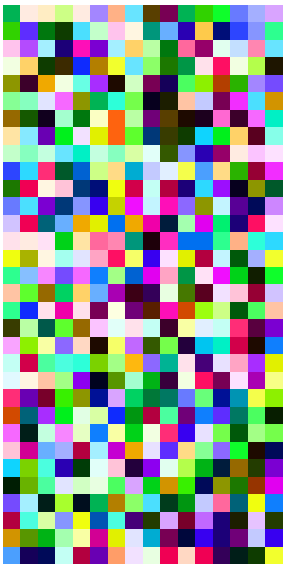
(f) Sektor 6

Abbildung A-10: Bild-Darstellung der Datei 0003-pdf.pdf

Datei 0003-png-c9.png

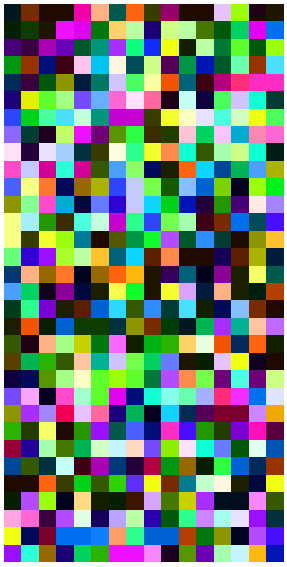
(a) Sektor 1

(b) Sektor 2



(c) Sektor 3

(d) Sektor 4

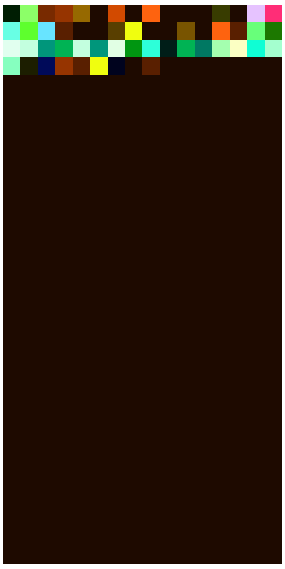


(e) Sektor 5

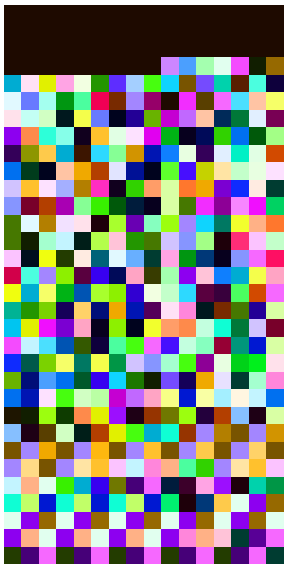
(f) Sektor 6

Abbildung A-11: Bild-Darstellung der Datei 0003-png-c9.png

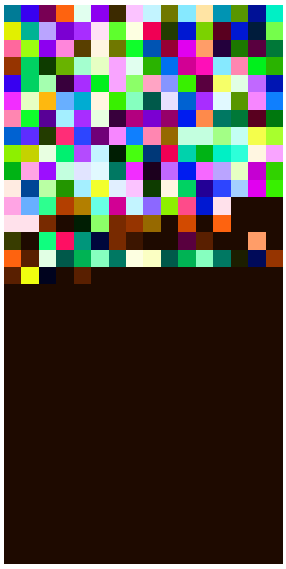
Datei 0003-pptx.pptx



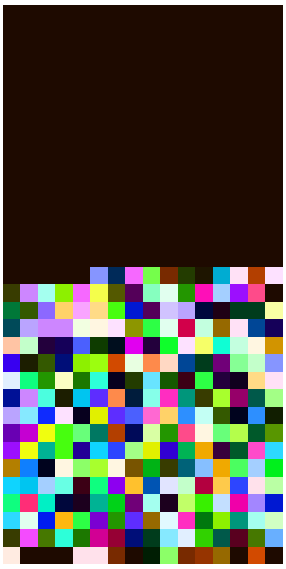
(a) Sektor 1



(b) Sektor 2



(c) Sektor 3



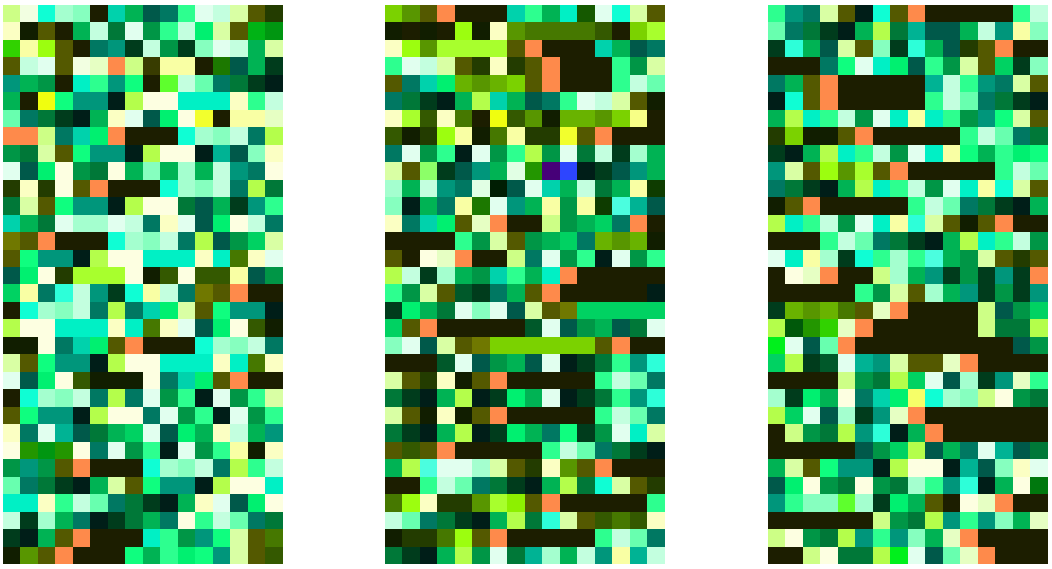
(d) Sektor 4

(e) Sektor 5

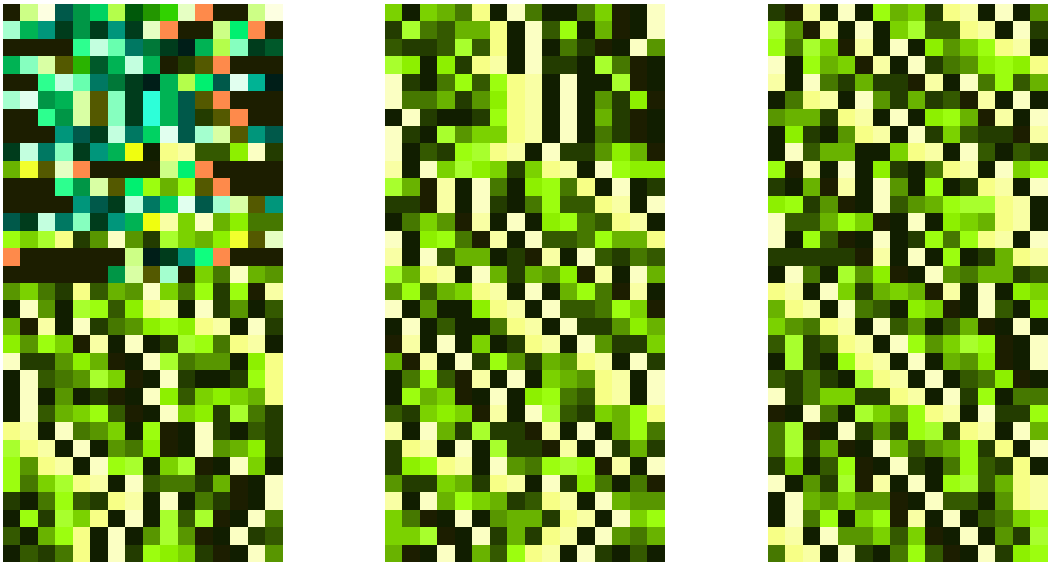
(f) Sektor 6

Abbildung A-12: Bild-Darstellung der Datei 0003-pptx.pptx

Datei 0003-svg-from-web.svg



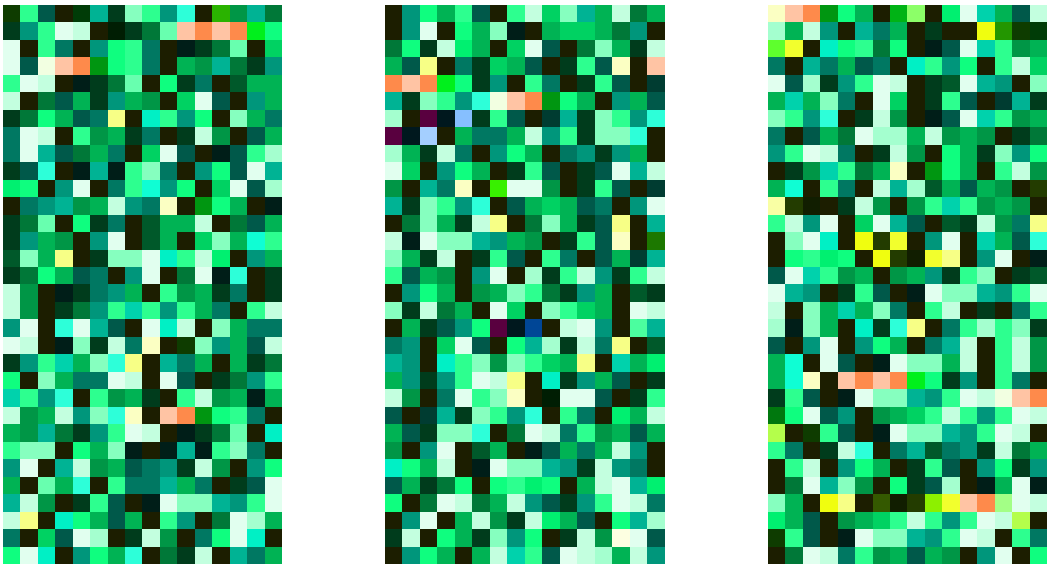
(a) Sektor 1 **(b)** Sektor 2 **(c)** Sektor 3



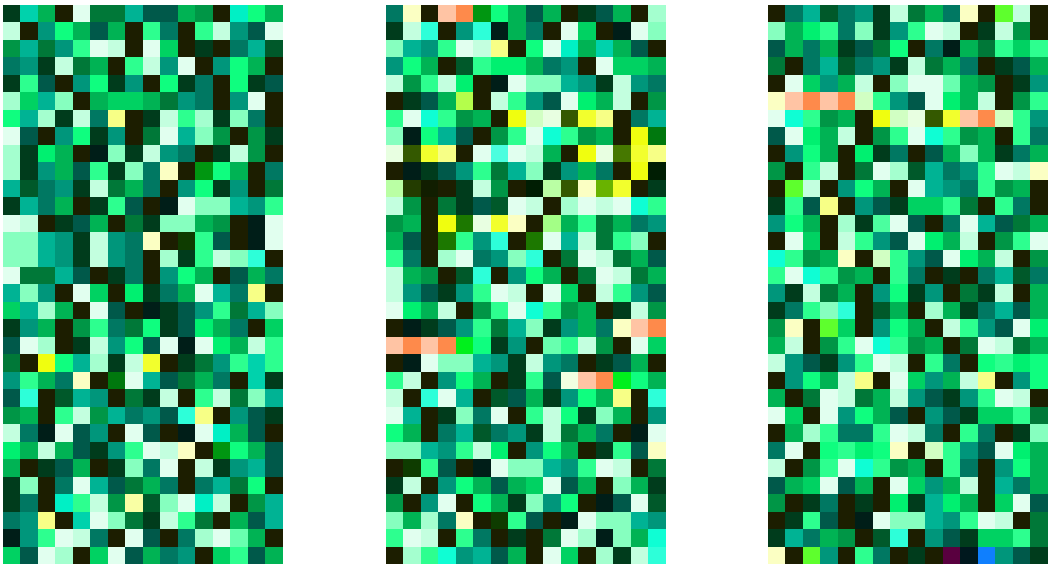
(d) Sektor 4 **(e)** Sektor 5 **(f)** Sektor 6

Abbildung A-13: Bild-Darstellung der Datei 0003-svg-from-web.svg

Datei 0003-txt.txt



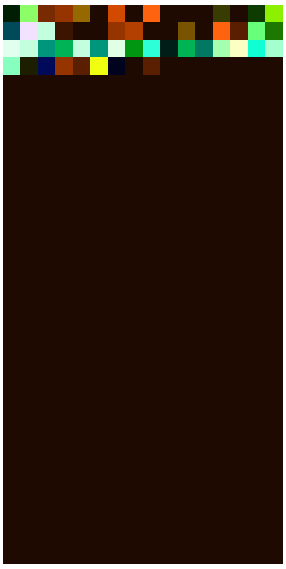
(a) Sektor 1 **(b)** Sektor 2 **(c)** Sektor 3



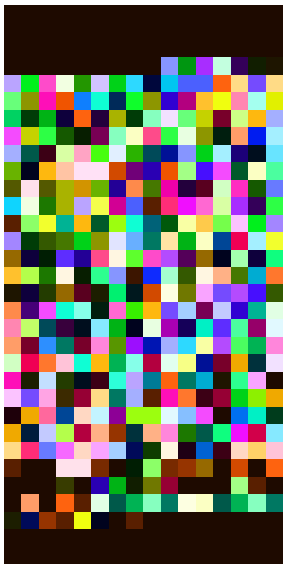
(d) Sektor 4 **(e)** Sektor 5 **(f)** Sektor 6

Abbildung A-14: Bild-Darstellung der Datei 0003-txt.txt

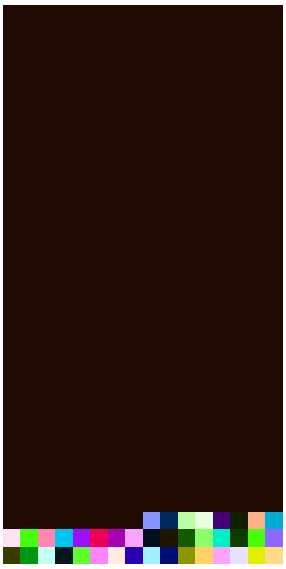
Datei 0003-xlsx.xlsx



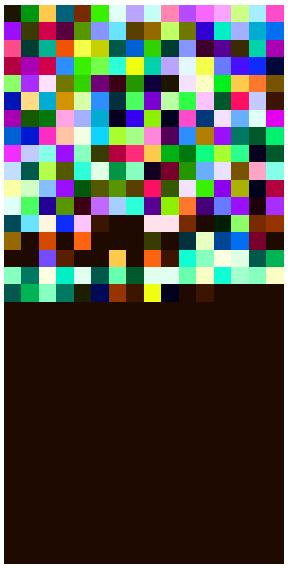
(a) Sektor 1



(b) Sektor 2



(c) Sektor 3



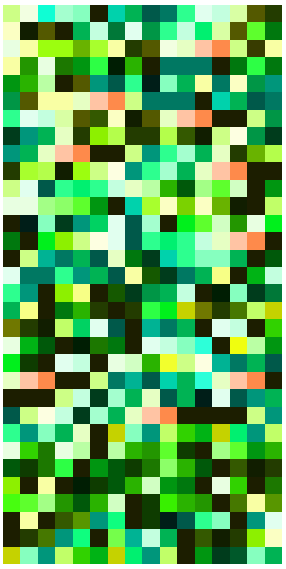
(d) Sektor 4

(e) Sektor 5

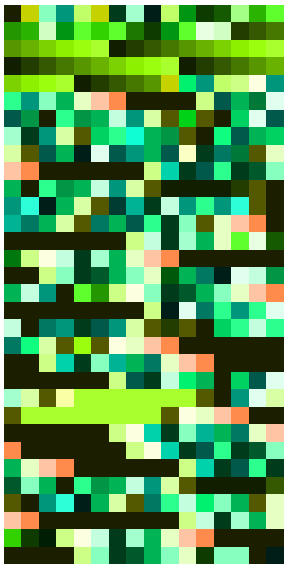
(f) Sektor 6

Abbildung A-15: Bild-Darstellung der Datei 0003-xlsx.xlsx

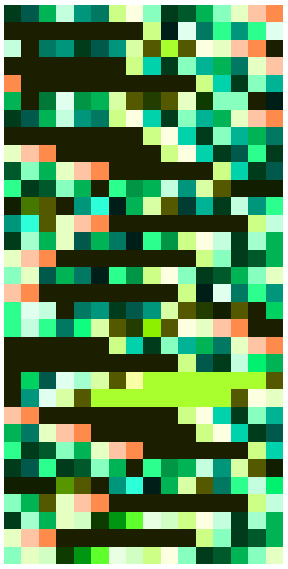
Datei 0003-xml.xml



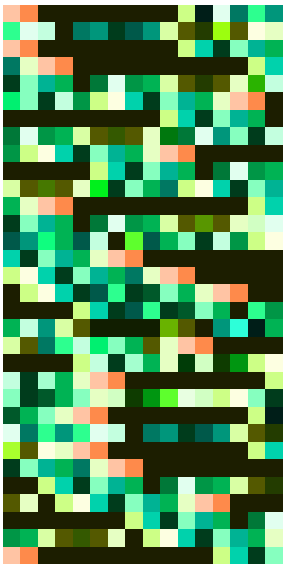
(a) Sektor 1



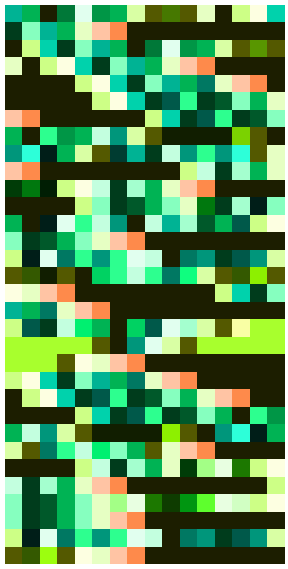
(b) Sektor 2



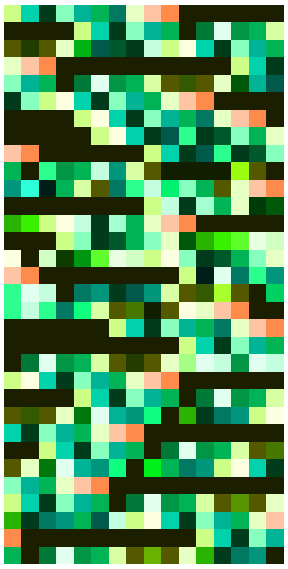
(c) Sektor 3



(d) Sektor 4



(e) Sektor 5



(f) Sektor 6

Abbildung A-16: Bild-Darstellung der Datei 0003-xml.xml

Anhang B: Digramm-Darstellungen

Nachfolgend sind die im Rahmen der visuellen Bewertung erstellten Visualisierungen der Digramm- und der Regionen-Analyse abgebildet. Die Abbildungen B-3 bis B-18 zeigen dabei für jeden Sektor die Digramm-Visualisierung auf der linken und die Regionen-Darstellung auf der rechten Seite. So können beide Visualisierungstechniken direkt verglichen werden. Die Bilder sind begleitend zur visuellen Bewertung in den Abschnitten 5.3.3.2 und 5.3.3.4 zu betrachten.

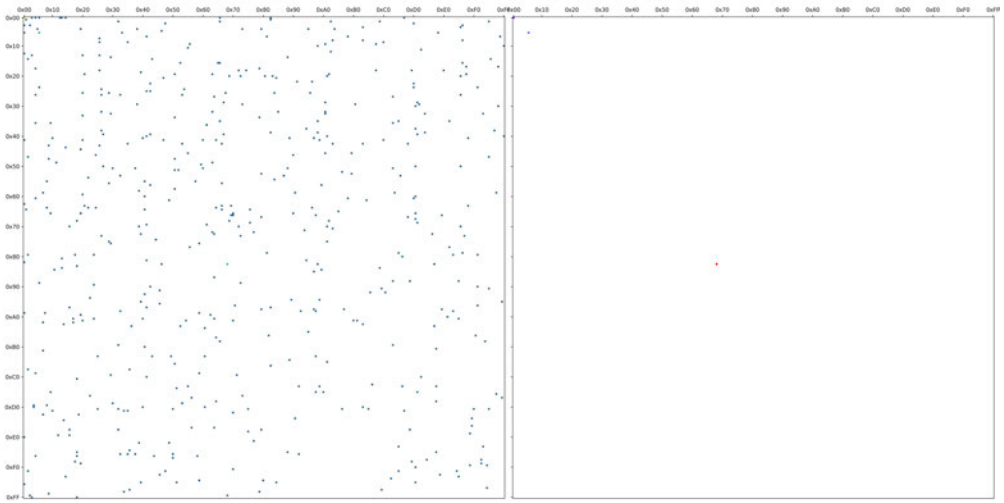
In den nachfolgenden Bildern sind die Punkte in der Digramm-Darstellung (Abbildungen B-3 bis B-18 links) entsprechend der Colormap `viridis` eingefärbt. Dabei sind kleine Werte durch dunkle Pixel dargestellt. Je häufiger ein Wert vorkommt, desto mehr nähert er sich einem gelben Farbton an. Abbildung B-1 zeigt den Farbverlauf der Colormap.

Abbildung B-1: Colormap `viridis`

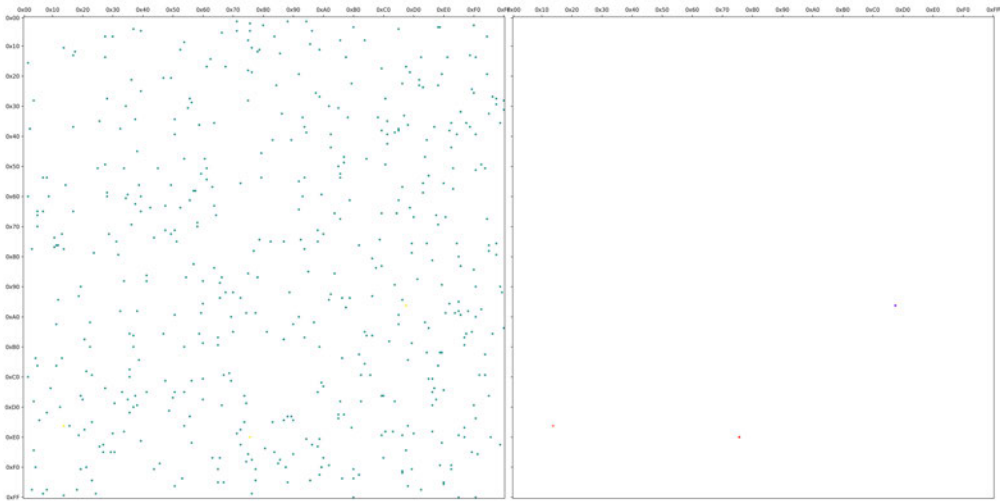
Für die Regionen-Darstellungen auf der jeweils rechten Seite der Abbildungen B-3 bis B-18 wurde dagegen die `rainbow` Colormap verwendet. Diese wurde ausgewählt, da sie alle Regionen gut sichtbar darstellt. Abbildung B-2 zeigt den Farbraum der Colormap.

Abbildung B-2: Colormap `rainbow`

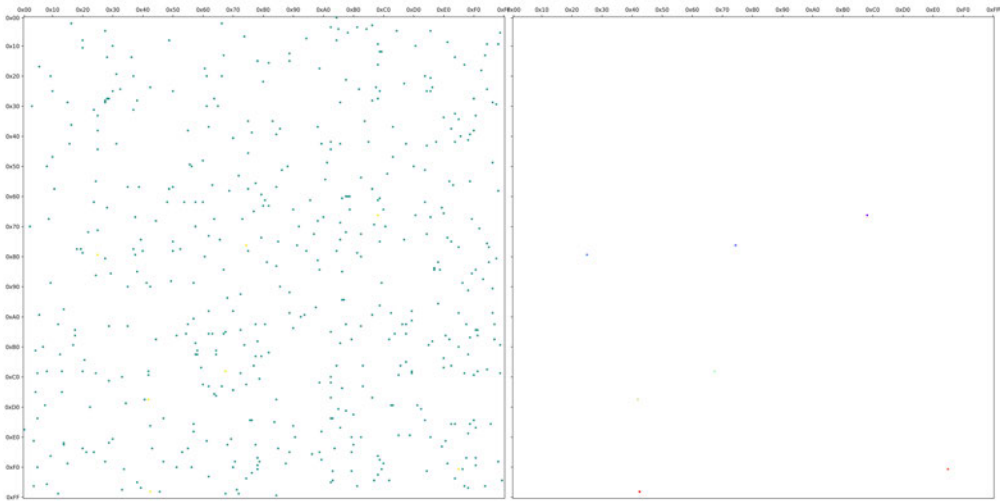
Datei 0003-apk.apk



(a) Sektor 1

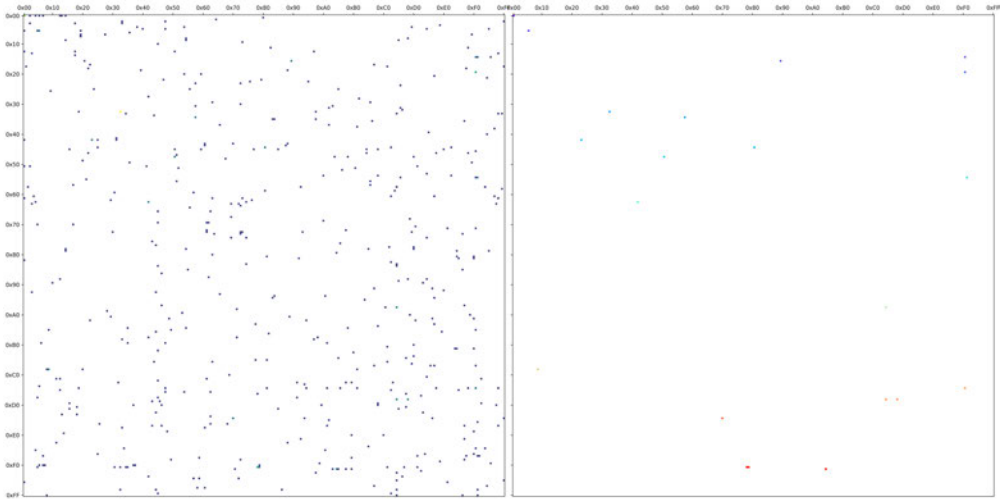


(b) Sektor 2

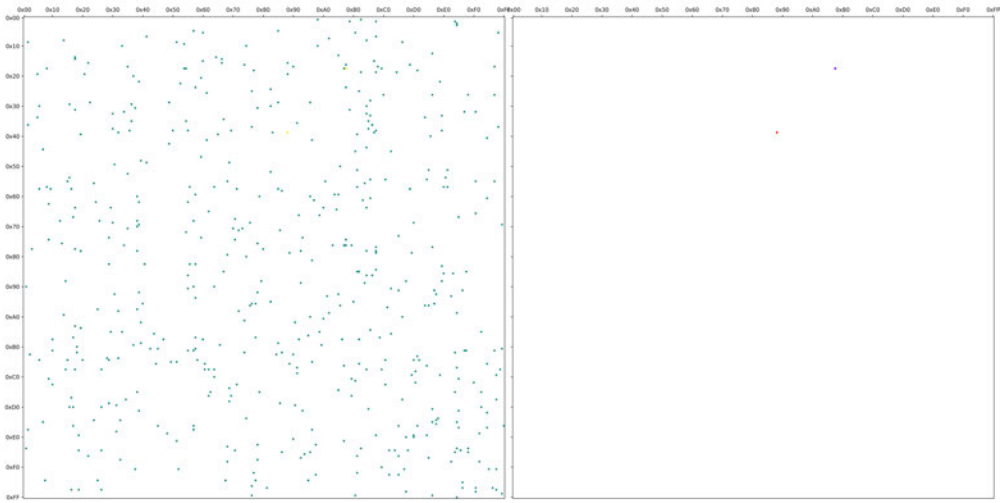


(c) Sektor 3

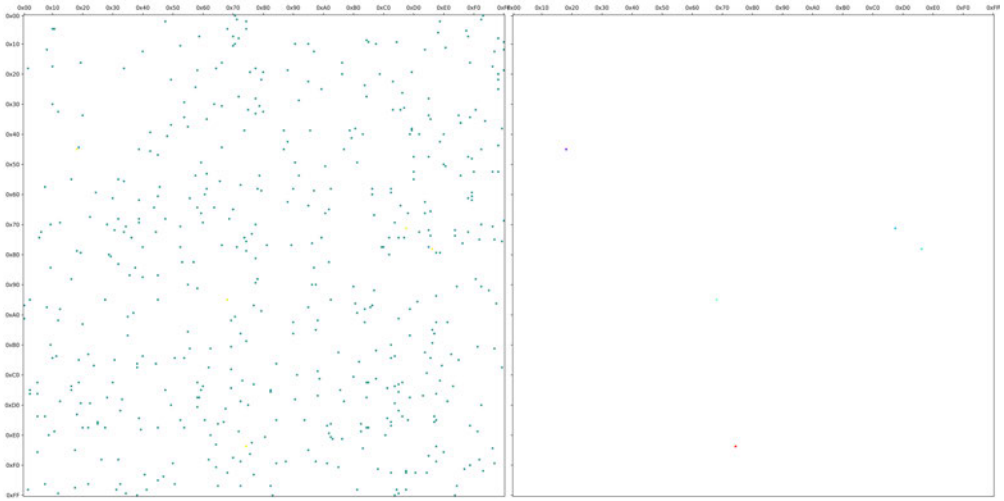
Abbildung B-3: Digramm- und Regionendarstellung der Datei 0003-apk.apk [67]



(d) Sektor 4



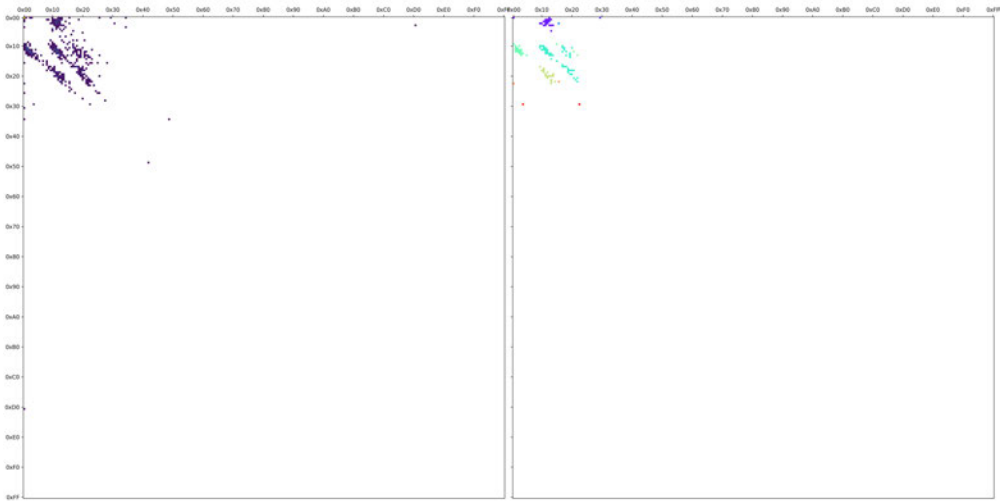
(e) Sektor 5



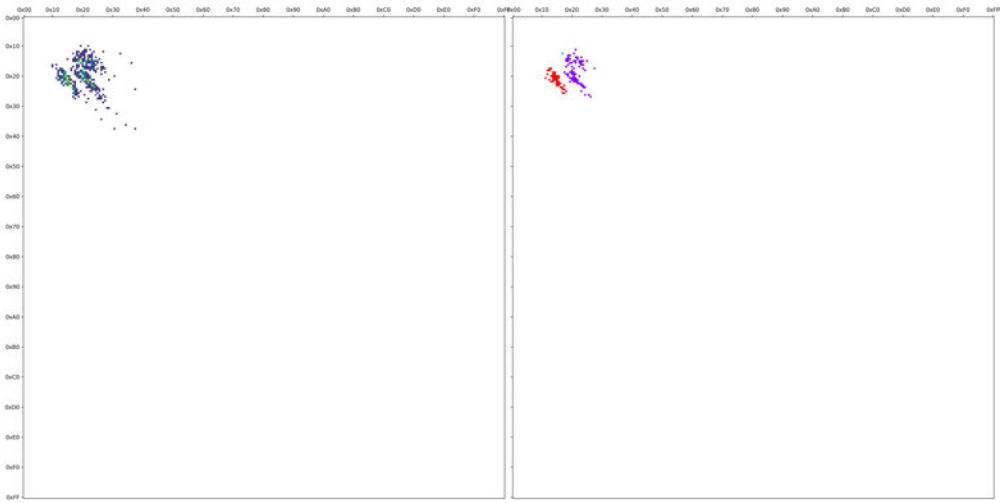
(f) Sektor 6

Abbildung B-3: Digramm- und Regionendarstellung der Datei 0003-apk.apk [67], fortgesetzt

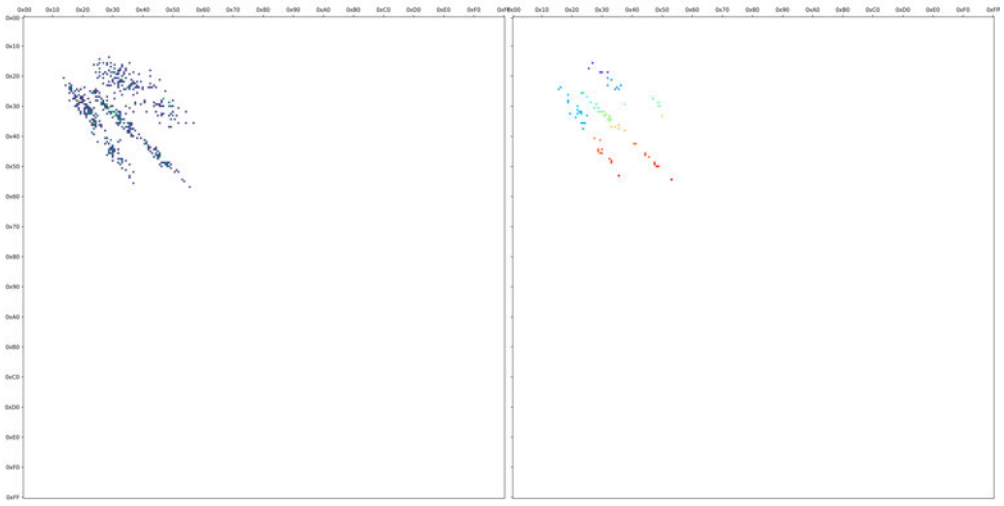
Datei 0003-bmp.bmp



(a) Sektor 1

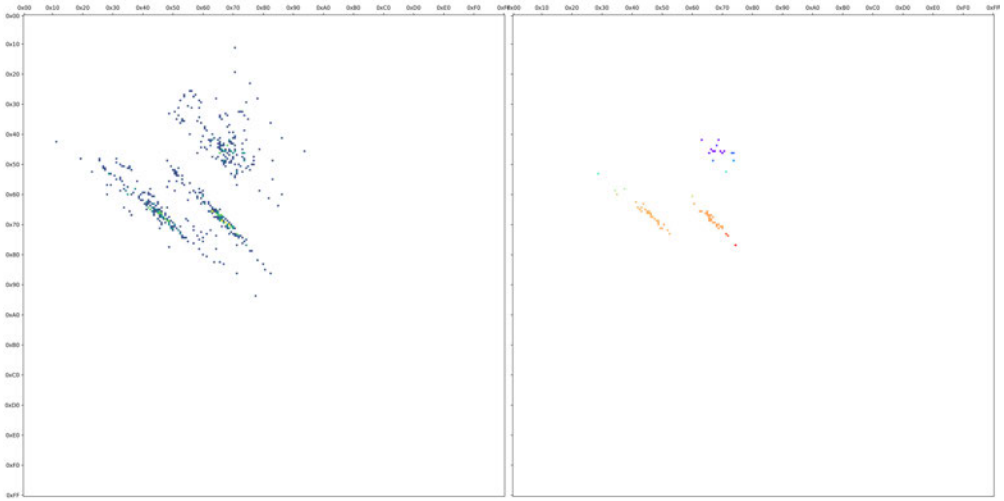


(b) Sektor 2

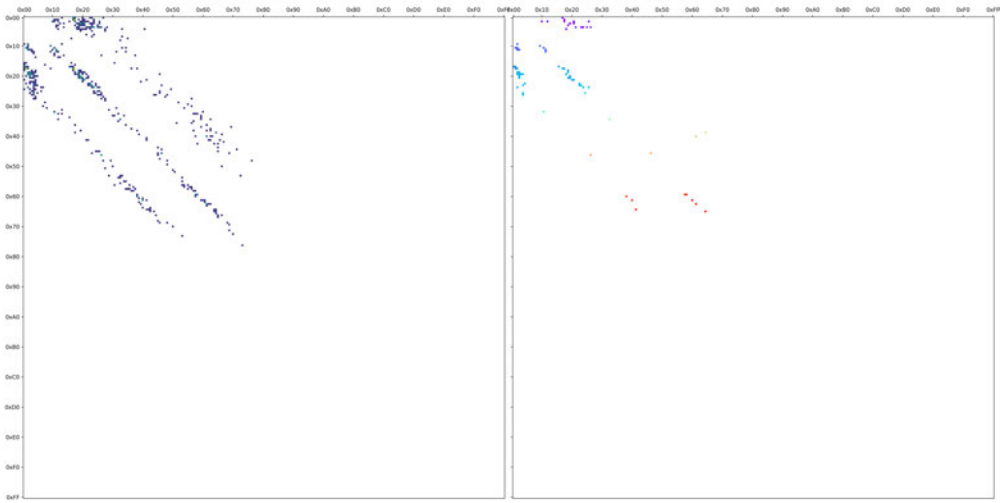


(c) Sektor 3

Abbildung B-4: Digramm- und Regionendarstellung der Datei 0003-bmp.bmp [67]



(d) Sektor 4

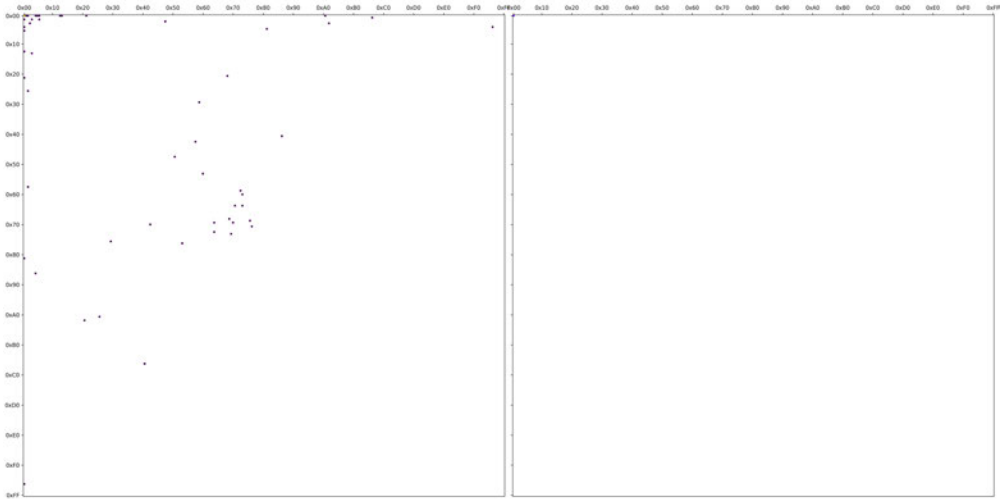


(e) Sektor 5

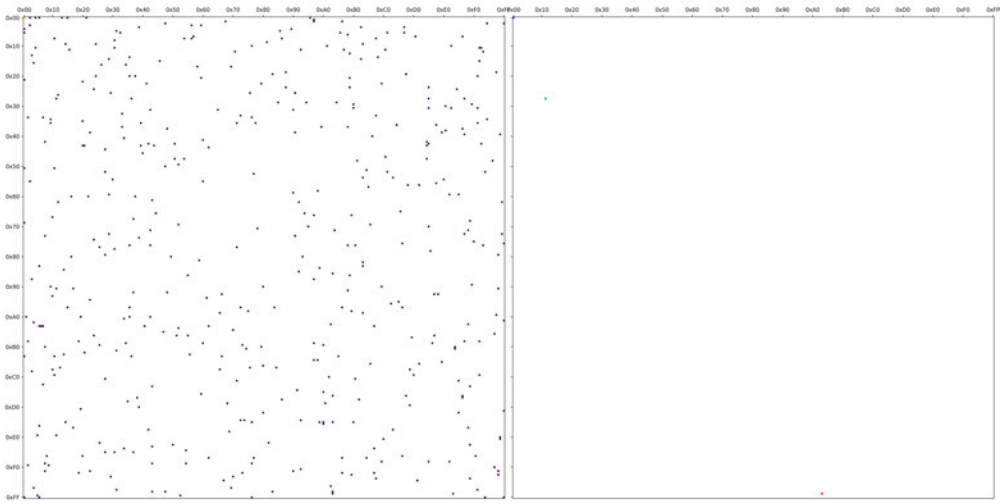
(f) Sektor 6

Abbildung B-4: Digramm- und Regionendarstellung der Datei 0003-bmp.bmp [67], fortgesetzt

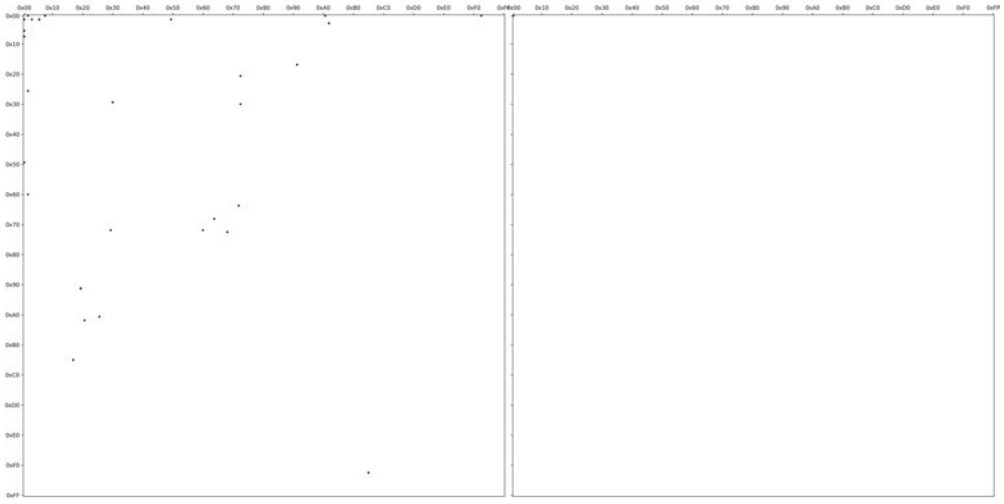
Datei 0003-docx.docx



(a) Sektor 1

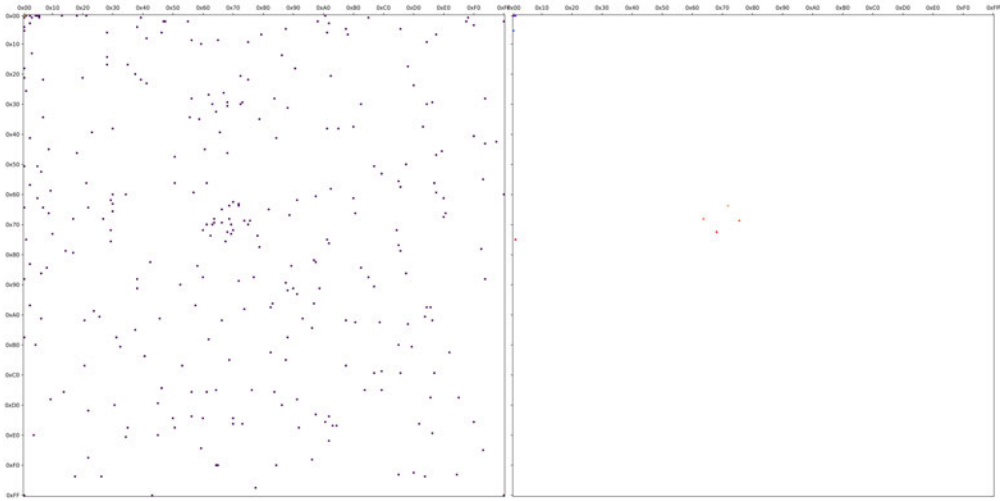


(b) Sektor 2

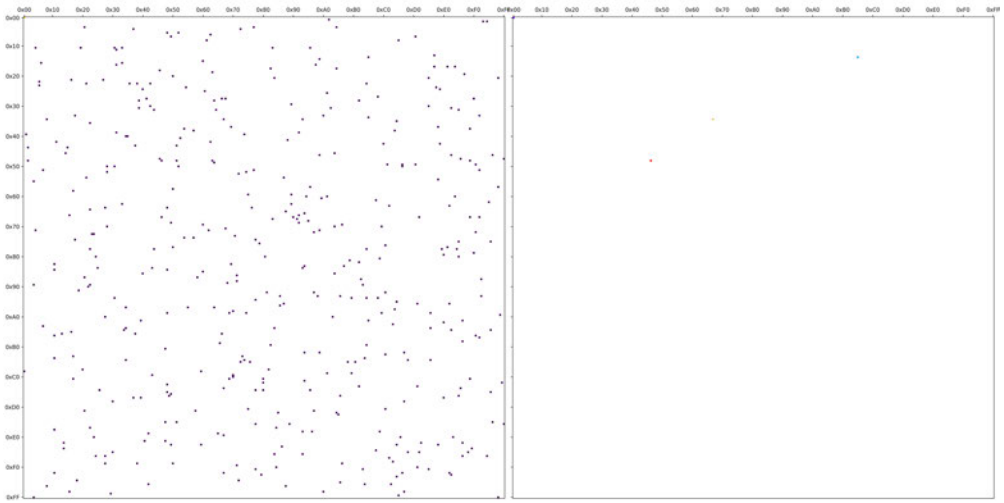


(c) Sektor 3

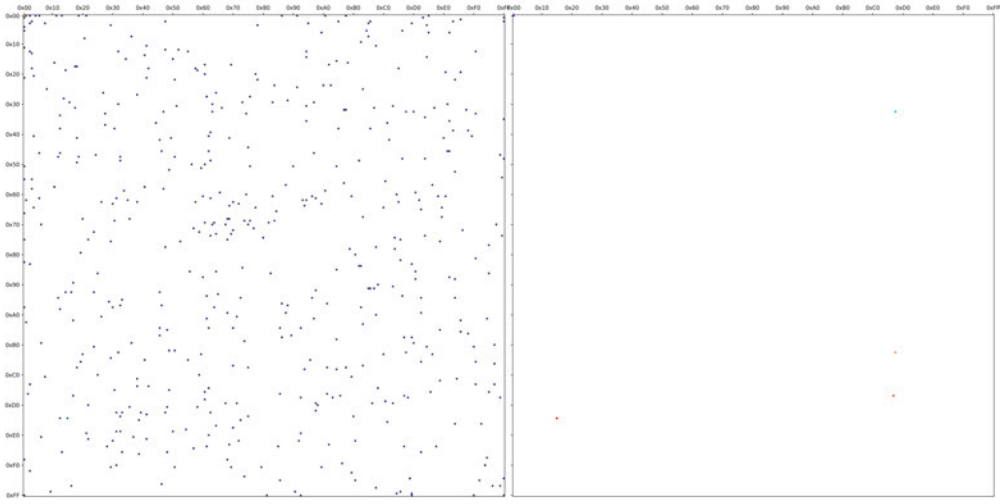
Abbildung B-5: Digramm- und Regionendarstellung der Datei 0003-docx.docx [67]



(d) Sektor 4



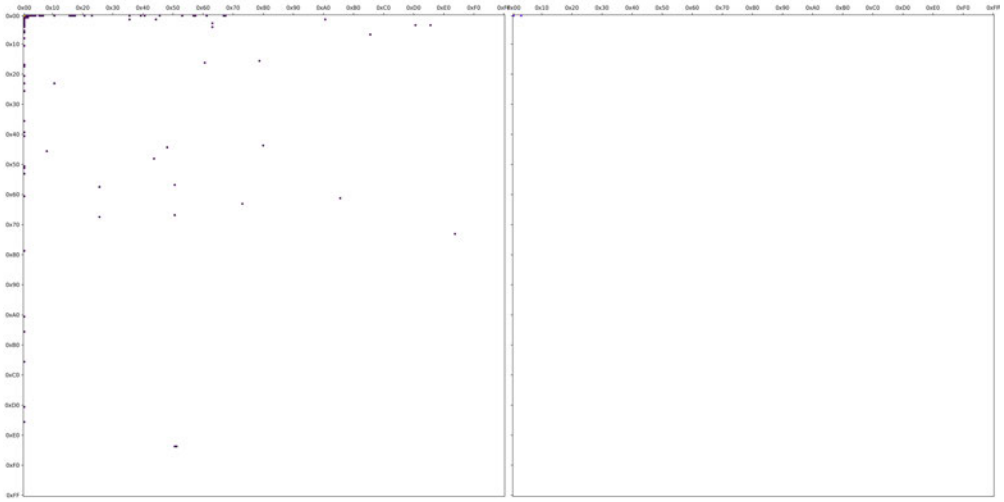
(e) Sektor 5



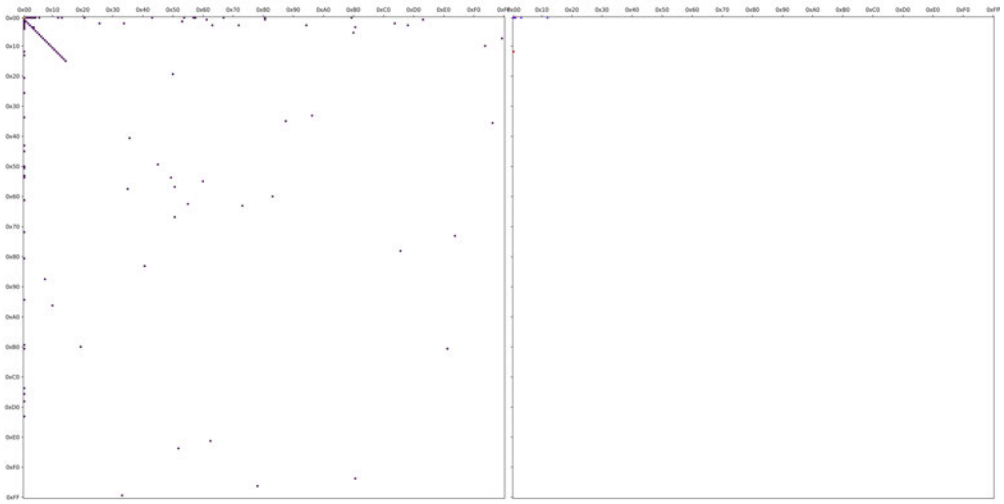
(f) Sektor 6

Abbildung B-5: Digramm- und Regionendarstellung der Datei 0003-docx.docx [67], fortgesetzt

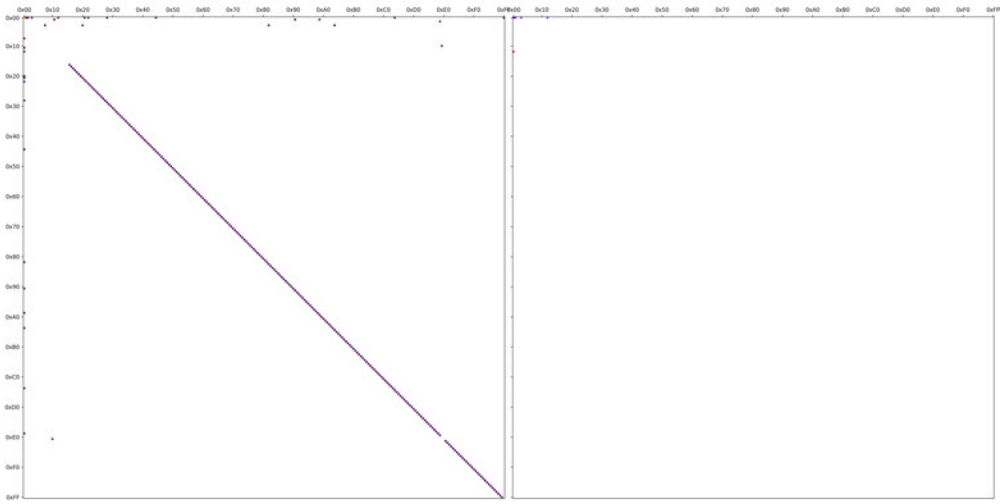
Datei 0003-so.so



(a) Sektor 1

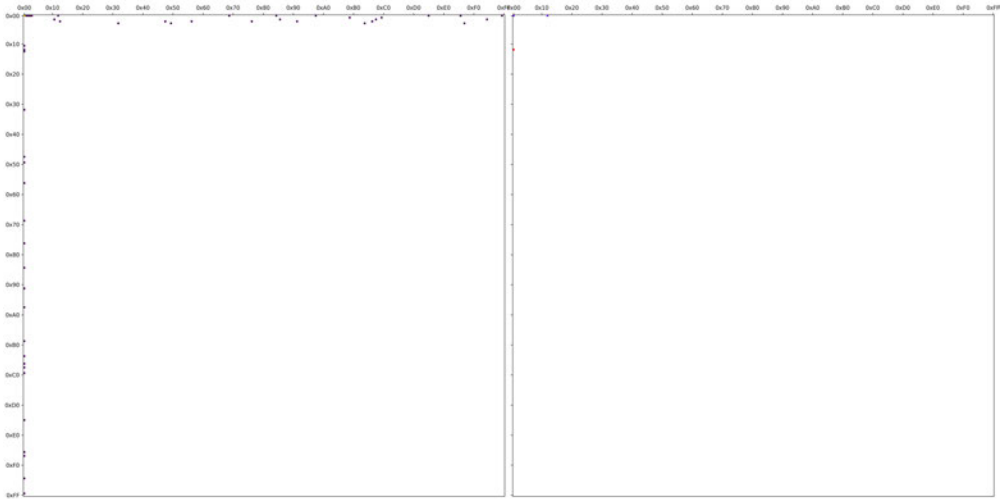


(b) Sektor 2

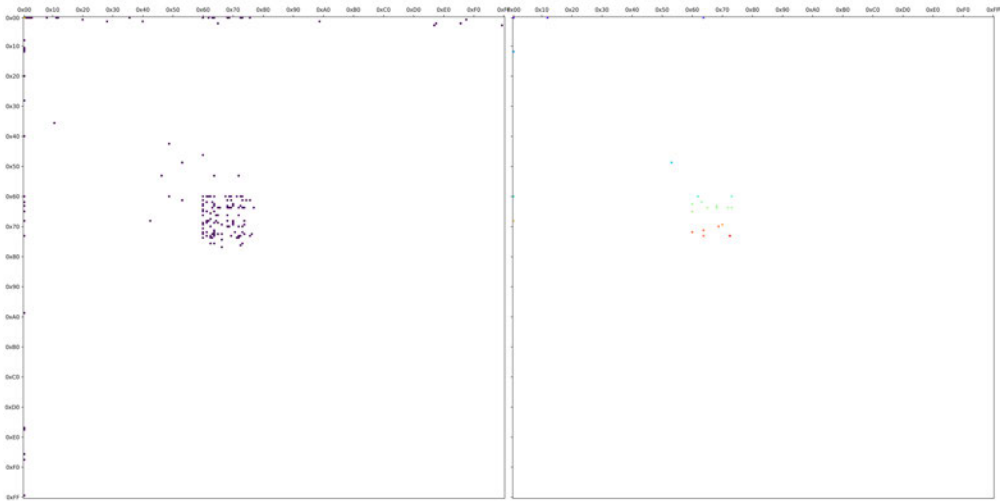


(c) Sektor 3

Abbildung B-6: Digramm- und Regionendarstellung der Datei 0003-so.so [67]



(d) Sektor 4

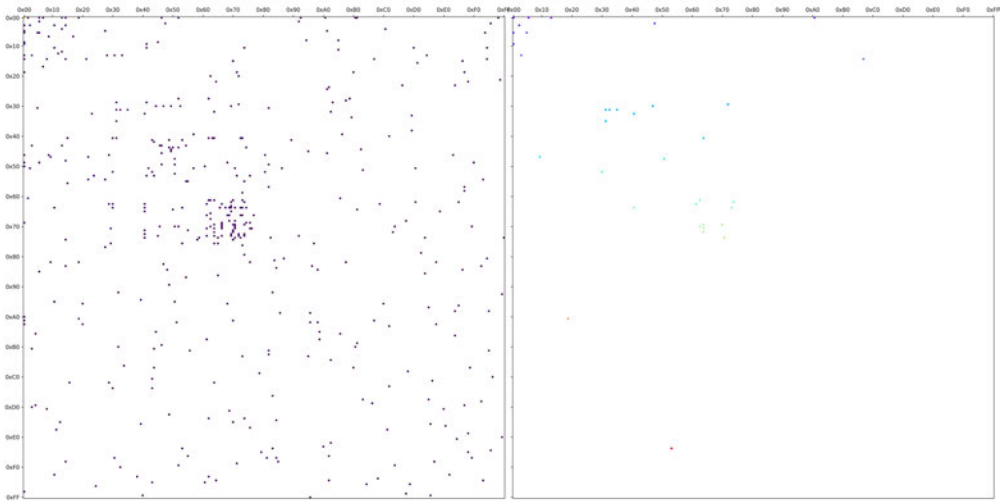


(e) Sektor 5

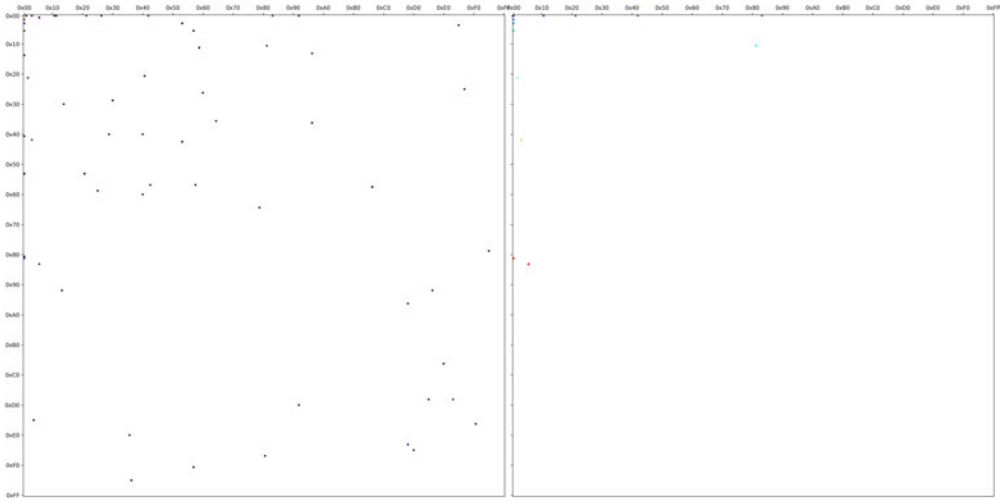
(f) Sektor 6

Abbildung B-6: Digramm- und Regionendarstellung der Datei 0003-so.so [67], fortgesetzt

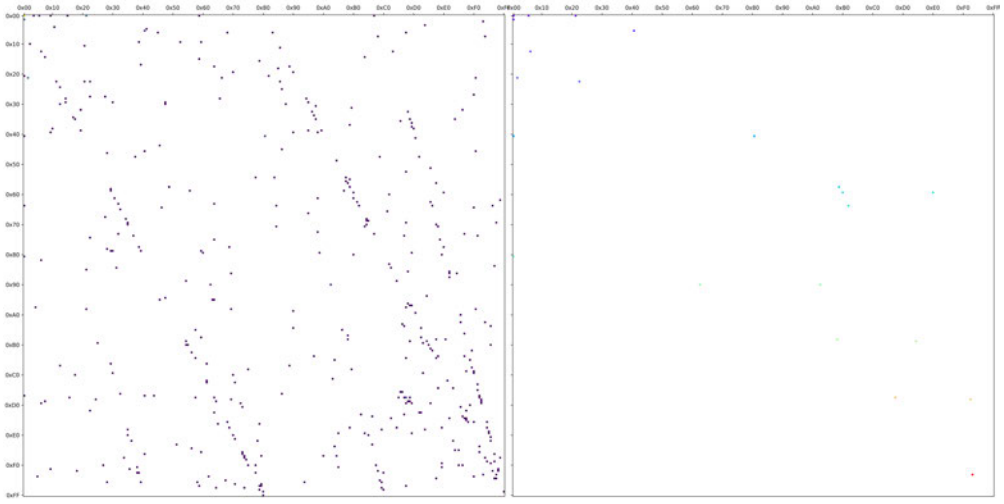
Datei 0003-epub.epub



(a) Sektor 1

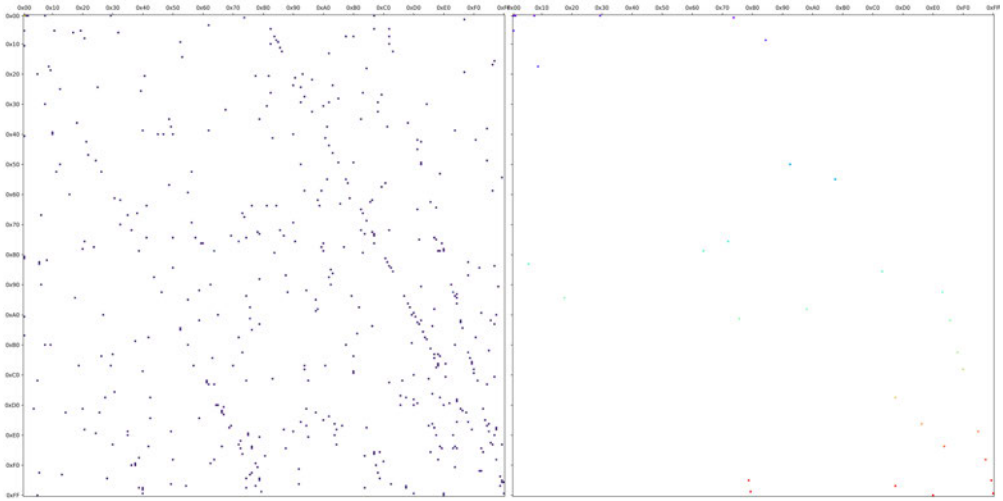


(b) Sektor 2

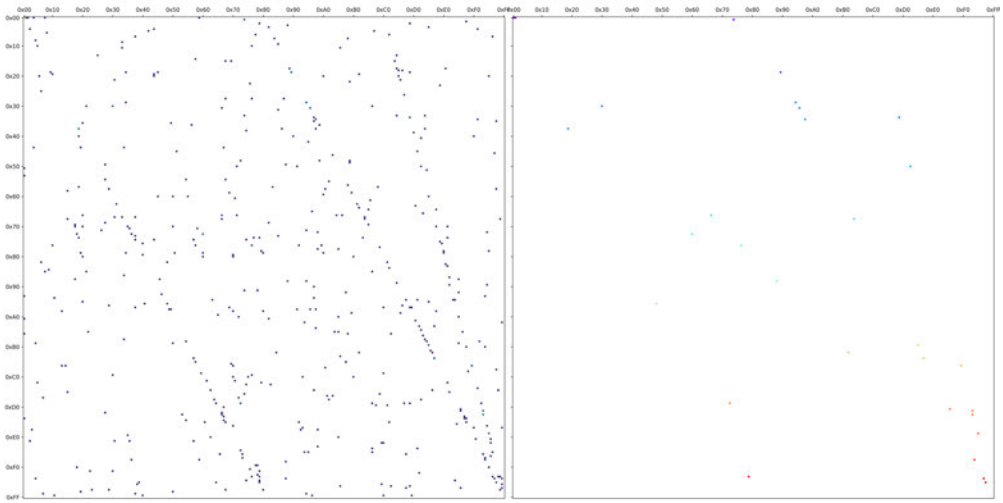


(c) Sektor 3

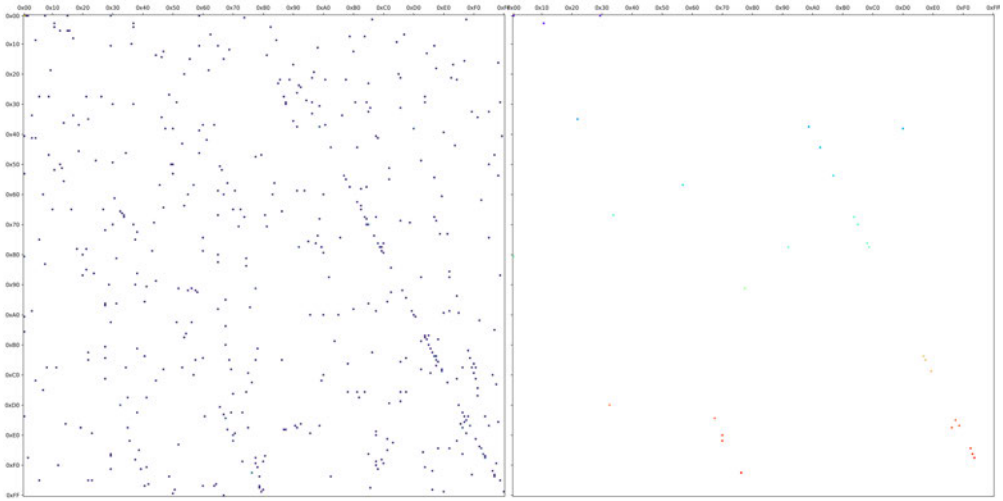
Abbildung B-7: Digramm- und Regionendarstellung der Datei 0003-epub.epub [67]



(d) Sektor 4



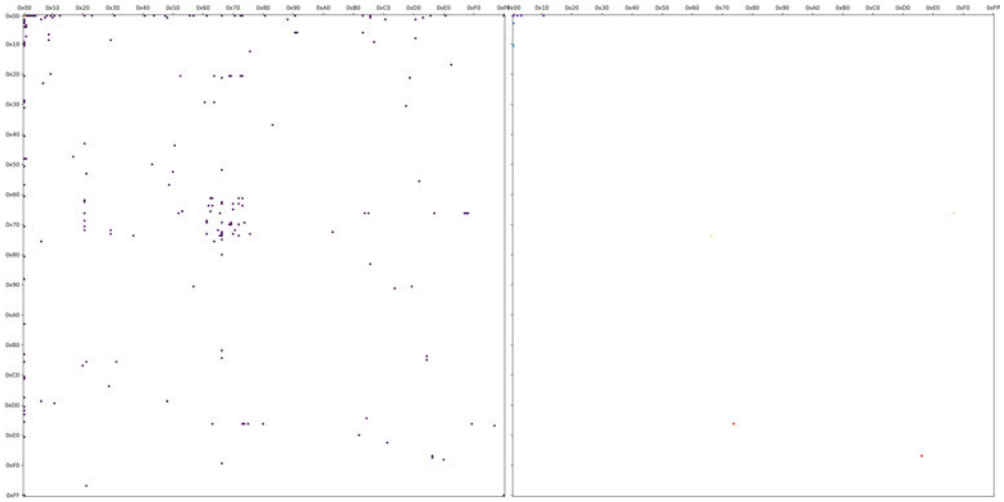
(e) Sektor 5



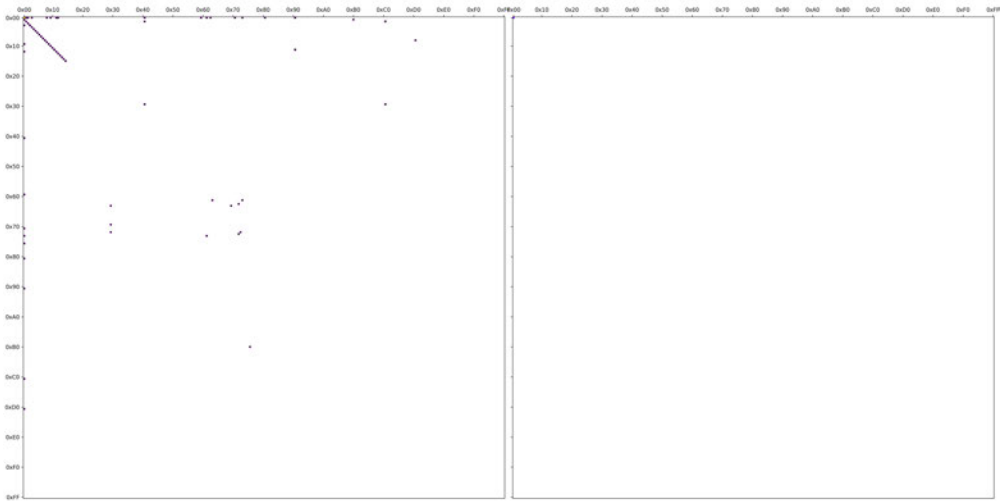
(f) Sektor 6

Abbildung B-7: Digramm- und Regionendarstellung der Datei 0003-epub.epub [67], fortgesetzt

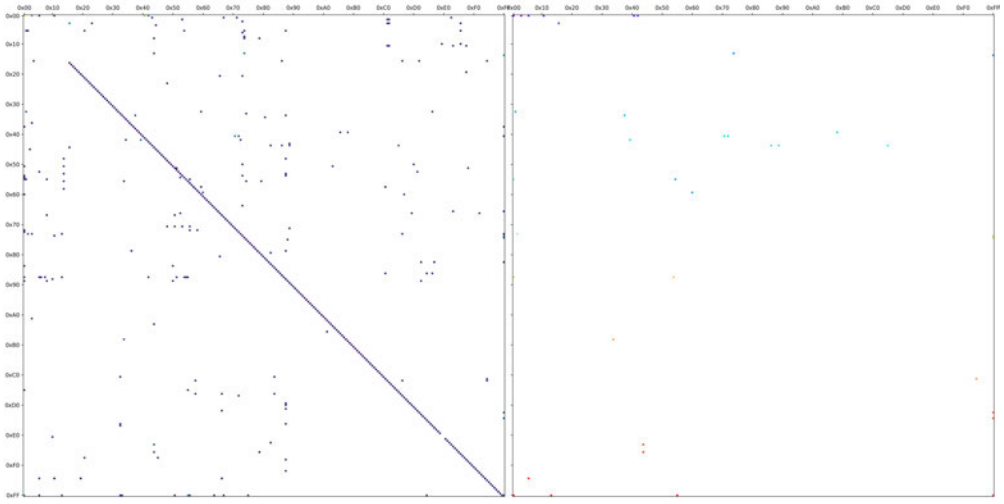
Datei 0003-exe.exe



(a) Sektor 1

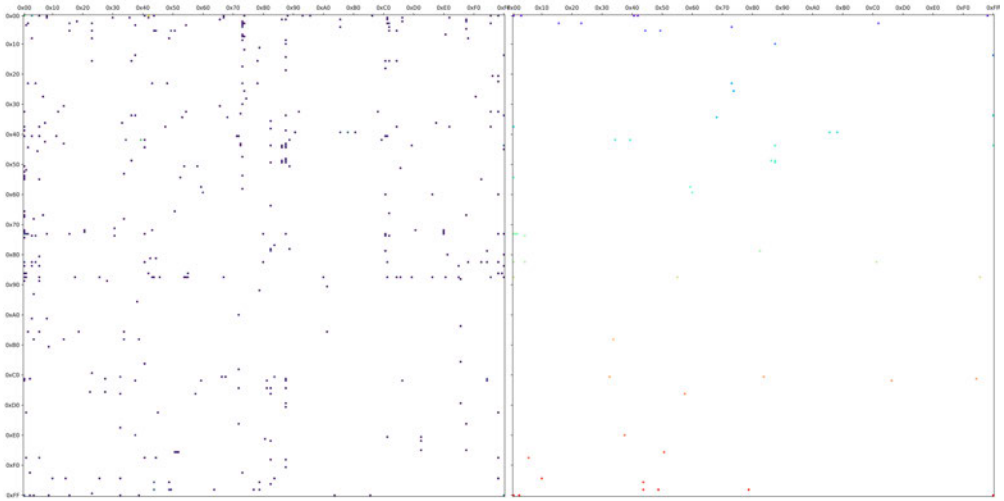


(b) Sektor 2

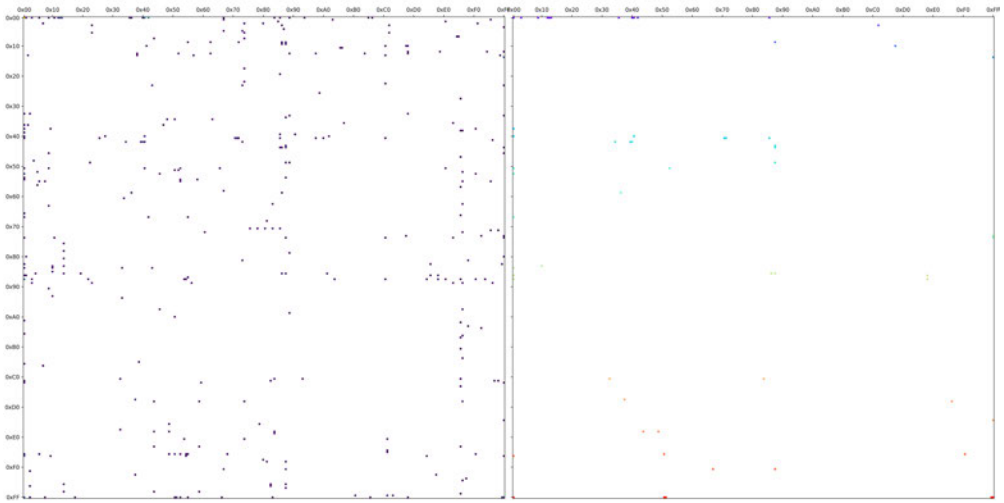


(c) Sektor 3

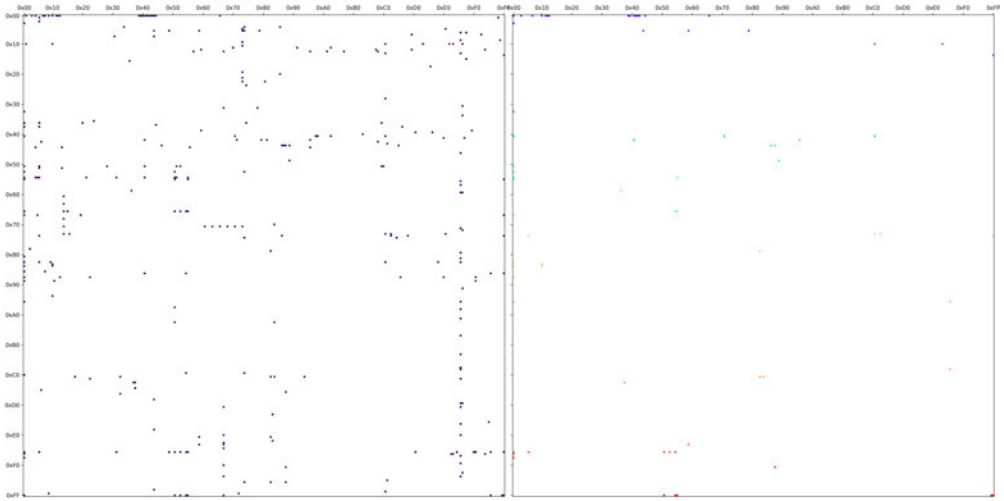
Abbildung B-8: Digramm- und Regionendarstellung der Datei 0003-exe.exe [67]



(d) Sektor 4



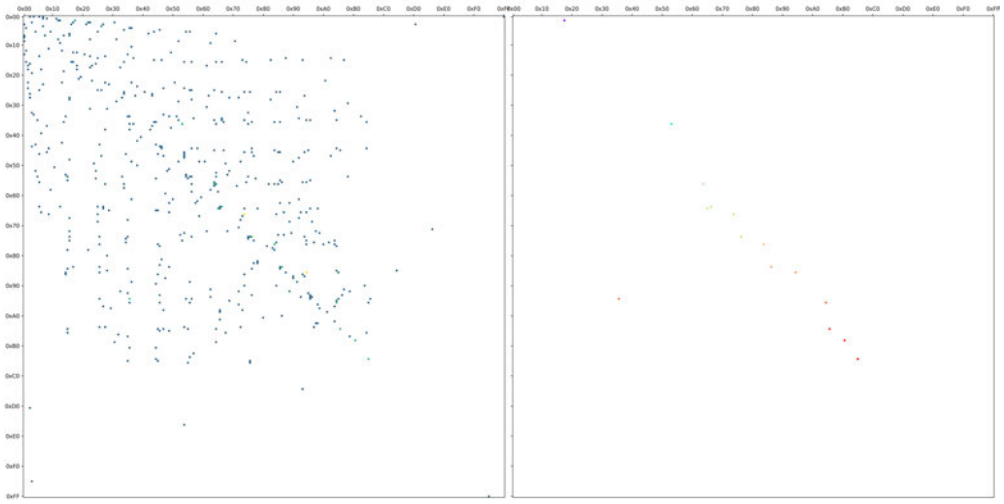
(e) Sektor 5



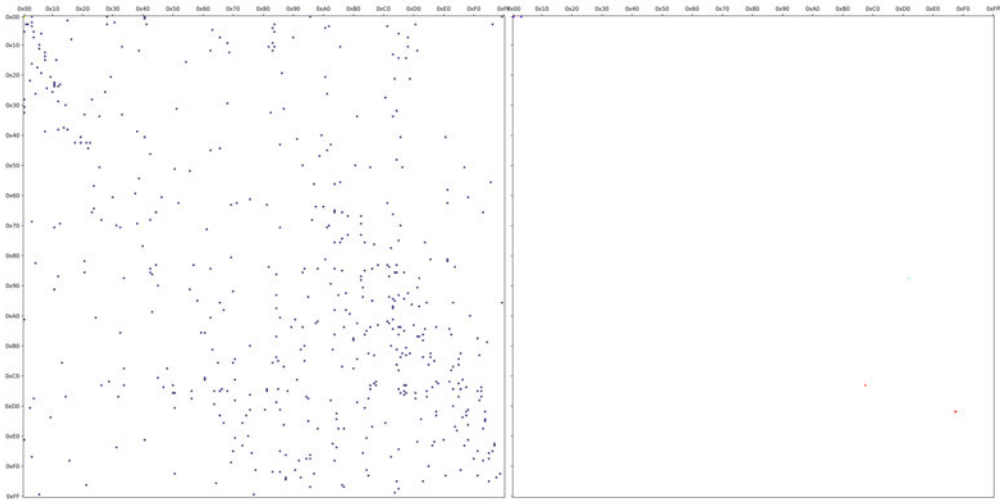
(f) Sektor 6

Abbildung B-8: Digramm- und Regionendarstellung der Datei 0003-exe.exe [67], fortgesetzt

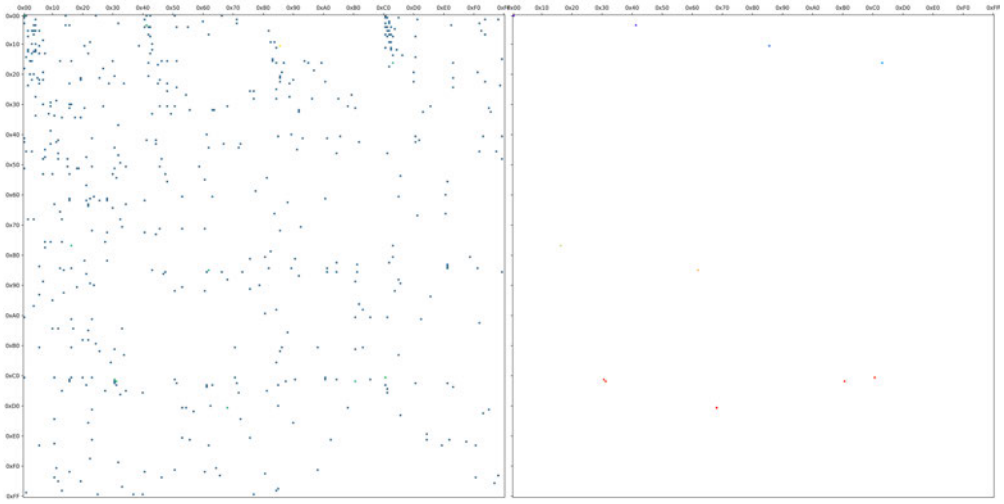
Datei 0003-gif.gif



(a) Sektor 1

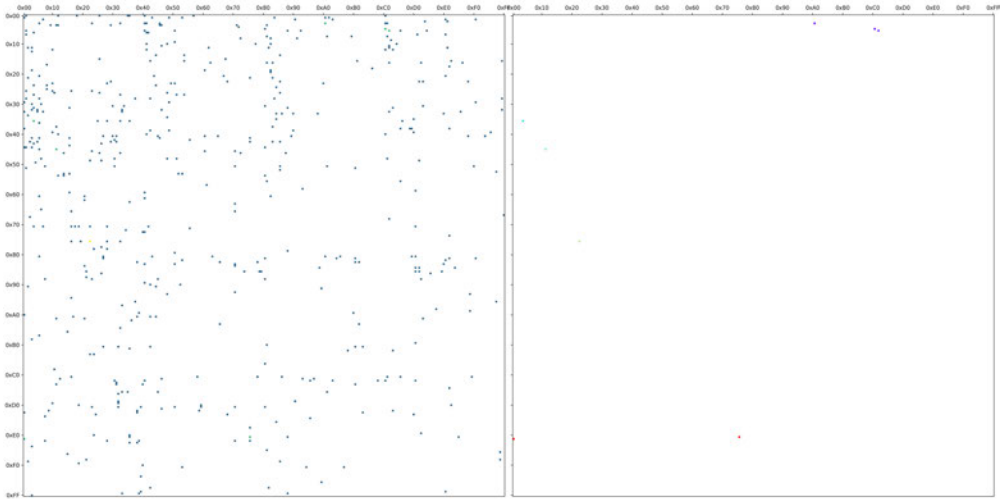


(b) Sektor 2

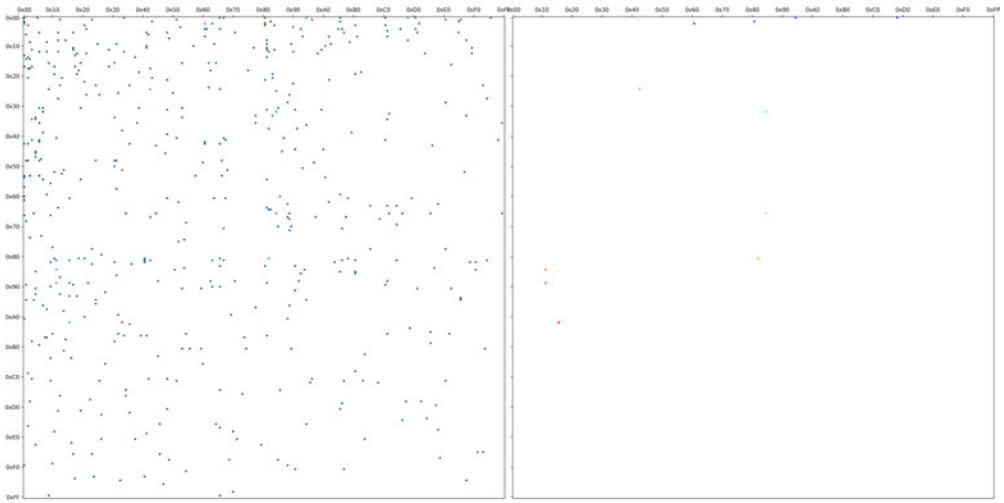


(c) Sektor 3

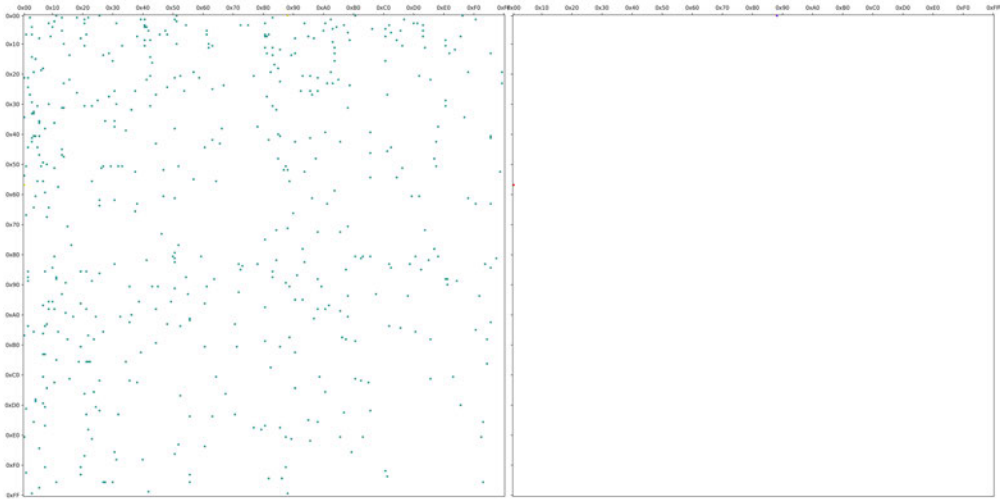
Abbildung B-9: Digramm- und Regionendarstellung der Datei 0003-gif.gif [67]



(d) Sektor 4



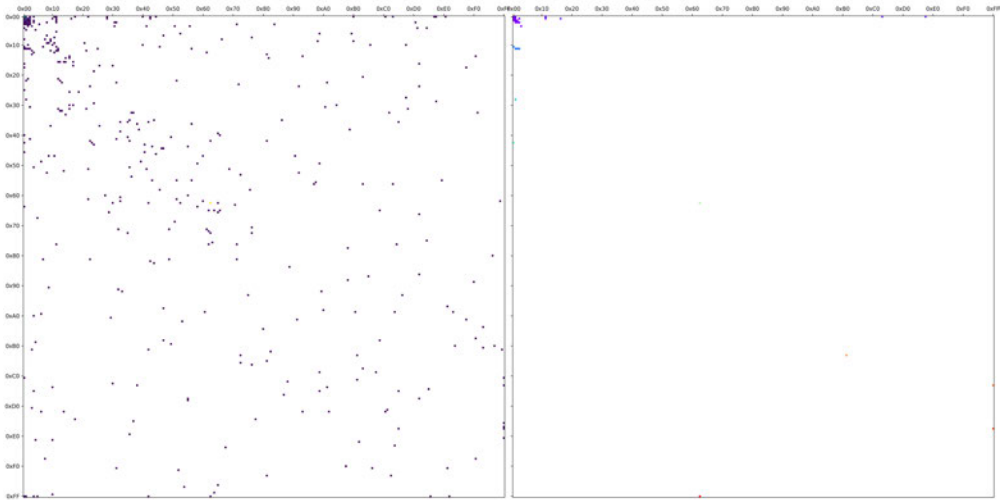
(e) Sektor 5



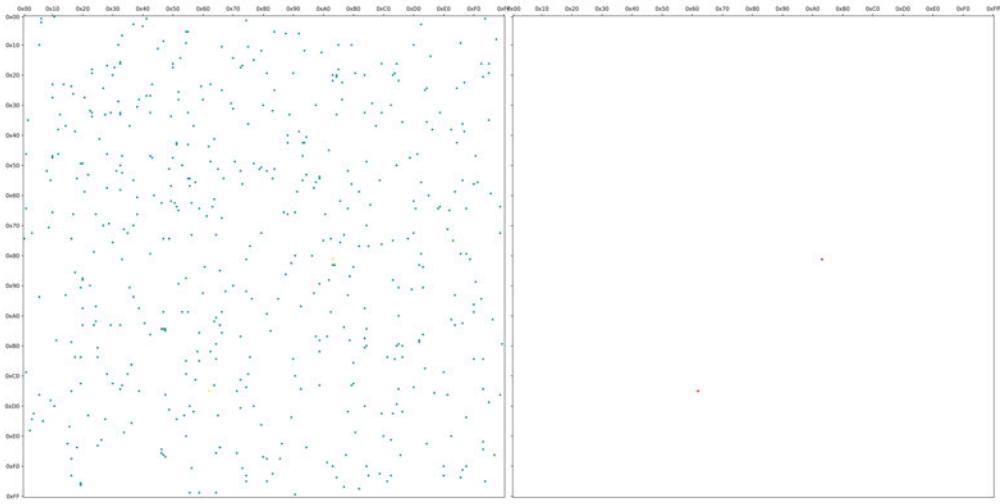
(f) Sektor 6

Abbildung B-9: Digramm- und Regionendarstellung der Datei 0003-gif.gif [67], fortgesetzt

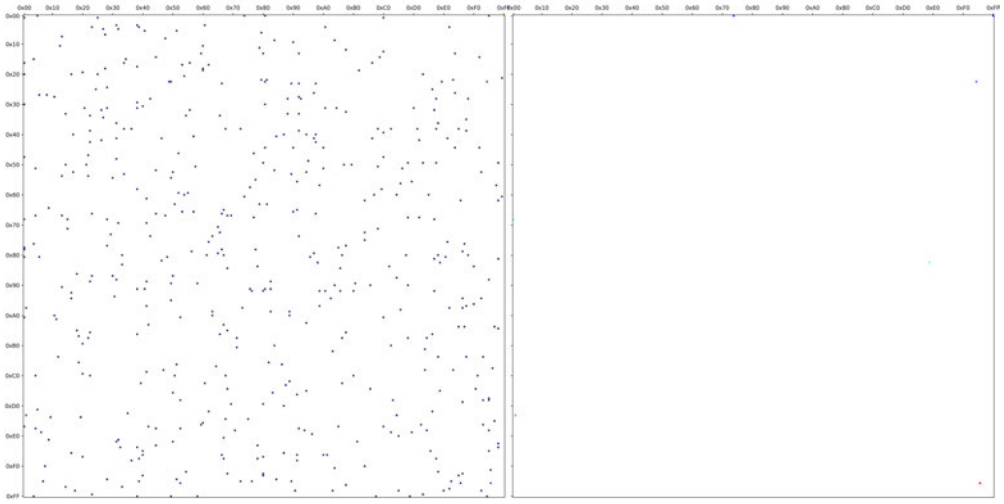
Datei 0003-jpg-q50.jpg



(a) Sektor 1

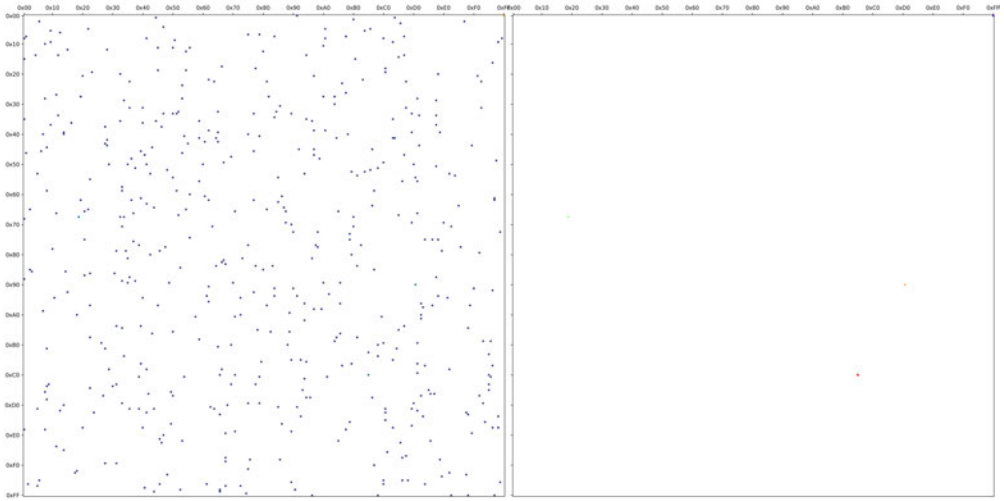


(b) Sektor 2

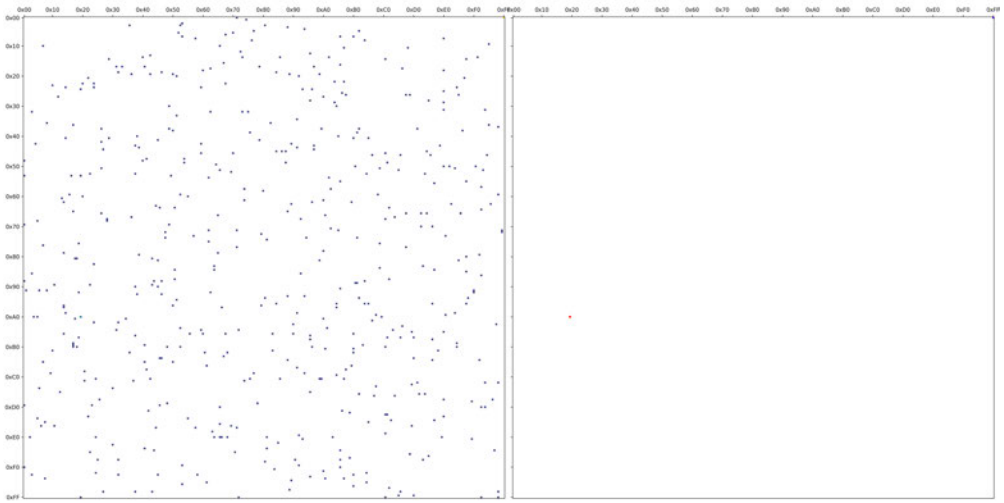


(c) Sektor 3

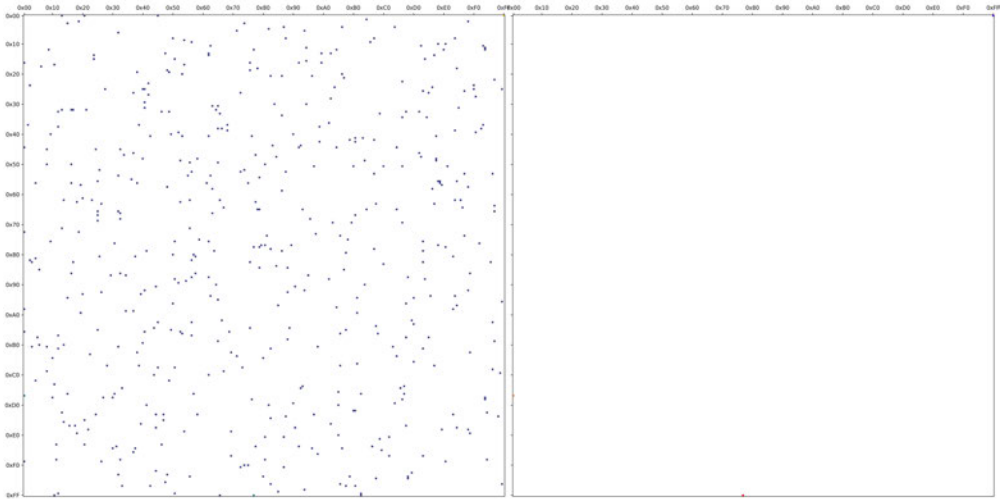
Abbildung B-10: Digramm- und Regionendarstellung der Datei 0003-jpg-q50.jpg [67]



(d) Sektor 4



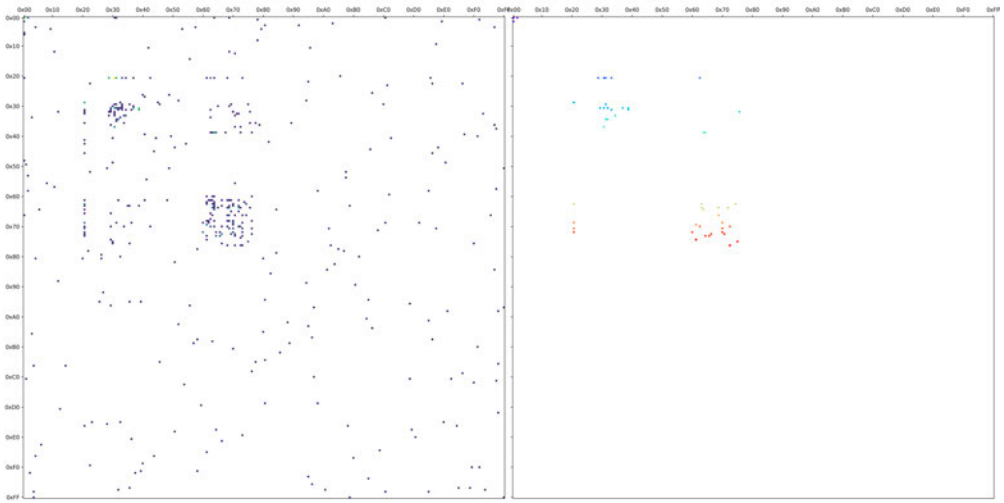
(e) Sektor 5



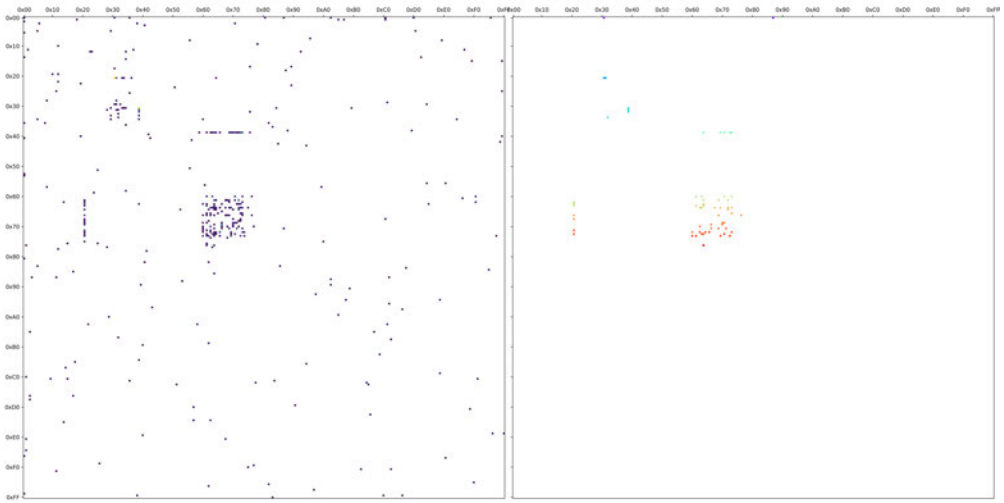
(f) Sektor 6

Abbildung B-10: Digramm- und Regionendarstellung der Datei 0003-jpg-q50.jpg [67], fortgesetzt

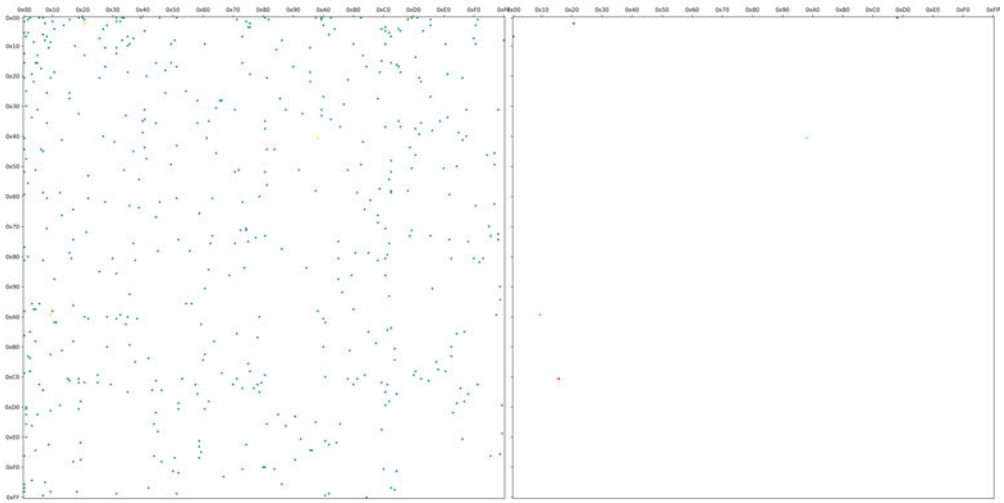
Datei 0003-mp4.mp4



(a) Sektor 1

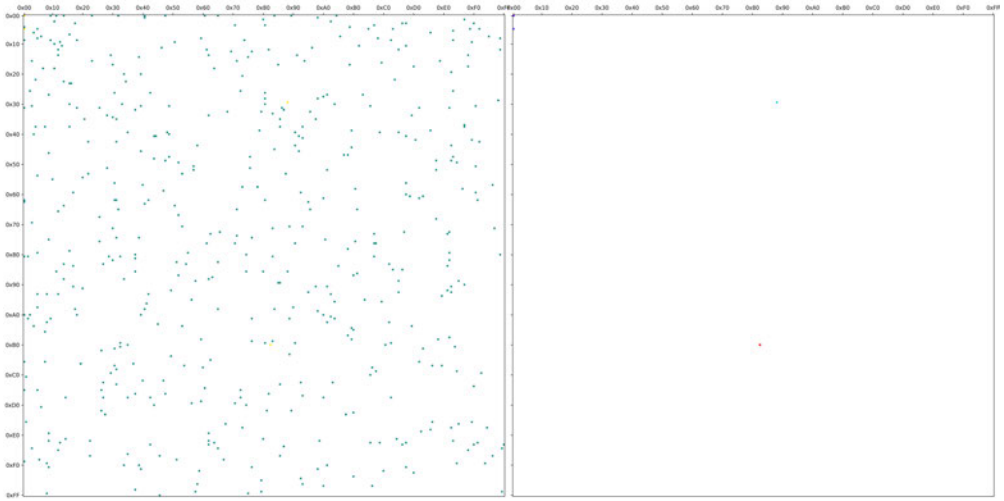


(b) Sektor 2

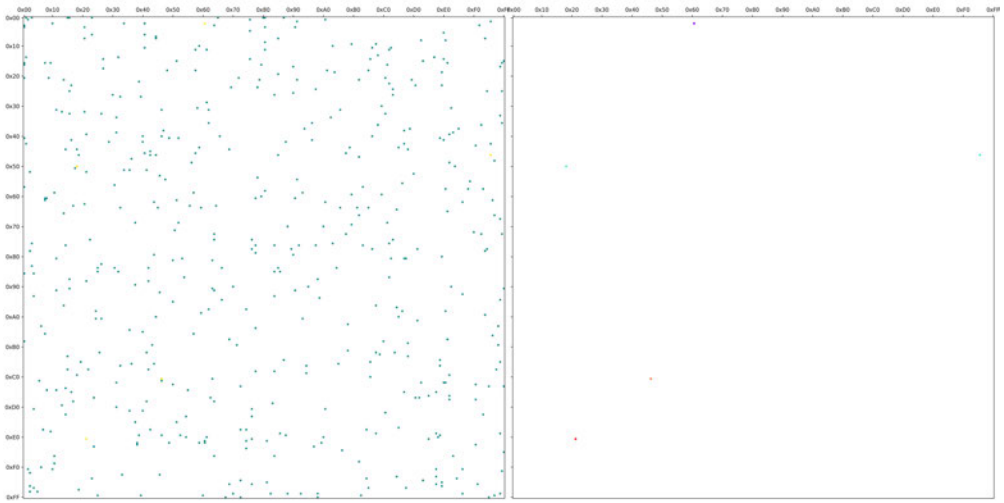


(c) Sektor 3

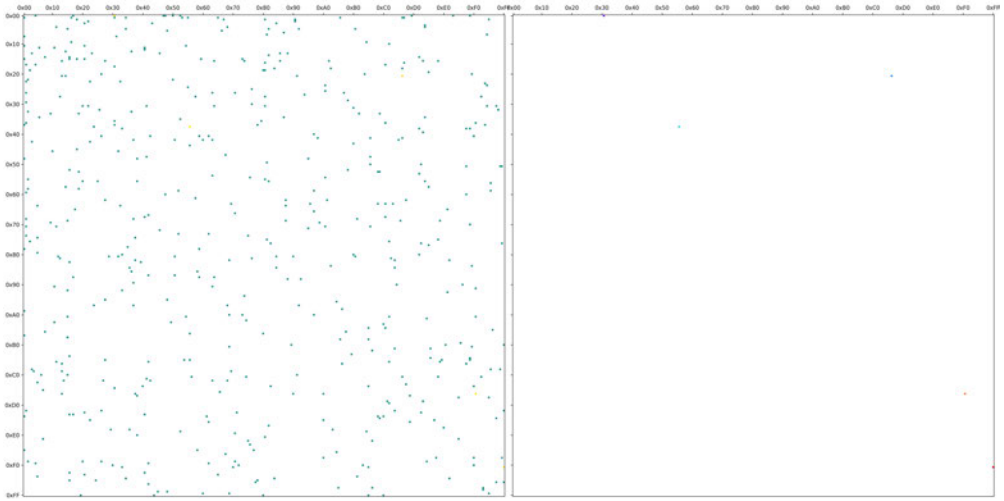
Abbildung B-11: Digramm- und Regionendarstellung der Datei 0003-mp4.mp4 [67]



(d) Sektor 4



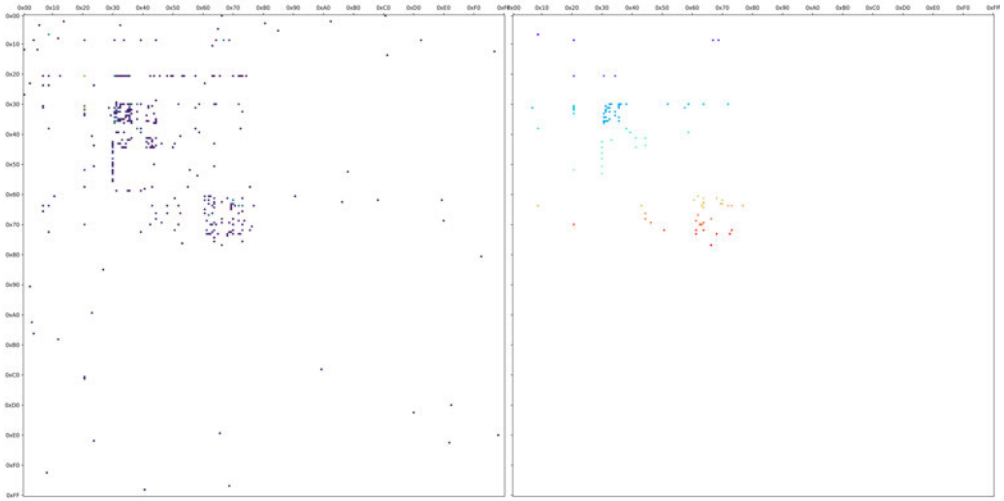
(e) Sektor 5



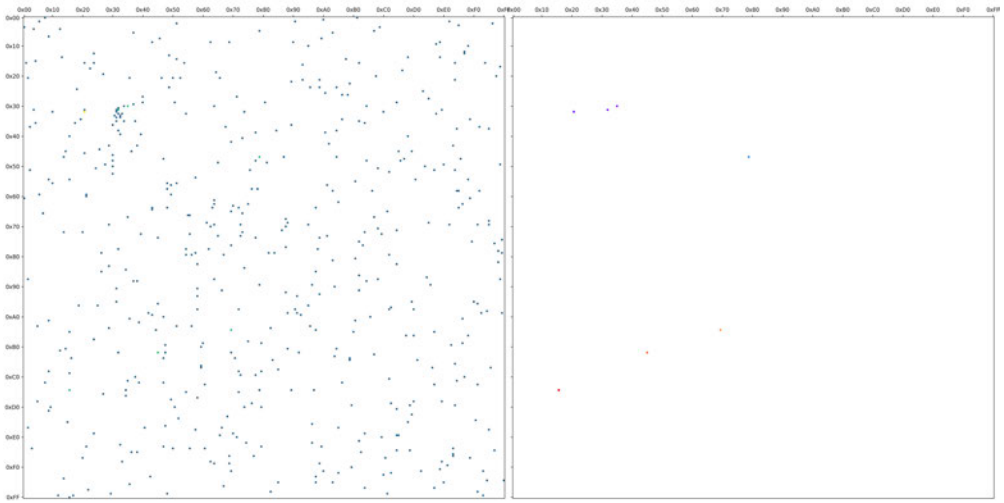
(f) Sektor 6

Abbildung B-11: Digramm- und Regionendarstellung der Datei 0003-mp4.mp4 [67], fortgesetzt

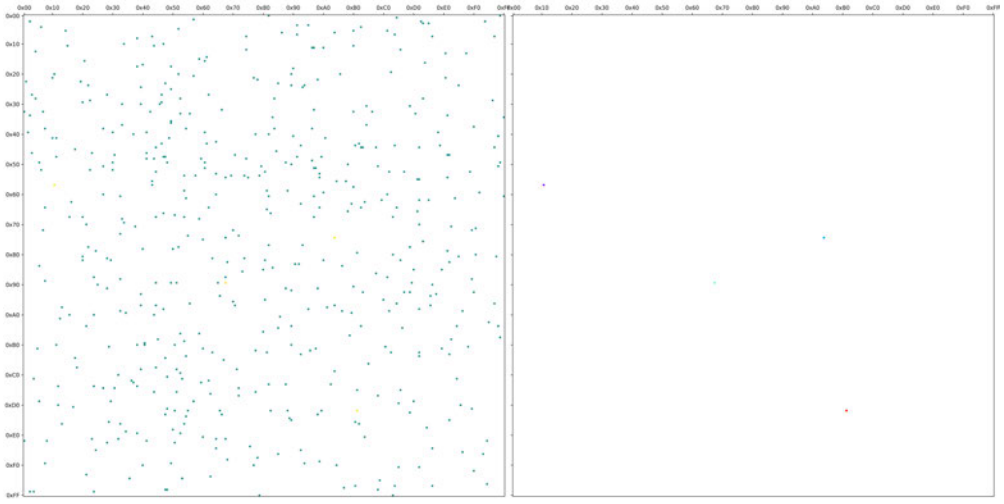
Datei 0003-pdf.pdf



(a) Sektor 1

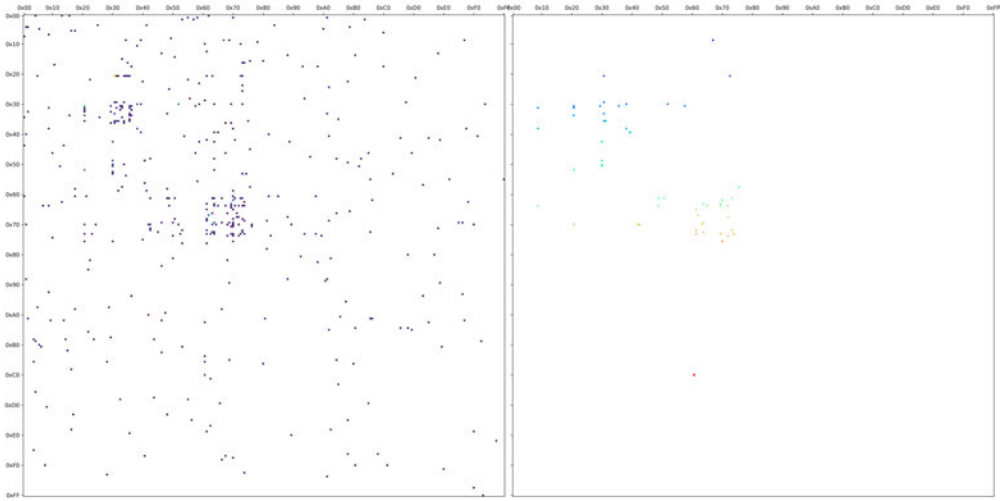


(b) Sektor 2

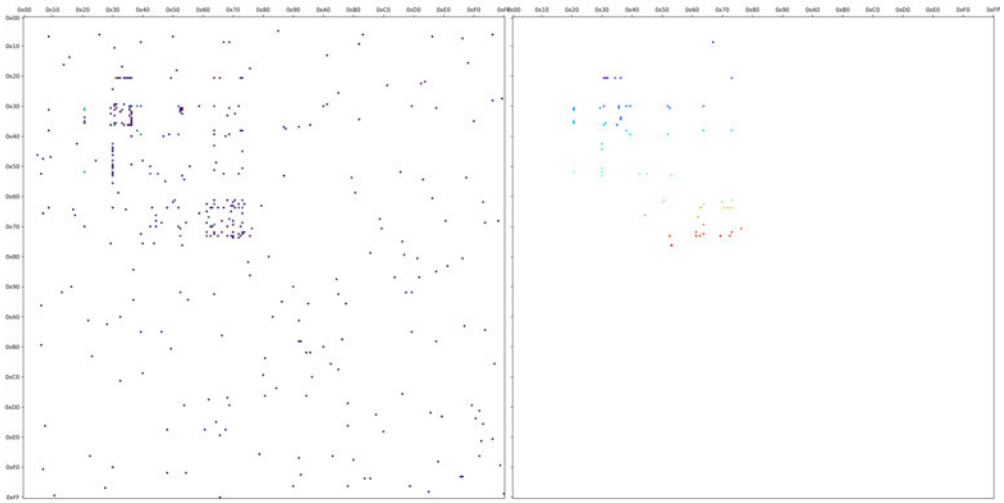


(c) Sektor 3

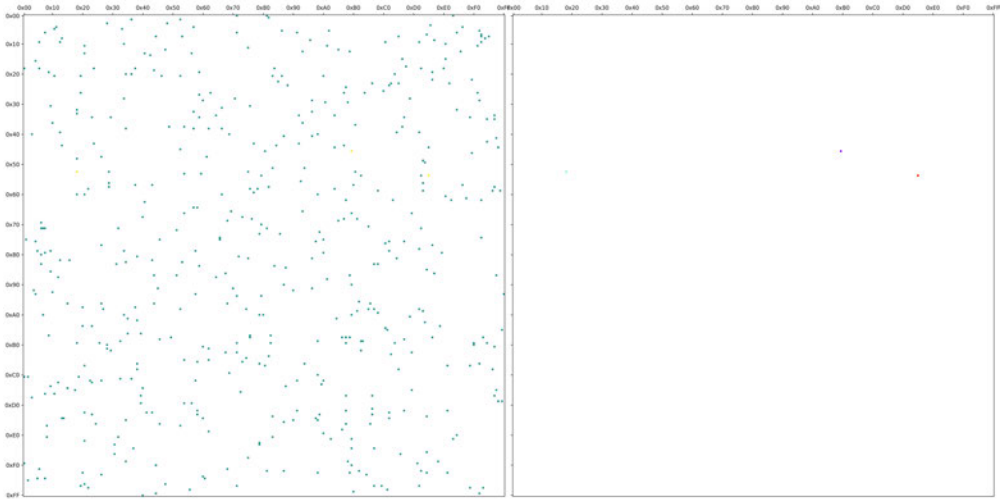
Abbildung B-12: Digramm- und Regionendarstellung der Datei 0003-pdf.pdf [67]



(d) Sektor 4



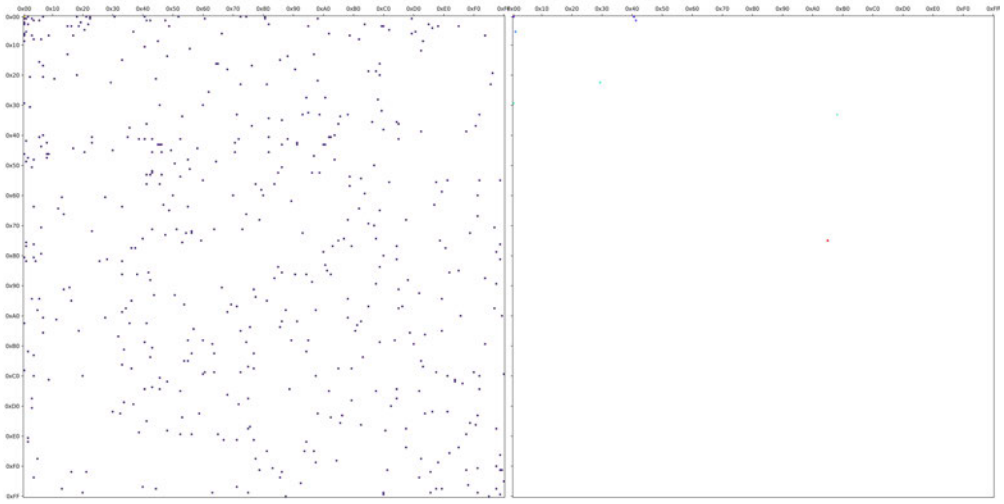
(e) Sektor 5



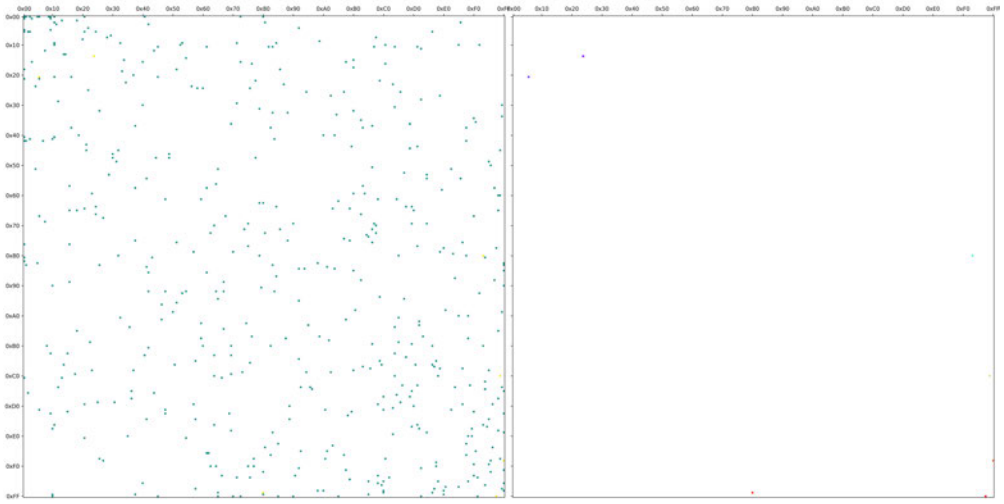
(f) Sektor 6

Abbildung B-12: Digramm- und Regionendarstellung der Datei 0003-pdf.pdf [67], fortgesetzt

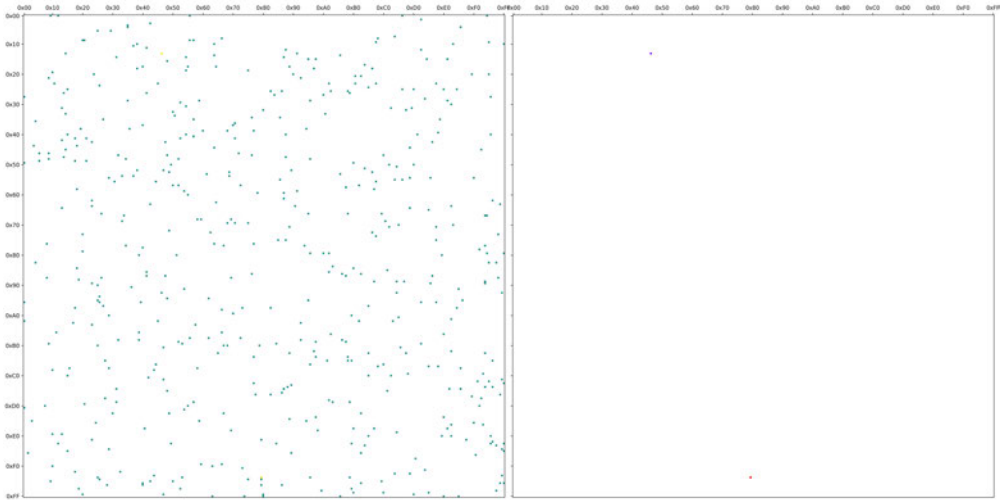
Datei 0003-png-c9.png



(a) Sektor 1

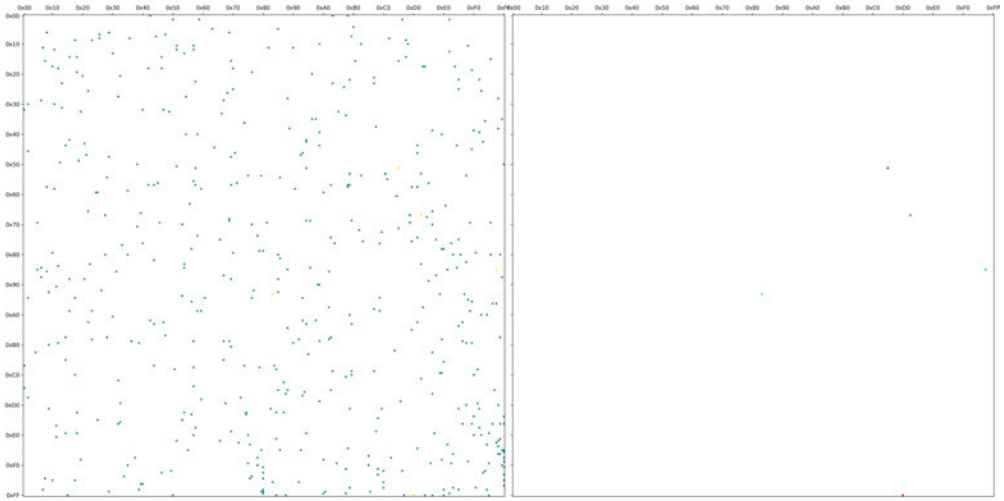


(b) Sektor 2

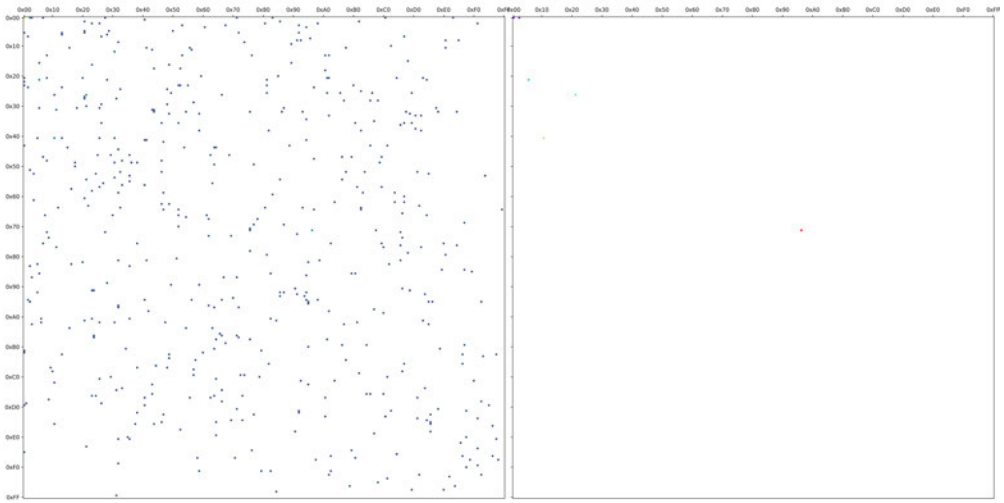


(c) Sektor 3

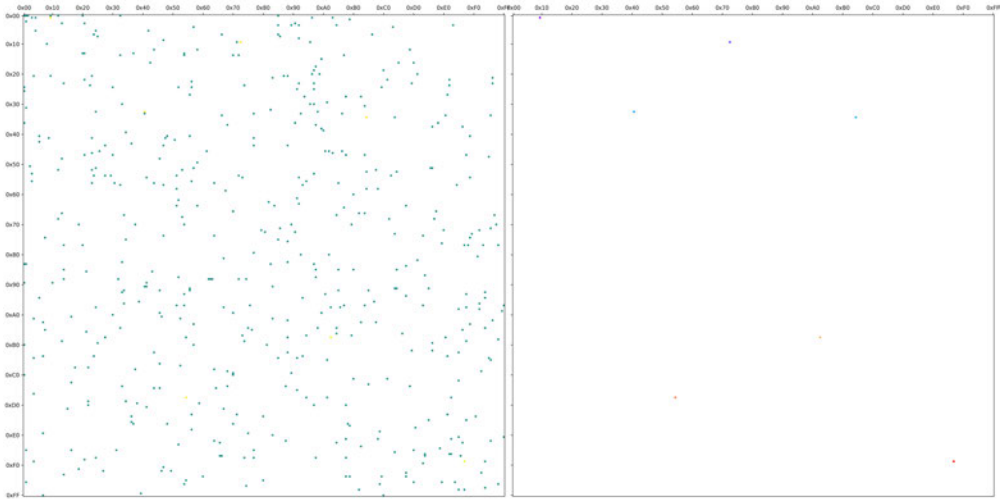
Abbildung B-13: Digramm- und Regionendarstellung der Datei 0003-png-from-web.png [67]



(d) Sektor 4



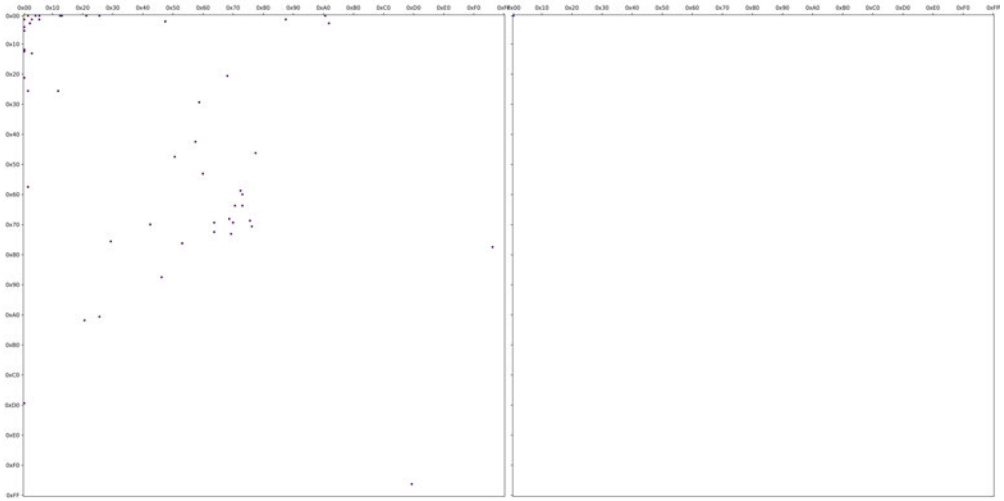
(e) Sektor 5



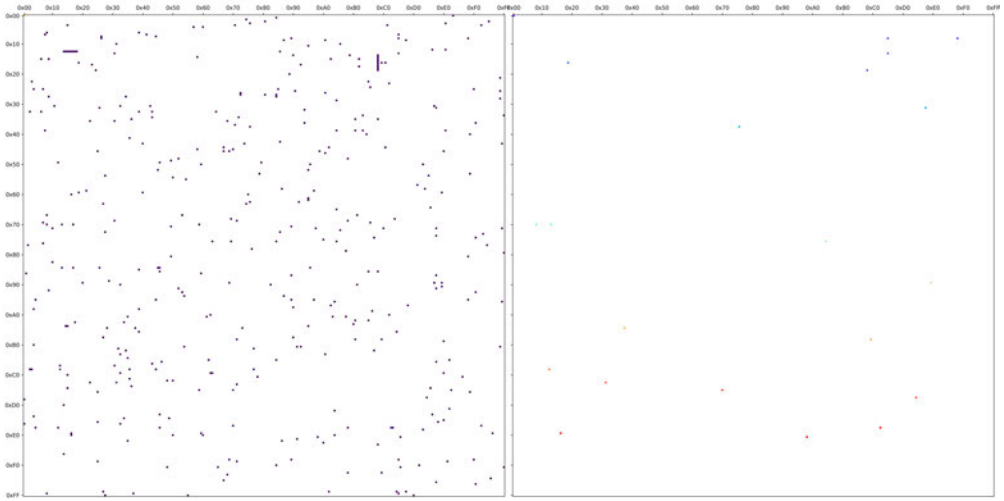
(f) Sektor 6

Abbildung B-13: Digramm- und Regionendarstellung der Datei 0003-png-from-web.png [67], fortgesetzt

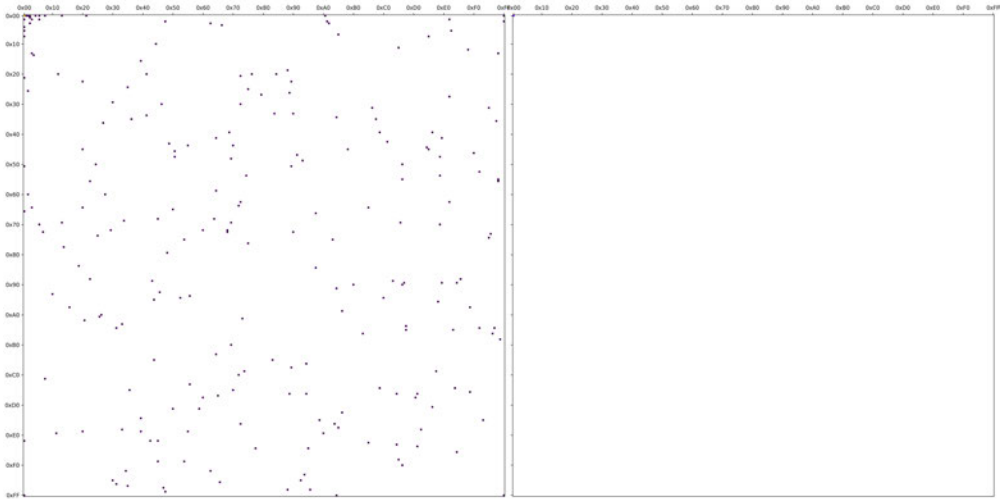
Datei 0003-pptx.pptx



(a) Sektor 1

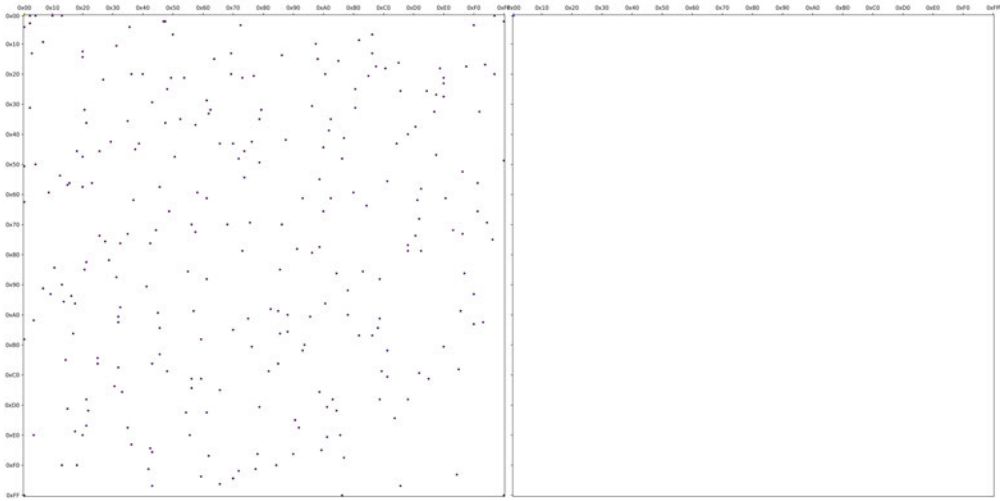


(b) Sektor 2

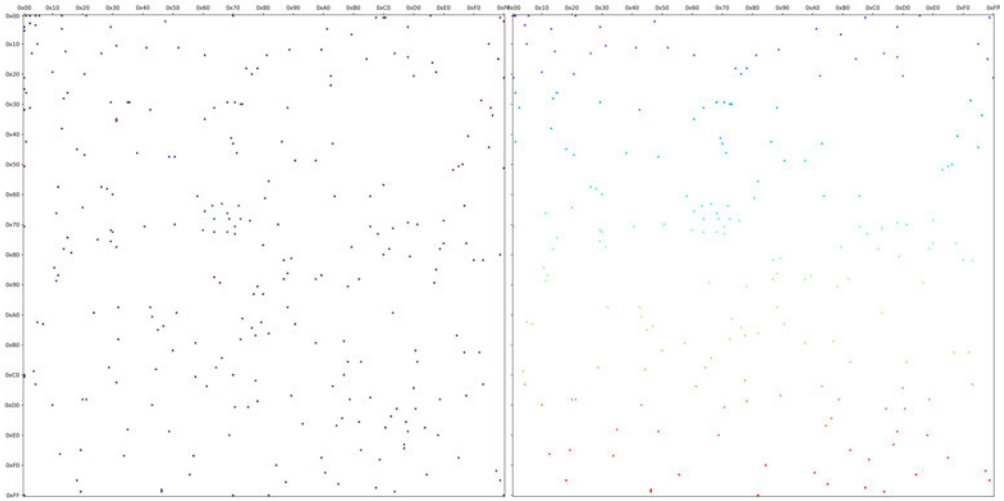


(c) Sektor 3

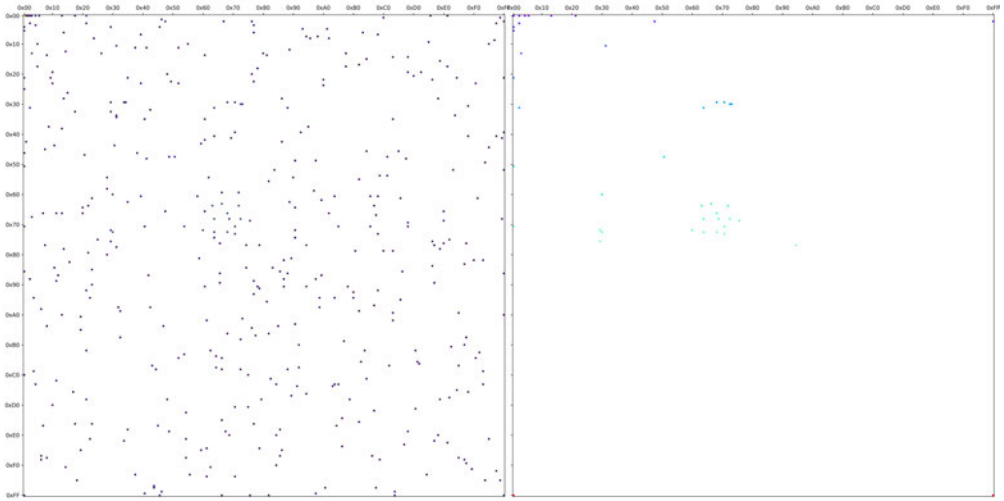
Abbildung B-14: Digramm- und Regionendarstellung der Datei 0003-pptx.pptx [67]



(d) Sektor 4



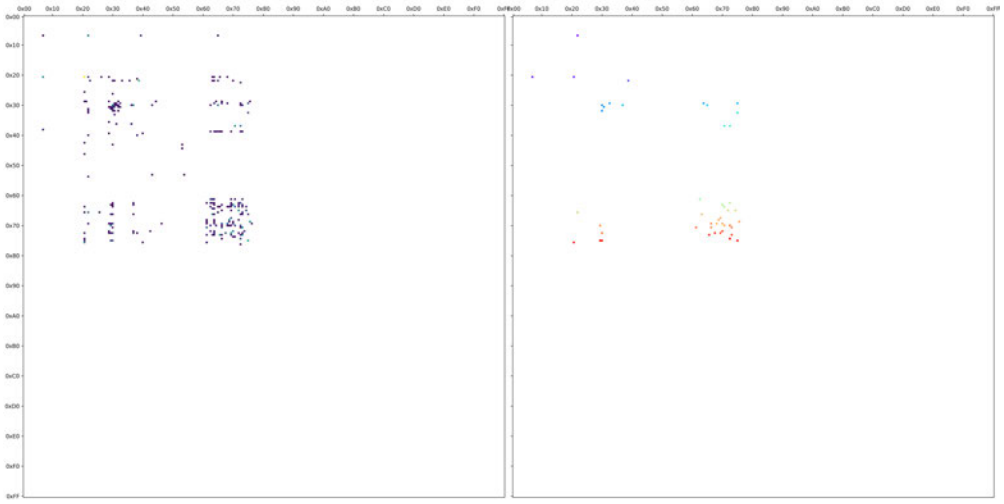
(e) Sektor 5



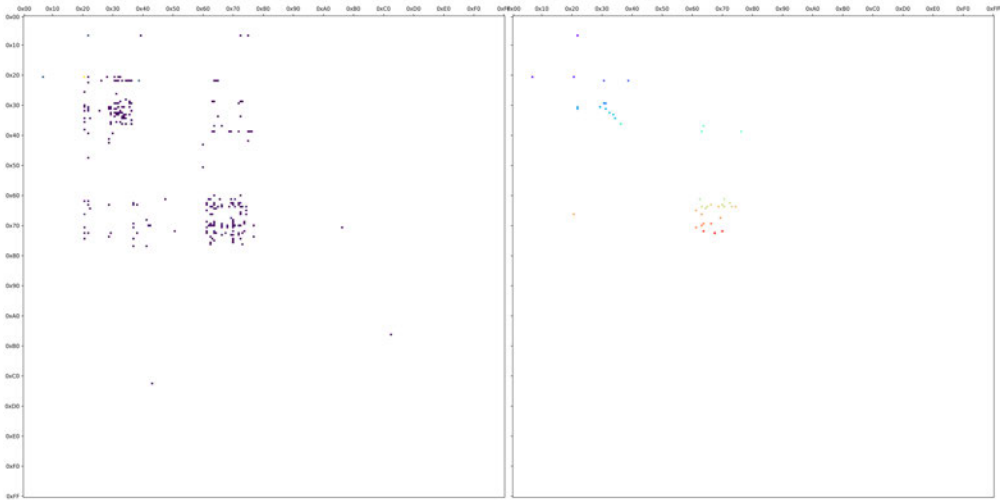
(f) Sektor 6

Abbildung B-14: Digramm- und Regionendarstellung der Datei 0003-pptx.pptx [67], fortgesetzt

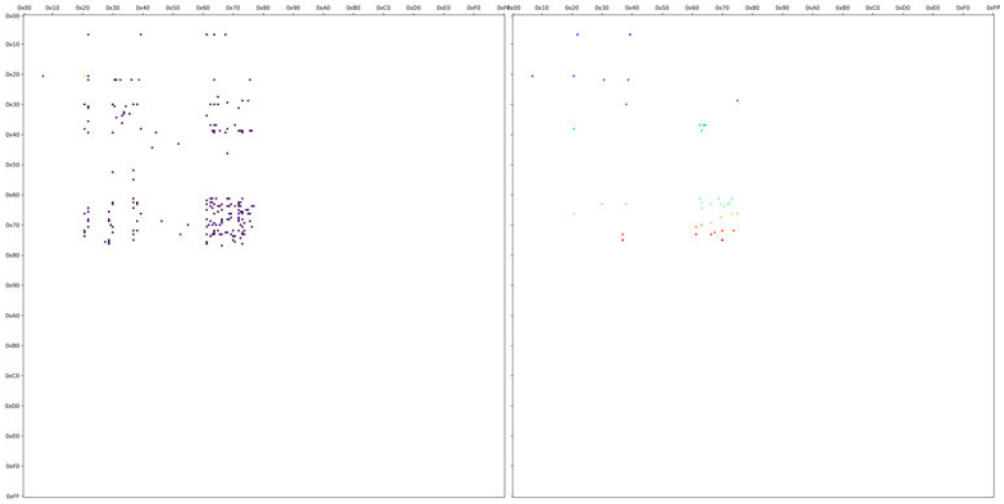
Datei 0003-svg-from-web.svg



(a) Sektor 1

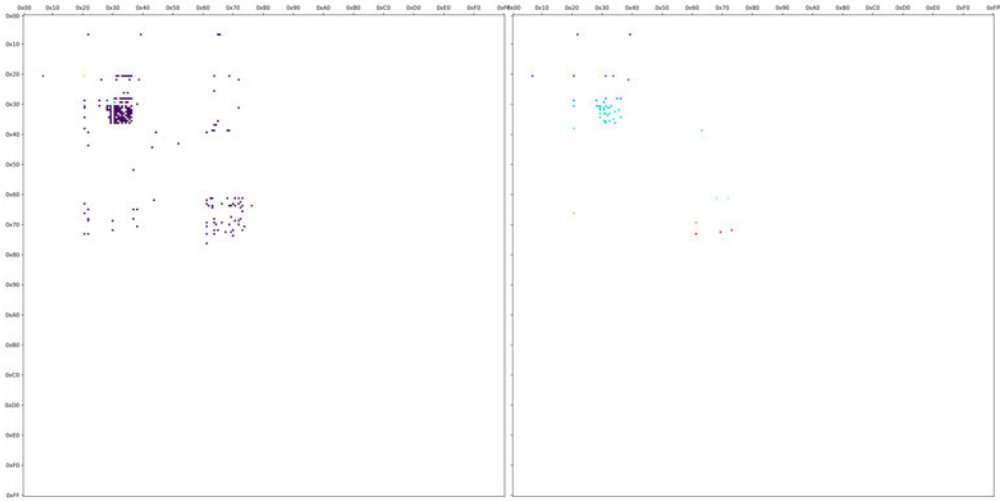


(b) Sektor 2

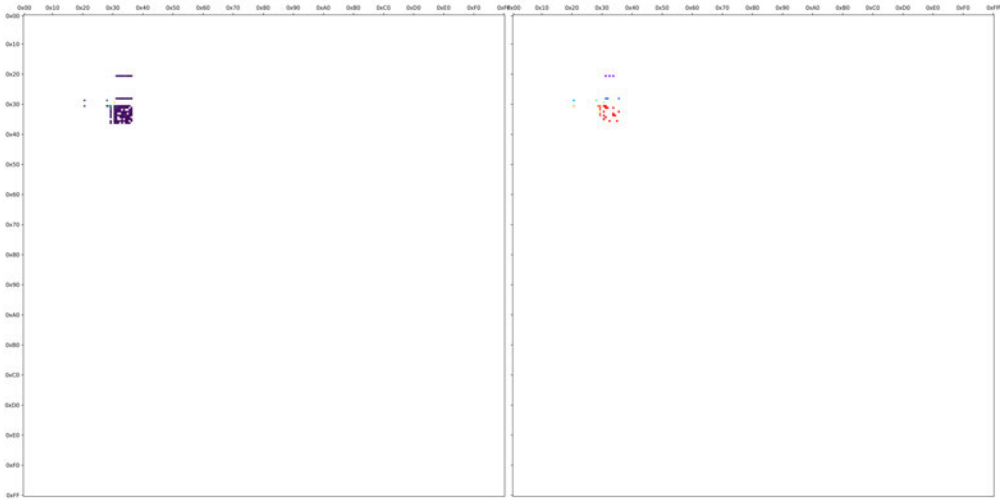


(c) Sektor 3

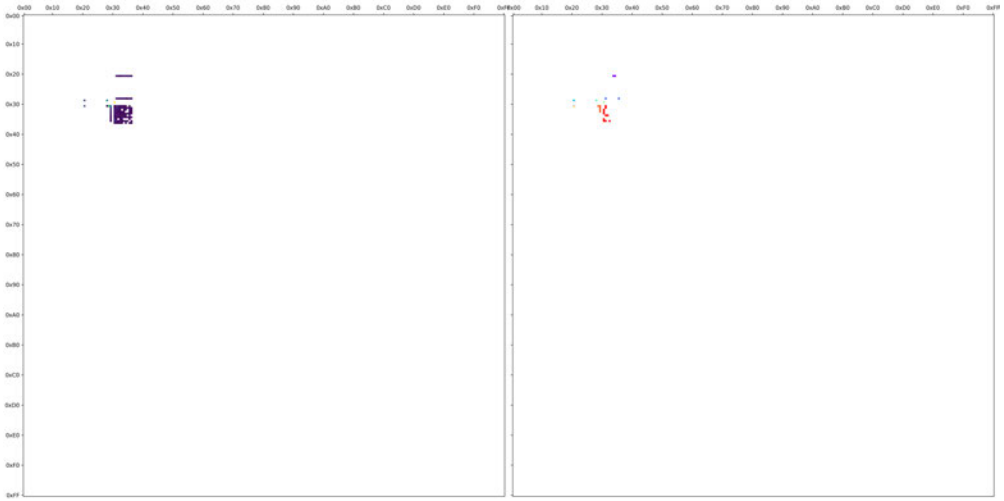
Abbildung B-15: Digramm- und Regionendarstellung der Datei 0003-svg-from-web.svg [67]



(d) Sektor 4



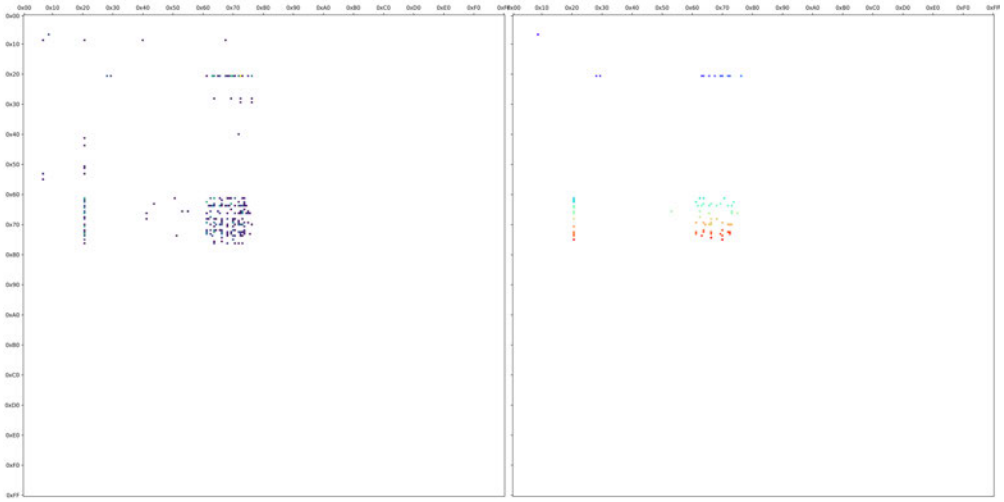
(e) Sektor 5



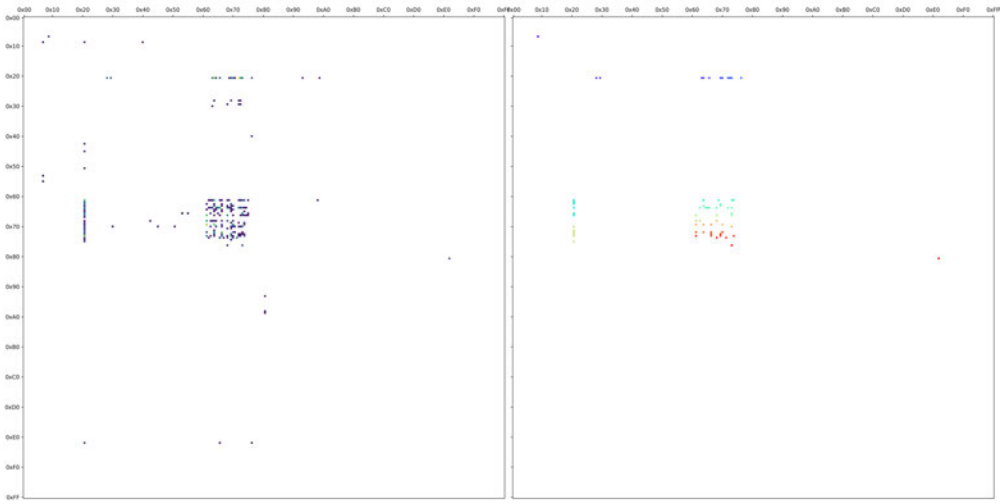
(f) Sektor 6

Abbildung B-15: Digramm- und Regionendarstellung der Datei 0003-svg-from-web.svg [67], fortgesetzt

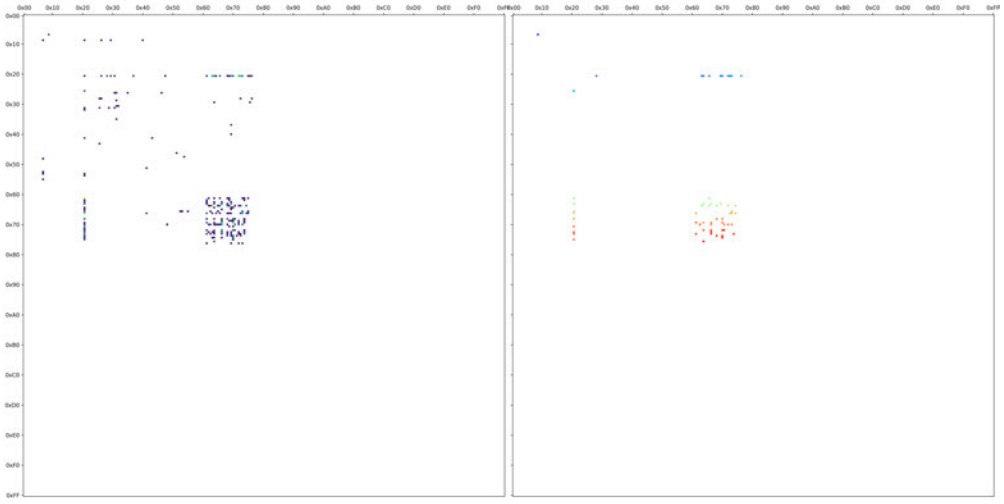
Datei 0003-txt.txt



(a) Sektor 1

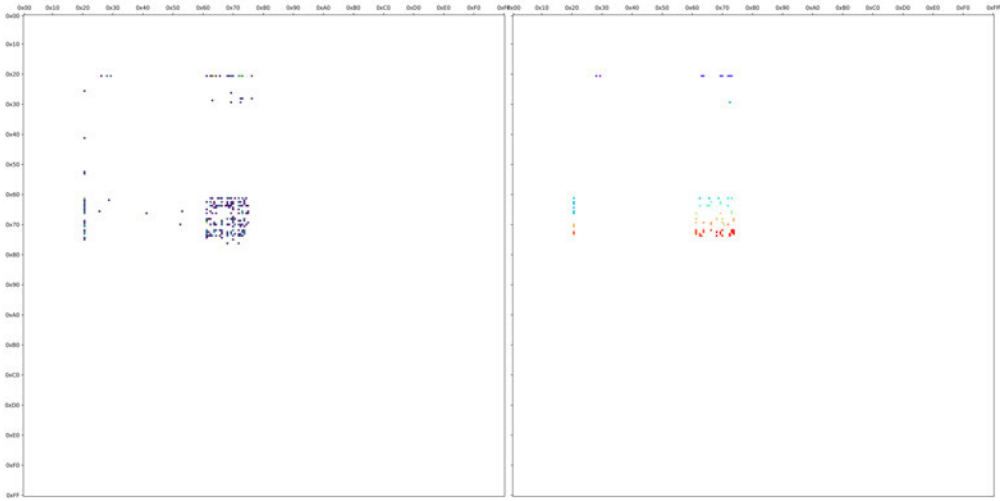


(b) Sektor 2

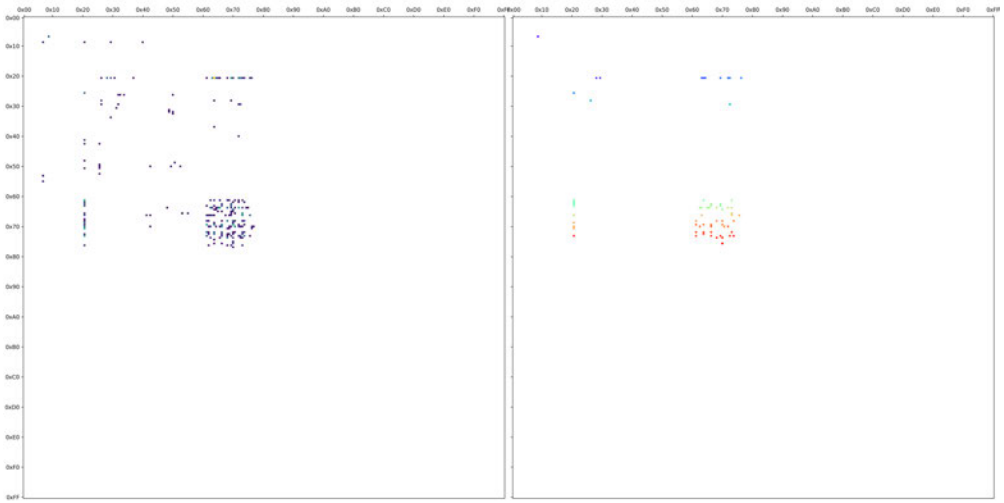


(c) Sektor 3

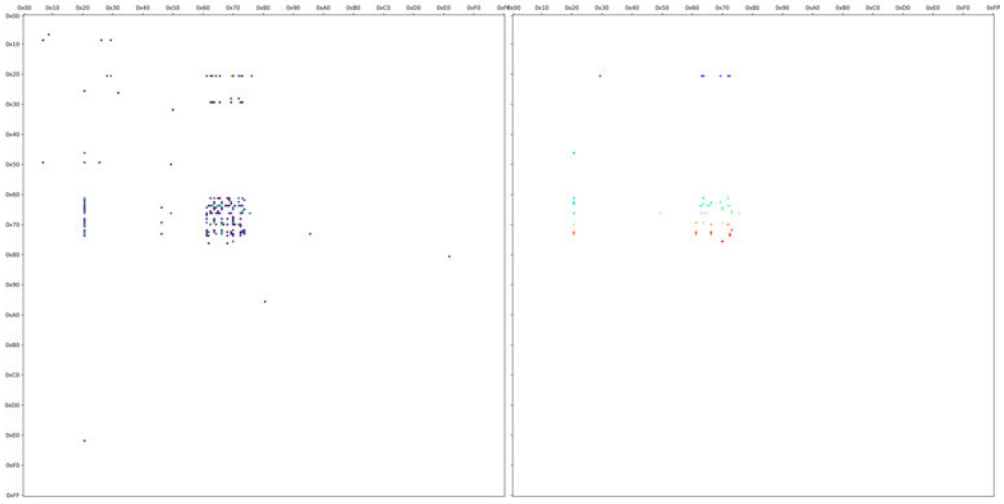
Abbildung B-16: Digramm- und Regionendarstellung der Datei 0003-txt.txt [67]



(d) Sektor 4



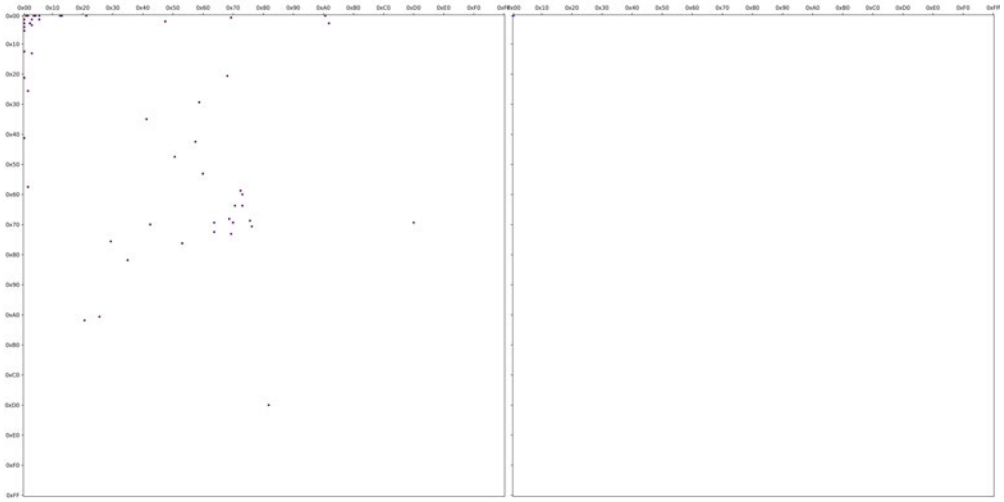
(e) Sektor 5



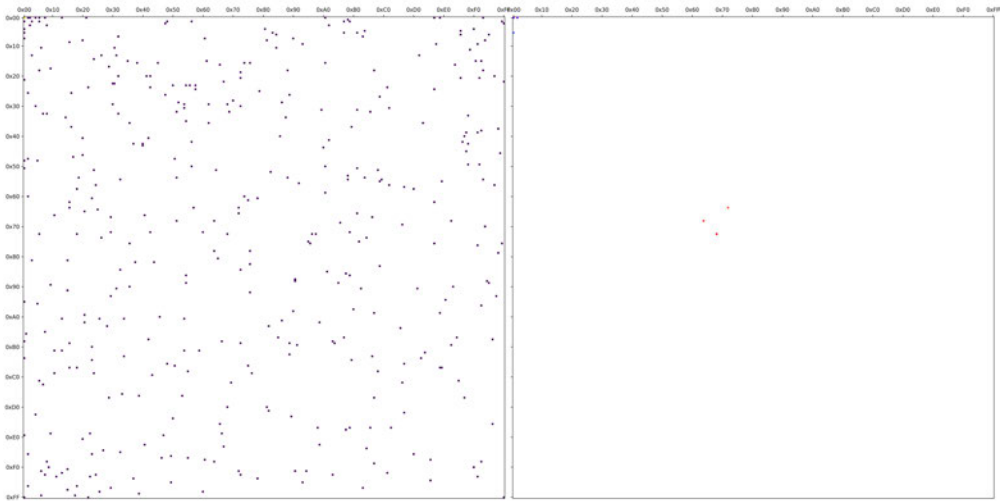
(f) Sektor 6

Abbildung B-16: Digramm- und Regionendarstellung der Datei 0003-txt.txt [67], fortgesetzt

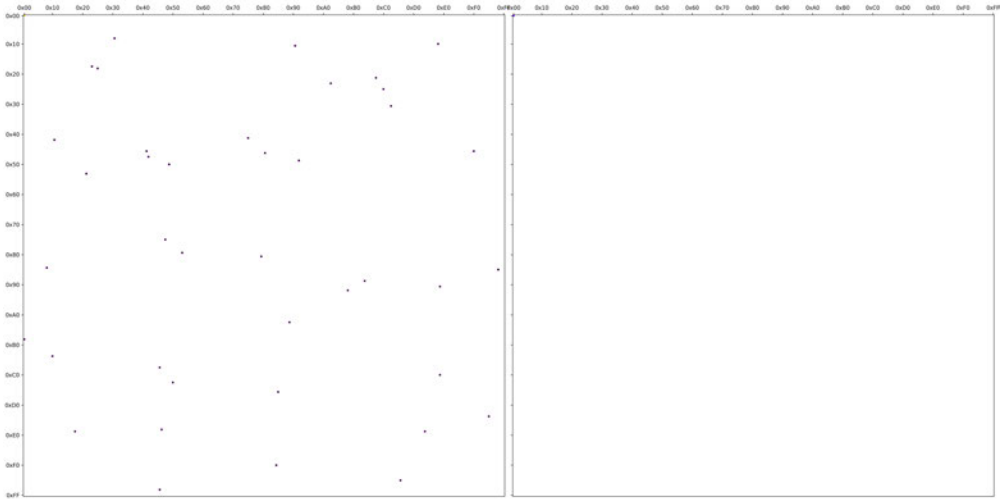
Datei 0003-xlsx.xlsx



(a) Sektor 1

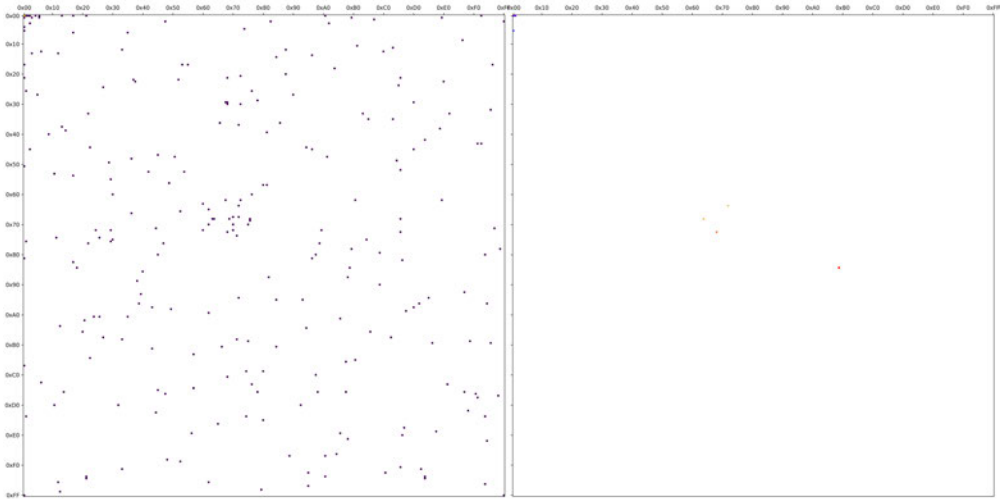


(b) Sektor 2

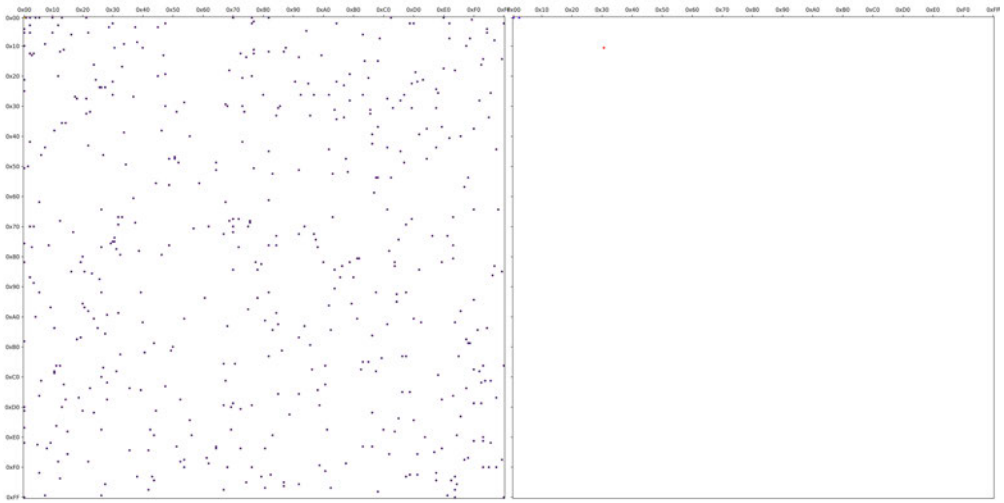


(c) Sektor 3

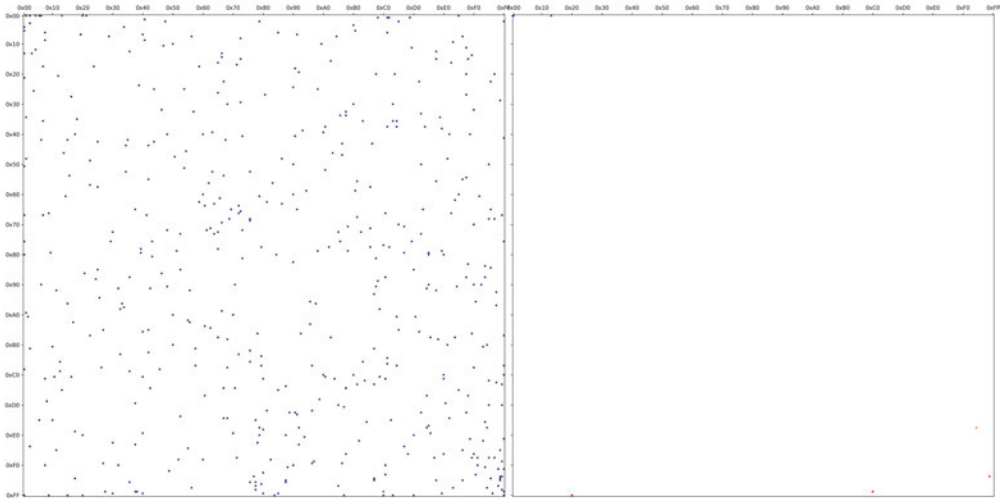
Abbildung B-17: Digramm- und Regionendarstellung der Datei 0003-xlsx.xlsx [67]



(d) Sektor 4



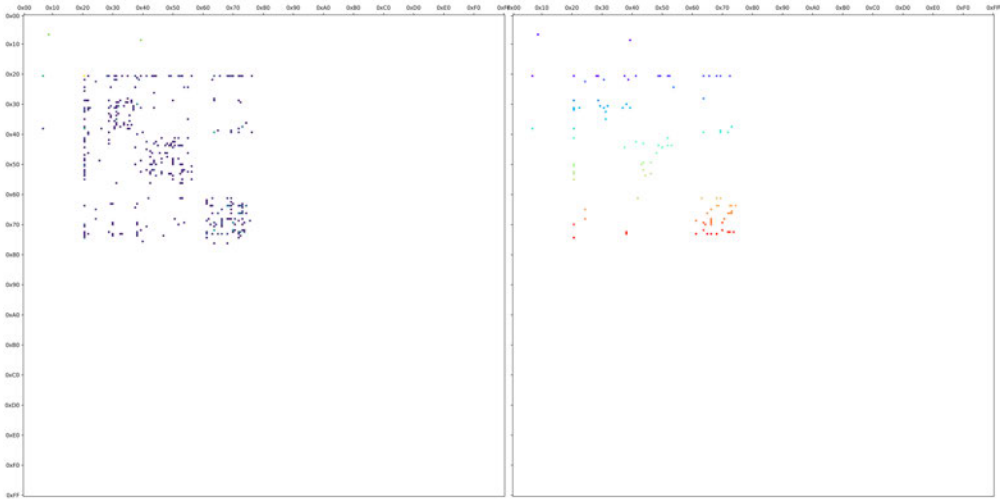
(e) Sektor 5



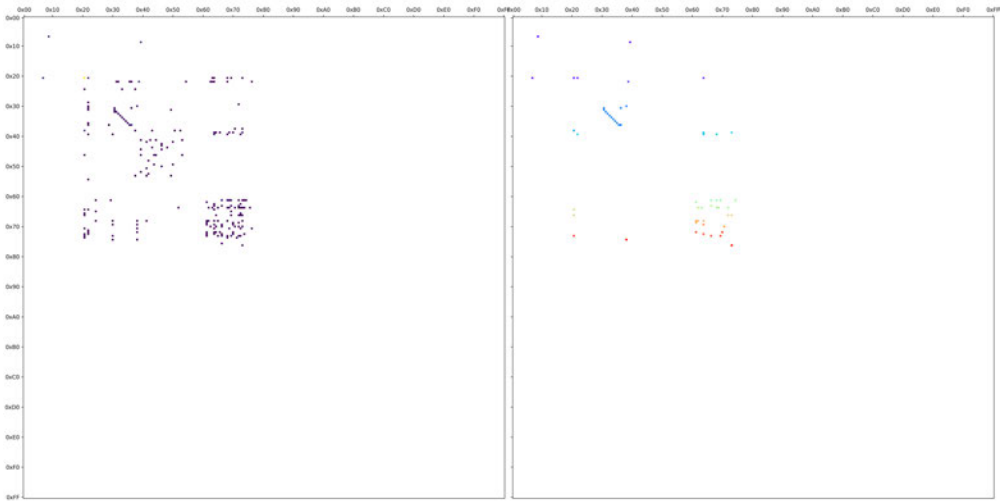
(f) Sektor 6

Abbildung B-17: Digramm- und Regionendarstellung der Datei 0003-xlsx.xlsx [67], fortgesetzt

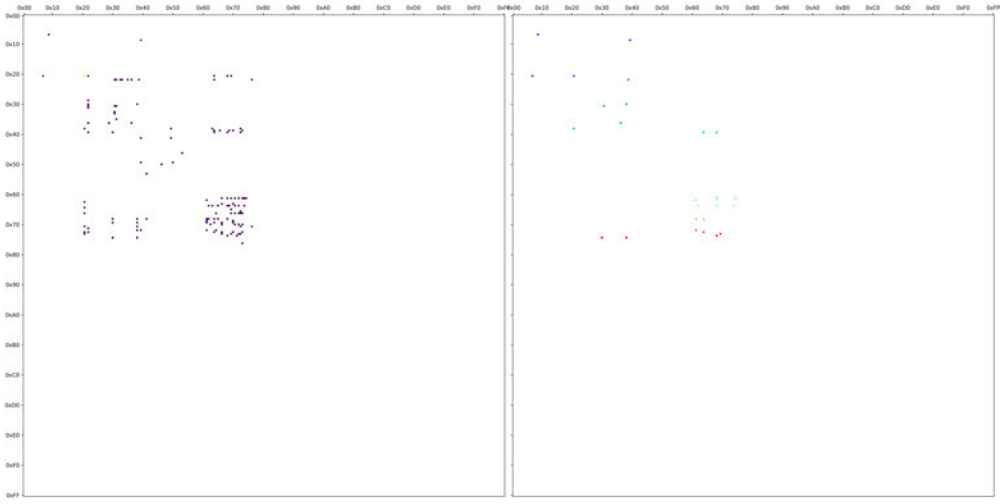
Datei 0003-xml.xml



(a) Sektor 1

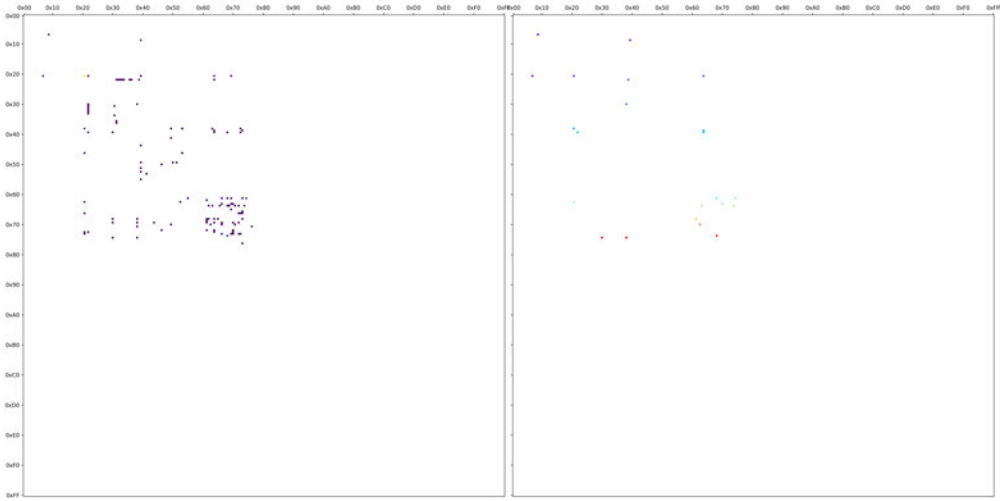


(b) Sektor 2

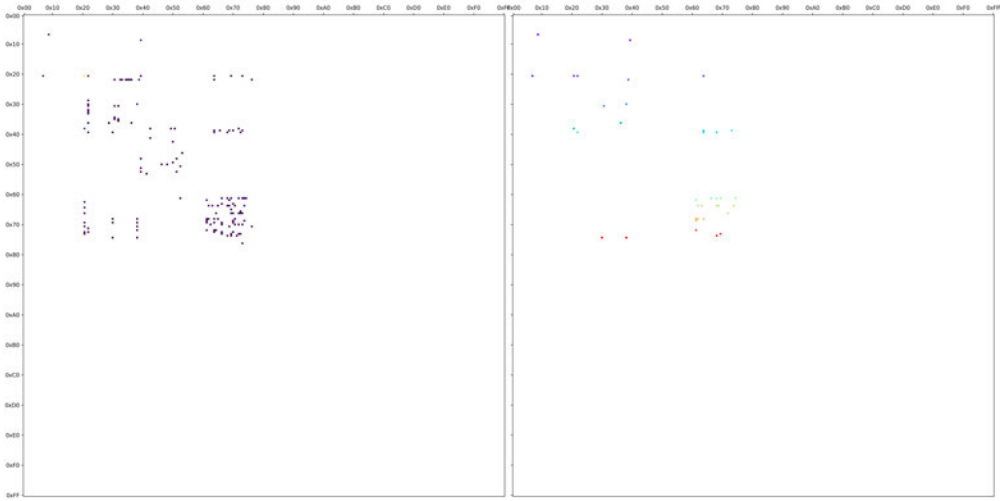


(c) Sektor 3

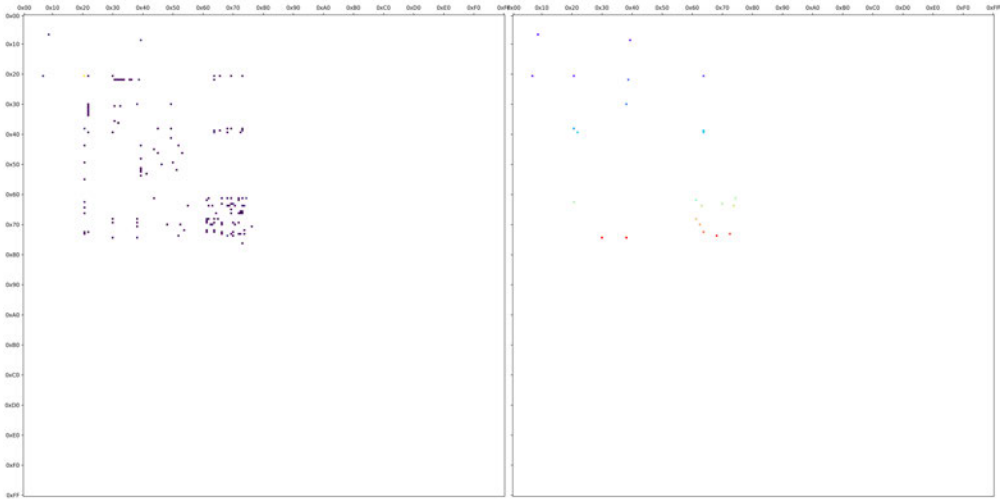
Abbildung B-18: Digramm- und Regionendarstellung der Datei 0003-xml.xml [67]



(d) Sektor 4



(e) Sektor 5



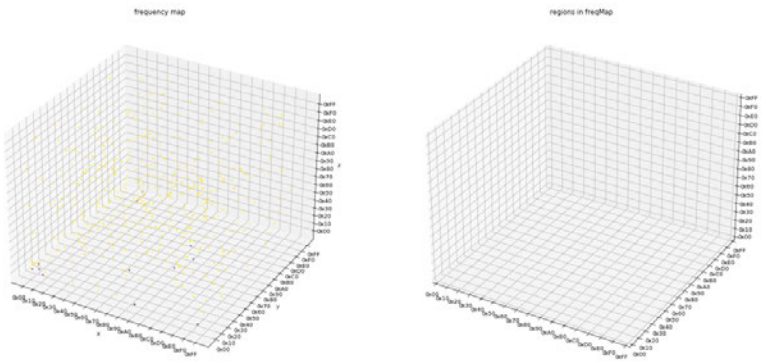
(f) Sektor 6

Abbildung B-18: Digramm- und Regionendarstellung der Datei 0003-xml.xml [67], fortgesetzt

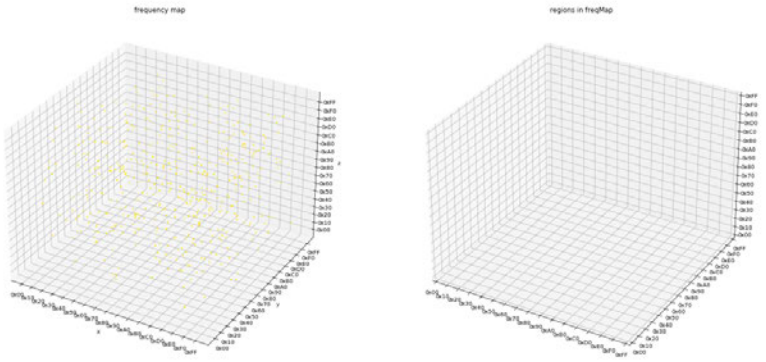
Anhang C: Trigramm-Darstellungen

Nachfolgend sind die im Rahmen der visuellen Bewertung erstellten Visualisierungen der Trigramm- und der Regionen-Analyse abgebildet. Analog zu Abschnitt B zeigen die Abbildungen C-1 bis C-16 für jeden Sektor die Trigramm-Visualisierung auf der linken und die Regionen-Darstellung auf der rechten Seite. Es wurden für diese Darstellungen dieselben Colormaps verwendet wie zuvor. Die Bilder sind begleitend zur visuellen Bewertung in den Abschnitten 5.3.3.3 und 5.3.3.4 zu betrachten.

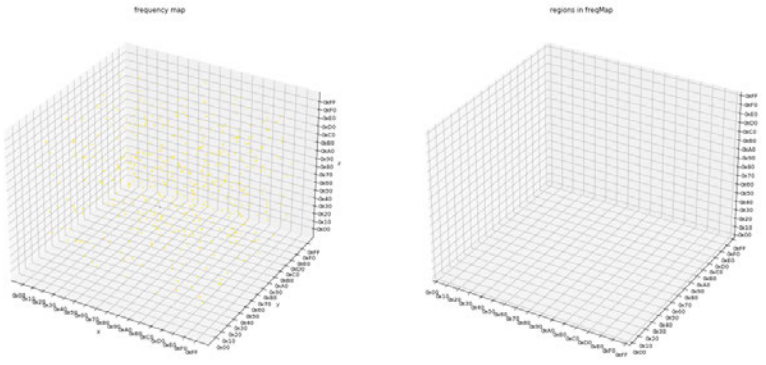
Datei 0003-apk.apk



(a) Sektor 1

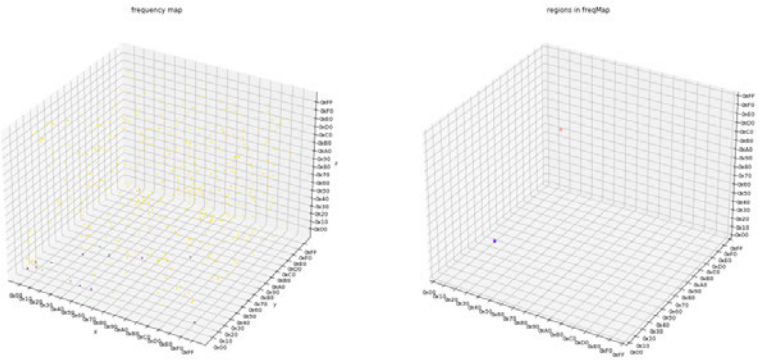


(b) Sektor 2

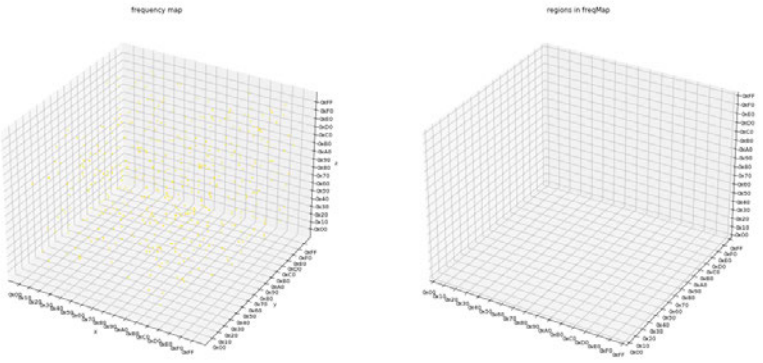


(c) Sektor 3

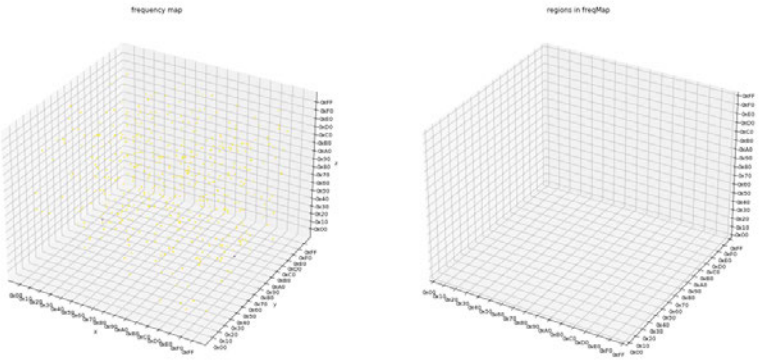
Abbildung C-1: Trigramm- und Regionendarstellung der Datei 0003-apk.apk [67]



(d) Sektor 4



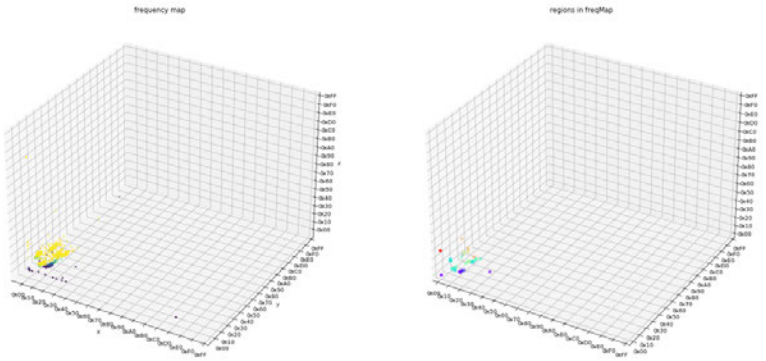
(e) Sektor 5



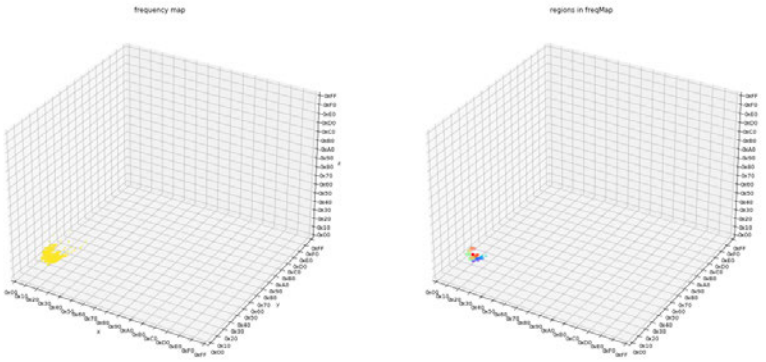
(f) Sektor 6

Abbildung C-1: Trigramm- und Regionendarstellung der Datei 0003-apk.apk [67], fortgesetzt

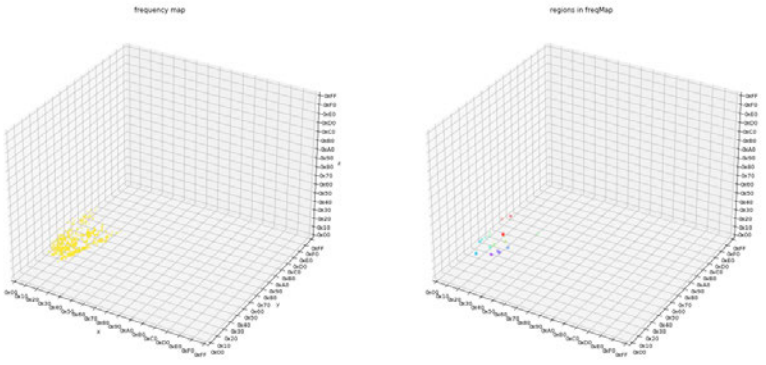
Datei 0003-bmp.bmp



(a) Sektor 1

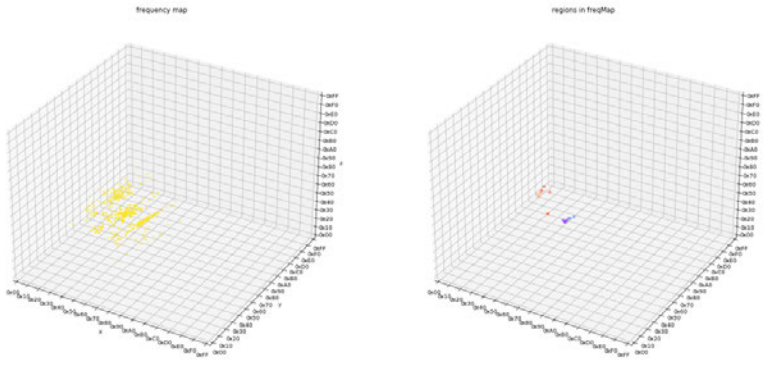


(b) Sektor 2

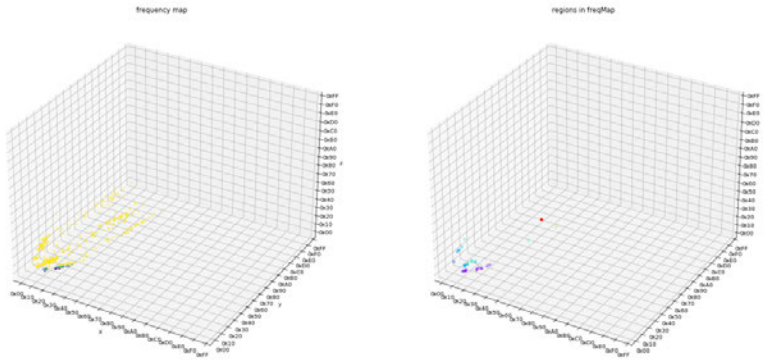


(c) Sektor 3

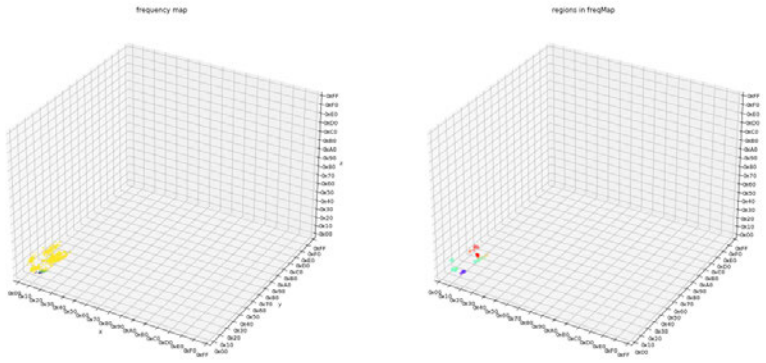
Abbildung C-2: Trigramm- und Regionendarstellung der Datei 0003-bmp.bmp [67]



(d) Sektor 4



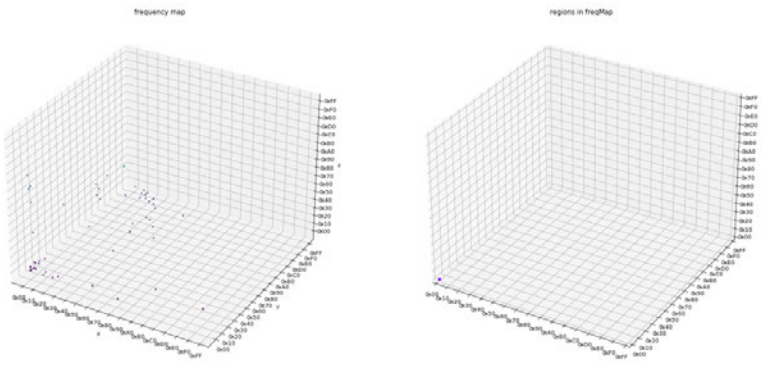
(e) Sektor 5



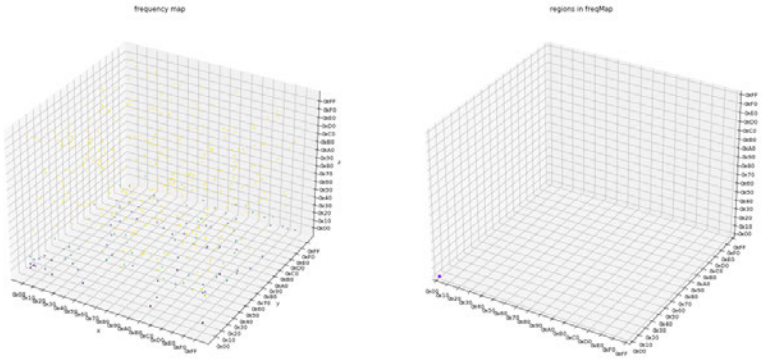
(f) Sektor 6

Abbildung C-2: Trigramm- und Regionendarstellung der Datei 0003-bmp.bmp [67], fortgesetzt

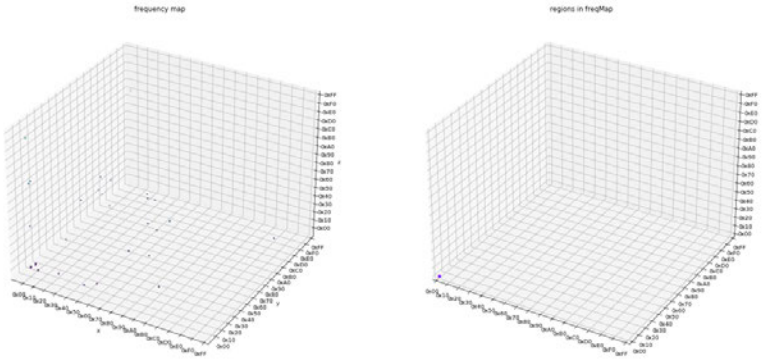
Datei 0003-docx.docx



(a) Sektor 1

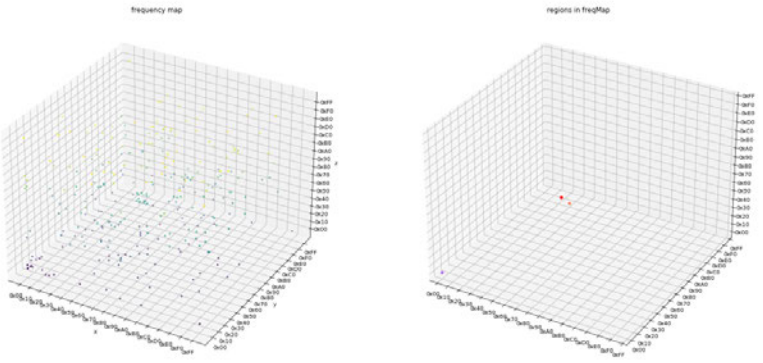


(b) Sektor 2

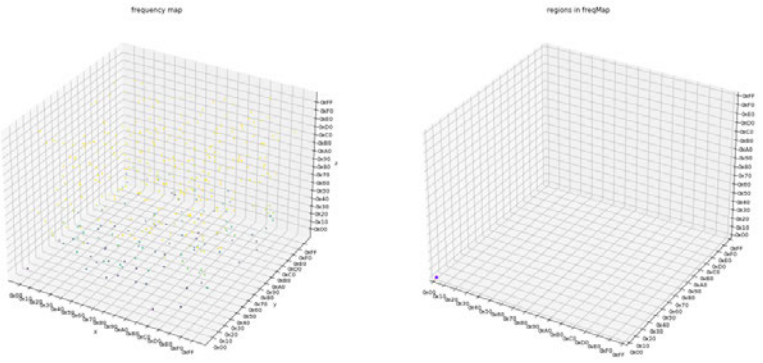


(c) Sektor 3

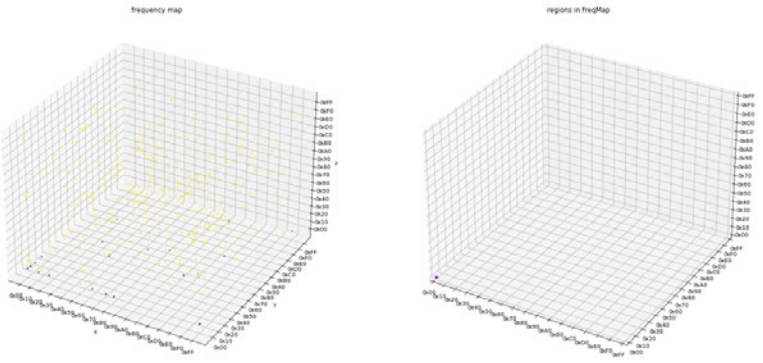
Abbildung C-3: Trigramm- und Regionendarstellung der Datei 0003-docx.docx [67]



(d) Sektor 4



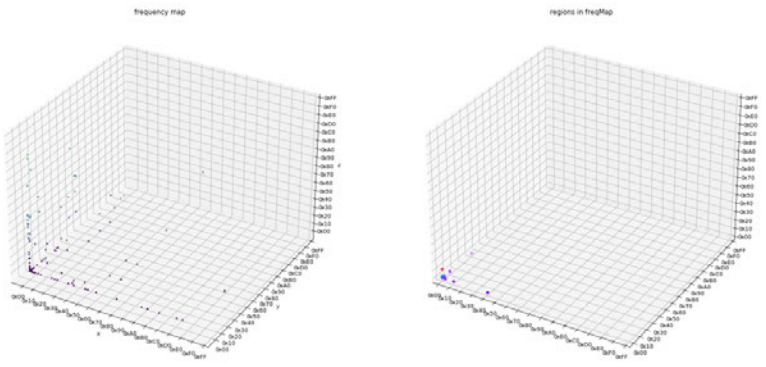
(e) Sektor 5



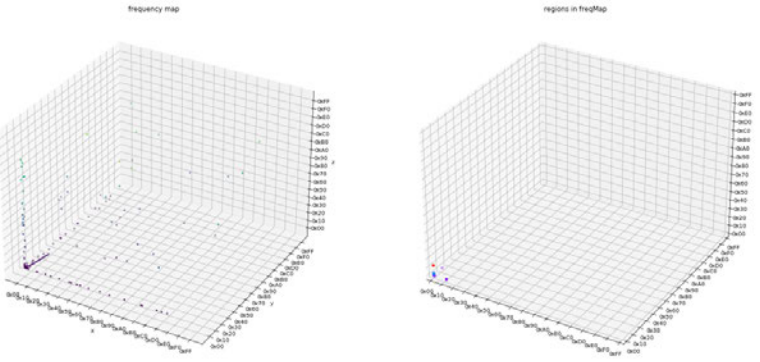
(f) Sektor 6

Abbildung C-3: Trigramm- und Regionendarstellung der Datei 0003-docx.docx [67], fortgesetzt

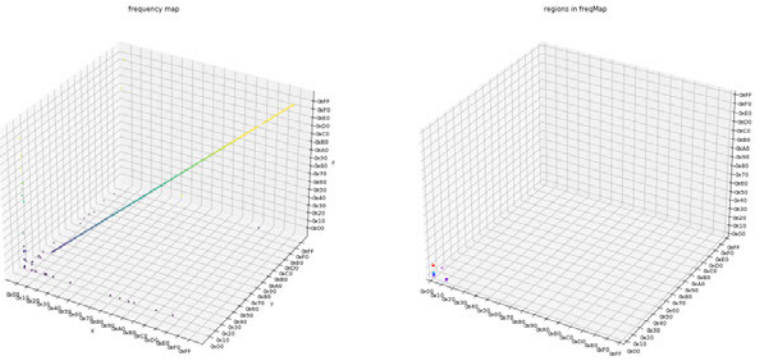
Datei 0003-so.so



(a) Sektor 1

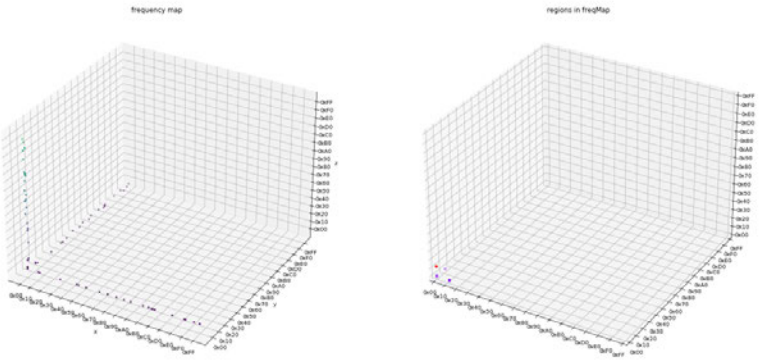


(b) Sektor 2

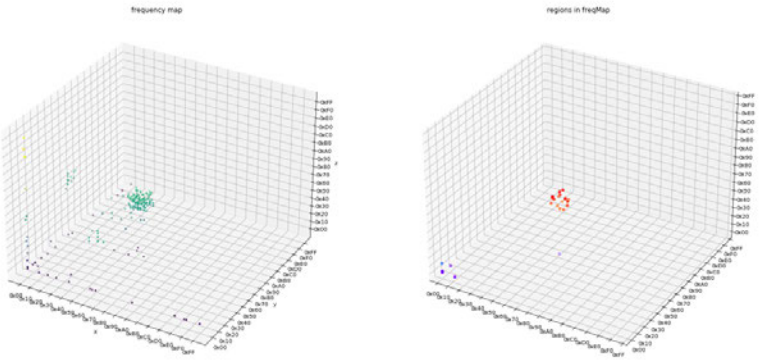


(c) Sektor 3

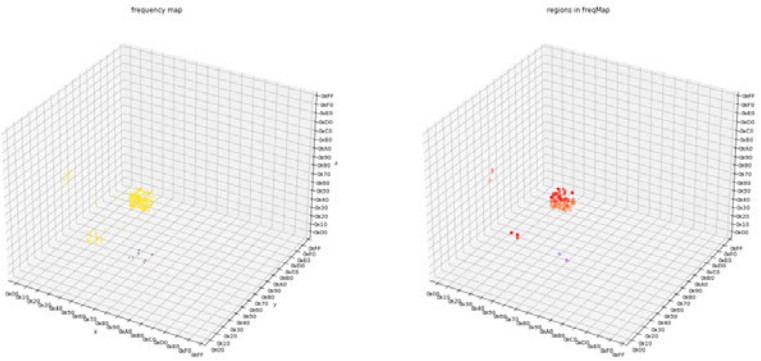
Abbildung C-4: Trigramm- und Regionendarstellung der Datei 0003-so.so [67]



(d) Sektor 4



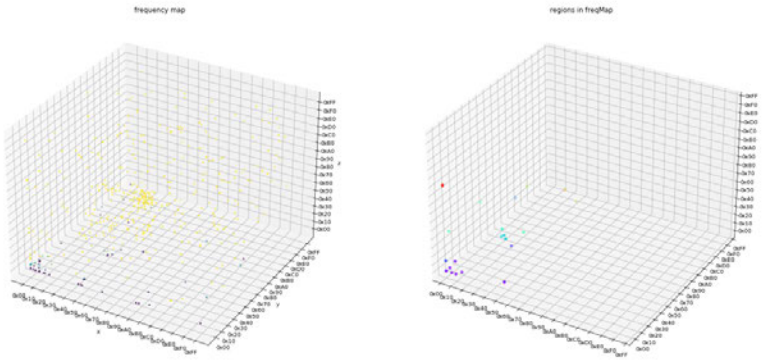
(e) Sektor 5



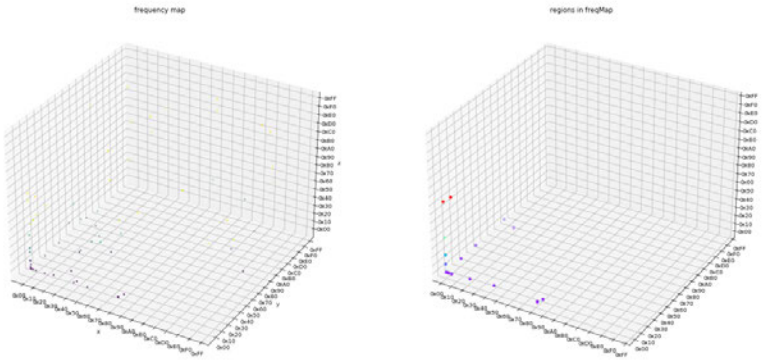
(f) Sektor 6

Abbildung C-4: Trigramm- und Regionendarstellung der Datei 0003-so.so [67], fortgesetzt

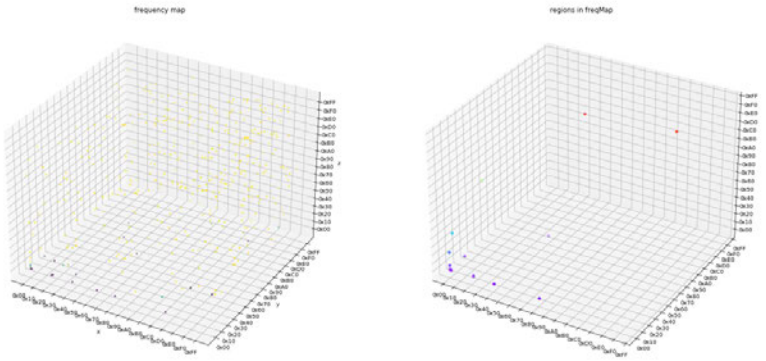
Datei 0003-epub.epub



(a) Sektor 1

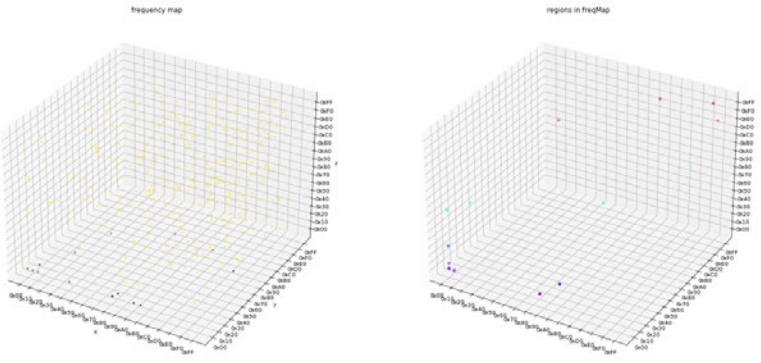


(b) Sektor 2

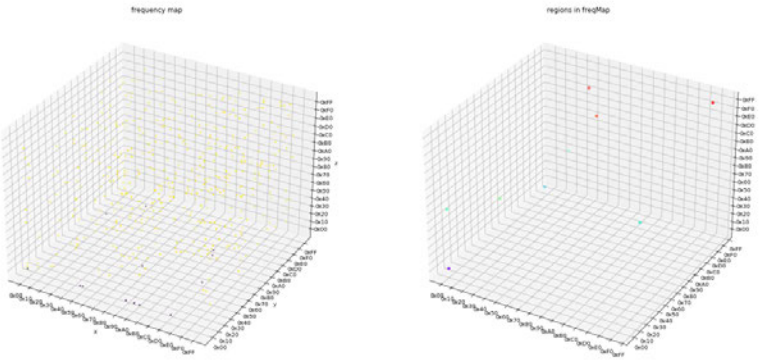


(c) Sektor 3

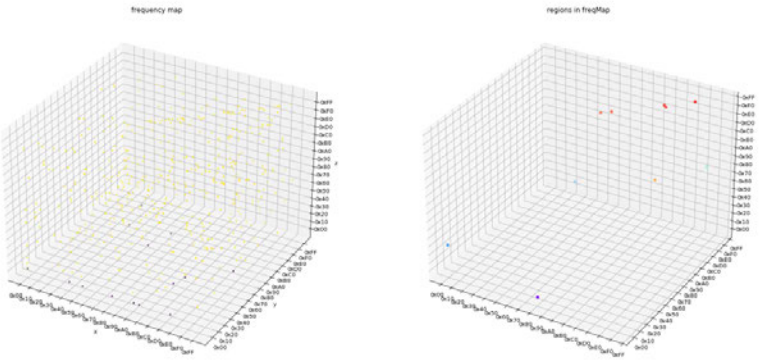
Abbildung C-5: Trigramm- und Regionendarstellung der Datei 0003-epub.epub [67]



(d) Sektor 4



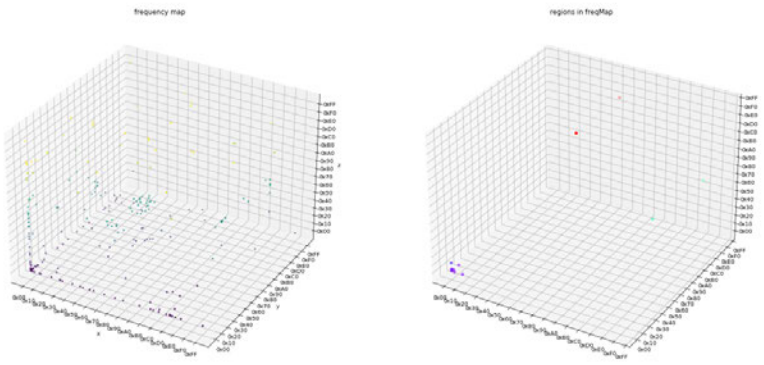
(e) Sektor 5



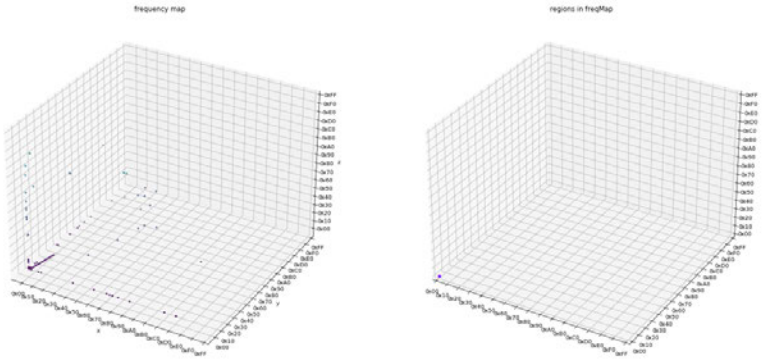
(f) Sektor 6

Abbildung C-5: Trigramm- und Regionendarstellung der Datei 0003-epub.epub [67], fortgesetzt

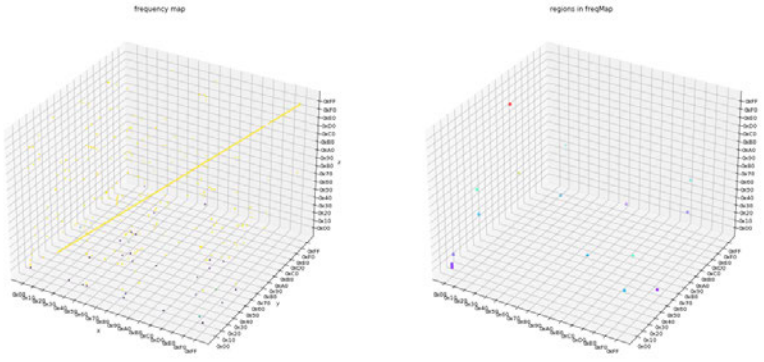
Datei 0003-exe.exe



(a) Sektor 1

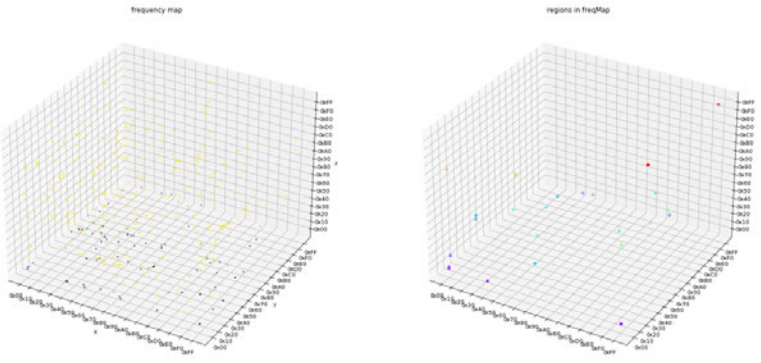


(b) Sektor 2

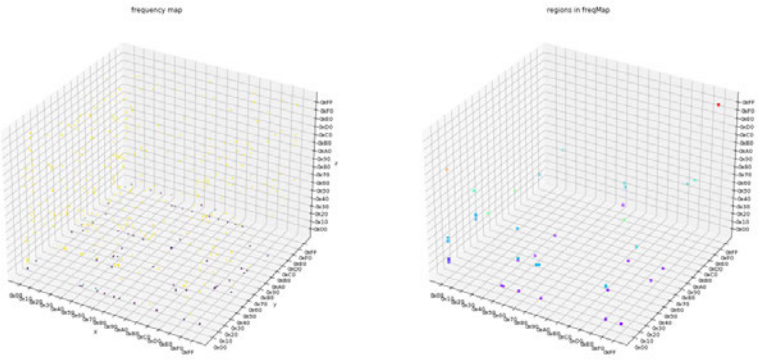


(c) Sektor 3

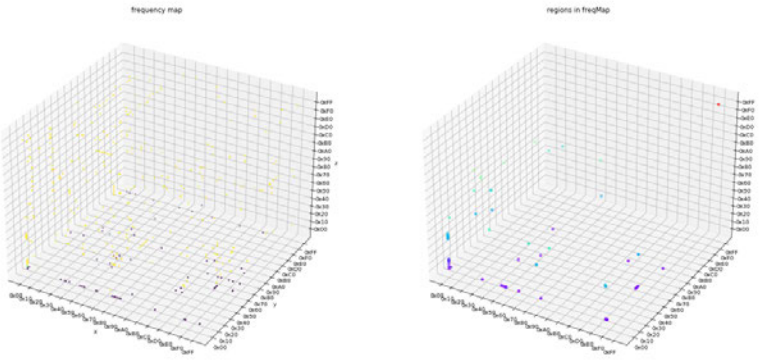
Abbildung C-6: Trigramm- und Regionendarstellung der Datei 0003-exe.exe [67]



(d) Sektor 4



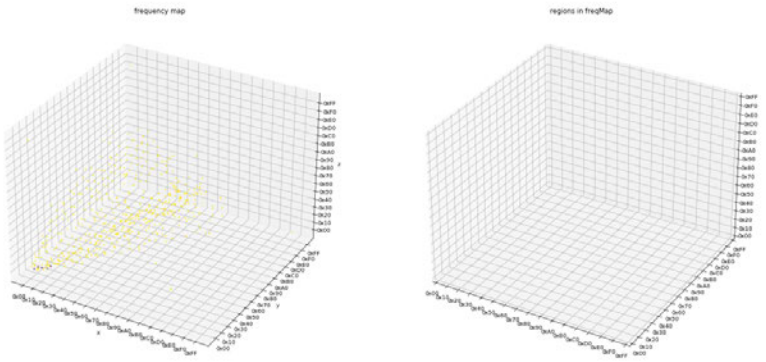
(e) Sektor 5



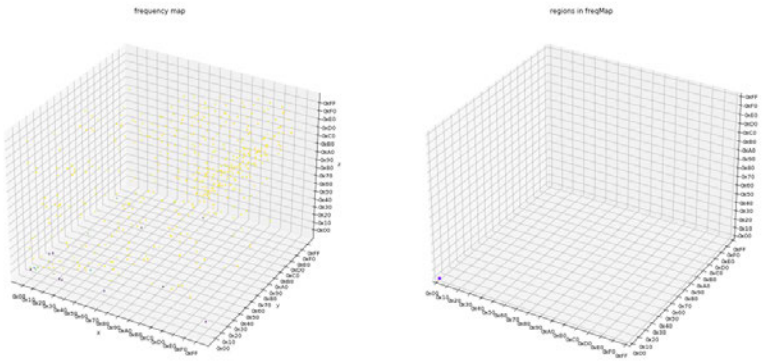
(f) Sektor 6

Abbildung C-6: Trigramm- und Regionendarstellung der Datei 0003-exe.exe [67], fortgesetzt

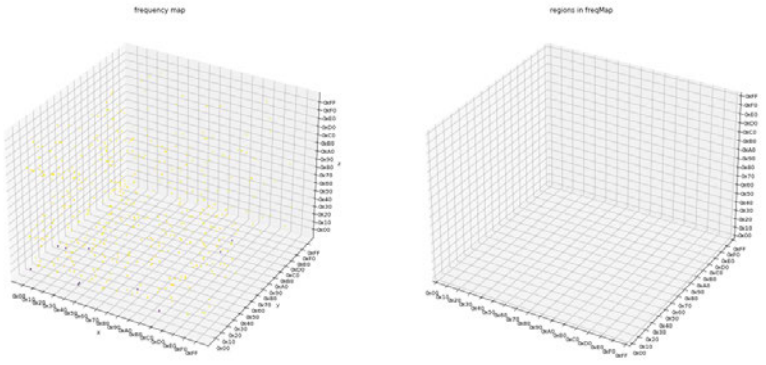
Datei 0003-gif.gif



(a) Sektor 1

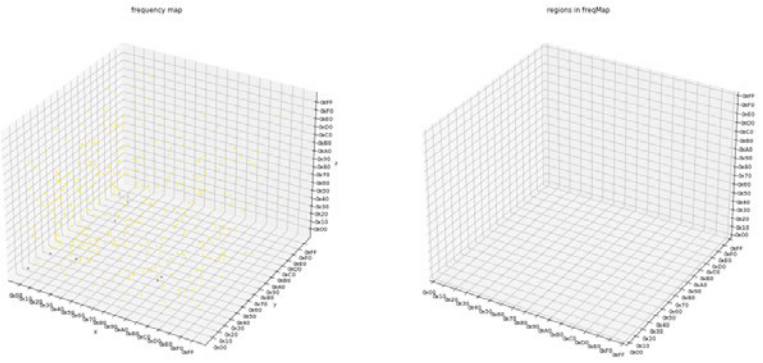


(b) Sektor 2

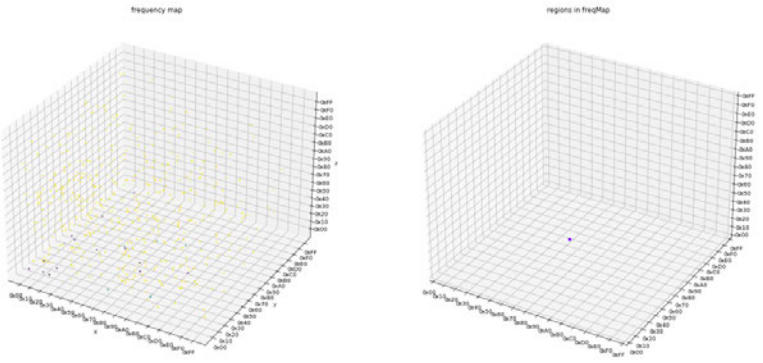


(c) Sektor 3

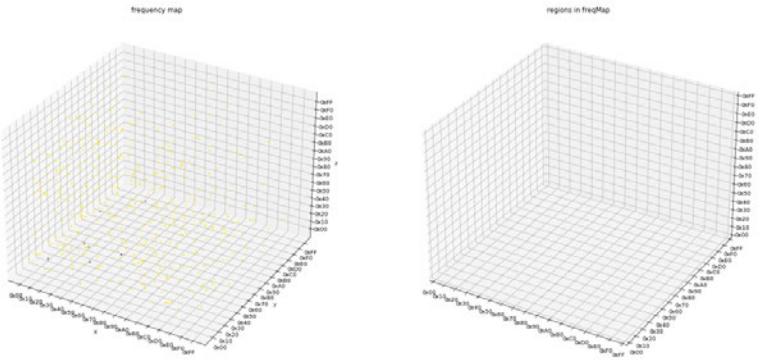
Abbildung C-7: Trigramm- und Regionendarstellung der Datei 0003-gif.gif [67]



(d) Sektor 4



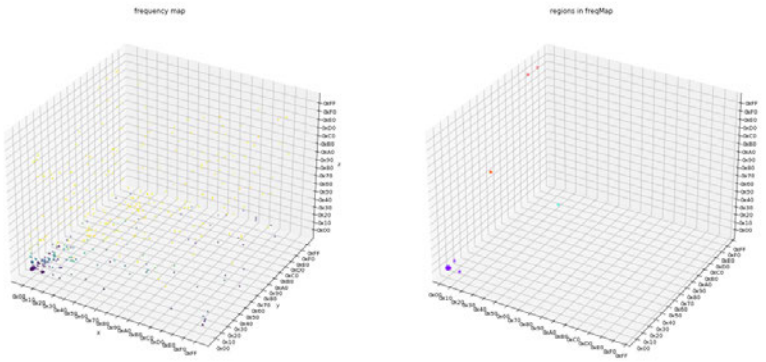
(e) Sektor 5



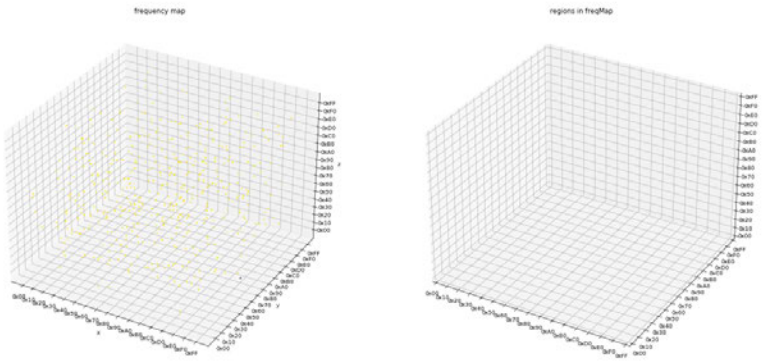
(f) Sektor 6

Abbildung C-7: Trigramm- und Regionendarstellung der Datei 0003-gif.gif [67], fortgesetzt

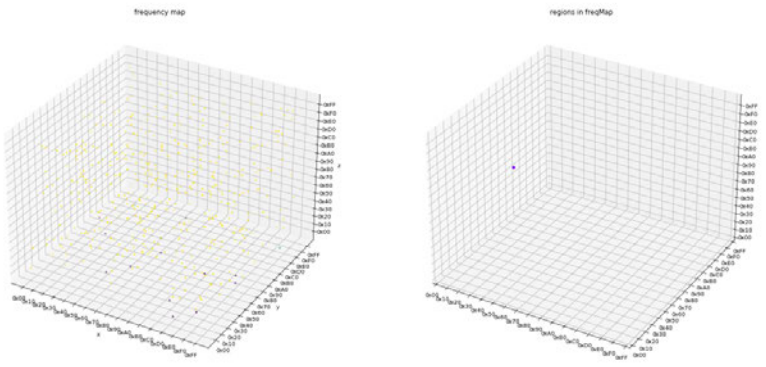
Datei 0003-jpg-q50.jpg



(a) Sektor 1

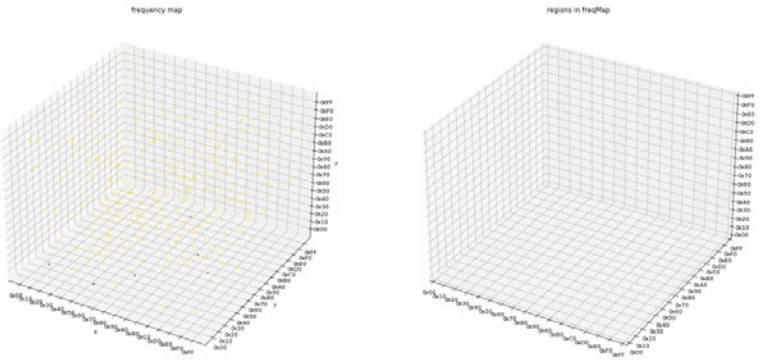


(b) Sektor 2

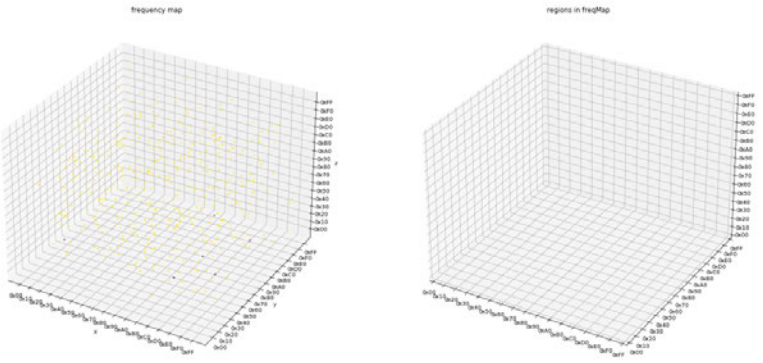


(c) Sektor 3

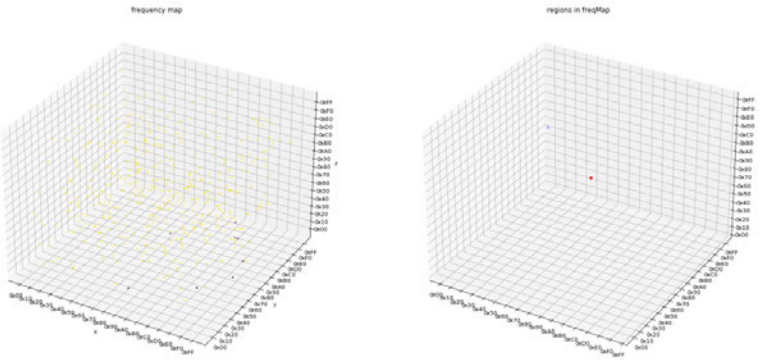
Abbildung C-8: Trigramm- und Regionendarstellung der Datei 0003-jpg-q50.jpg [67]



(d) Sektor 4



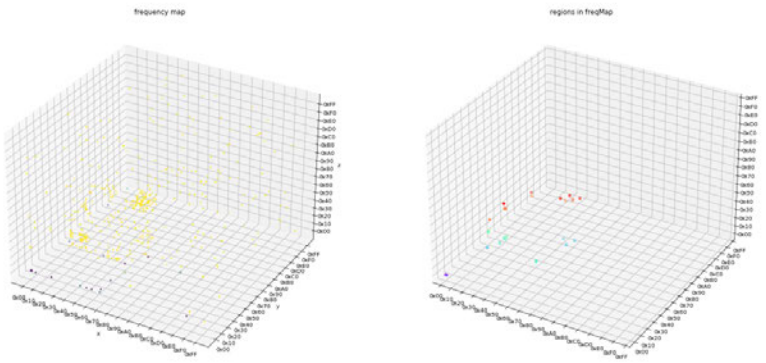
(e) Sektor 5



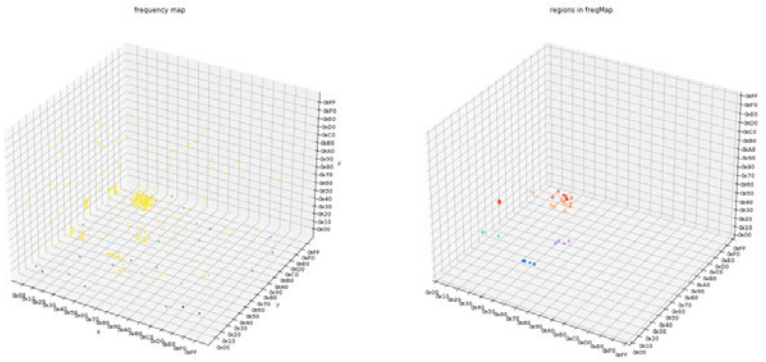
(f) Sektor 6

Abbildung C-8: Trigramm- und Regionendarstellung der Datei 0003-jpg-q50.jpg [67], fortgesetzt

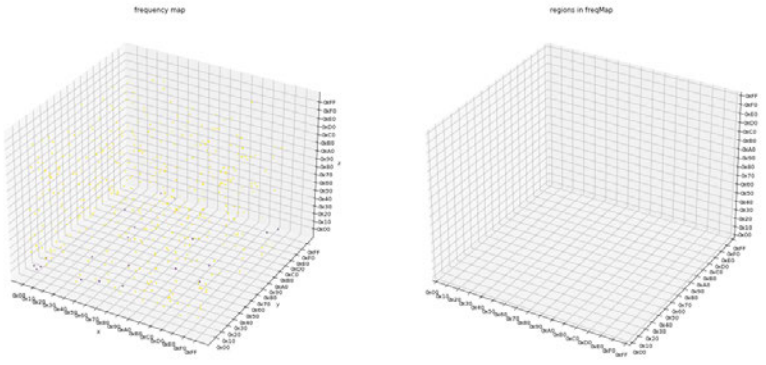
Datei 0003-mp4.mp4



(a) Sektor 1

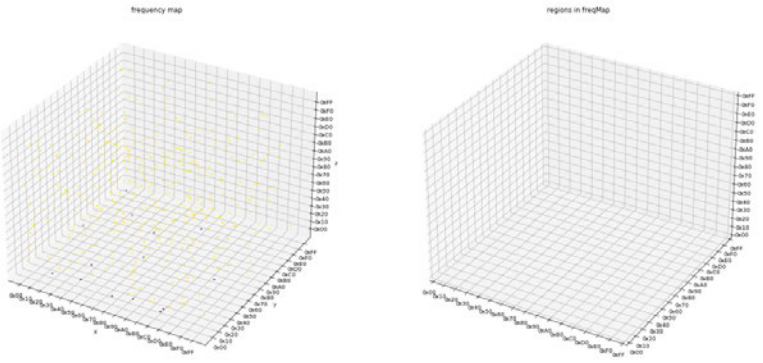


(b) Sektor 2

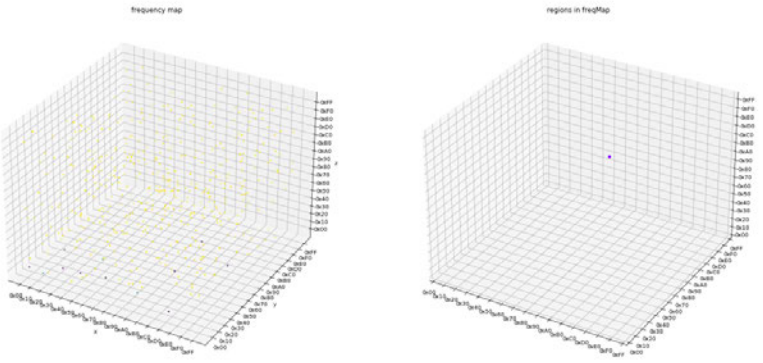


(c) Sektor 3

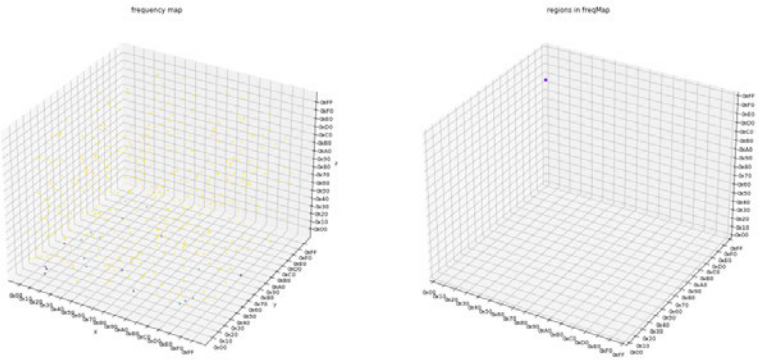
Abbildung C-9: Trigramm- und Regionendarstellung der Datei 0003-mp4.mp4 [67]



(d) Sektor 4



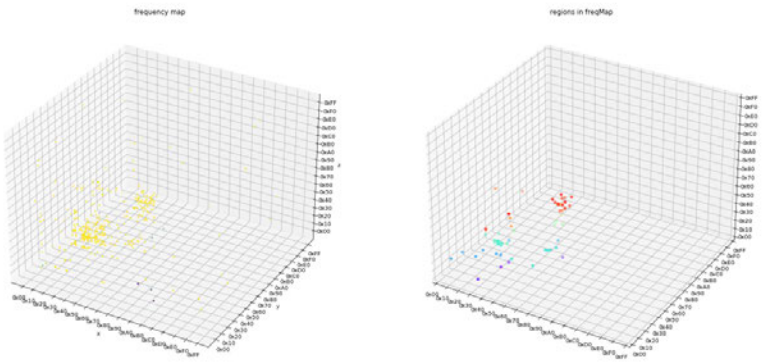
(e) Sektor 5



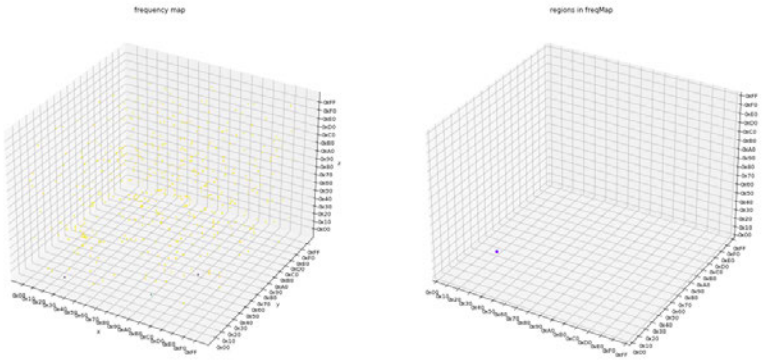
(f) Sektor 6

Abbildung C-9: Trigramm- und Regionendarstellung der Datei 0003-mp4.mp4 [67], fortgesetzt

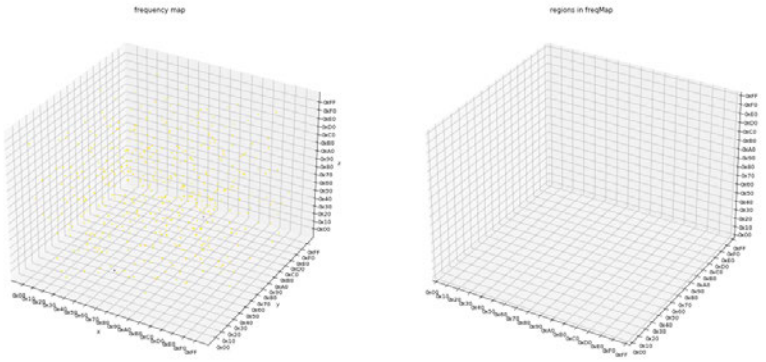
Datei 0003-pdf.pdf



(a) Sektor 1

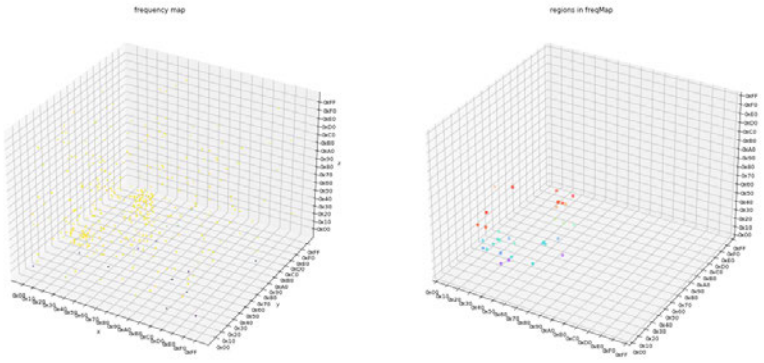


(b) Sektor 2

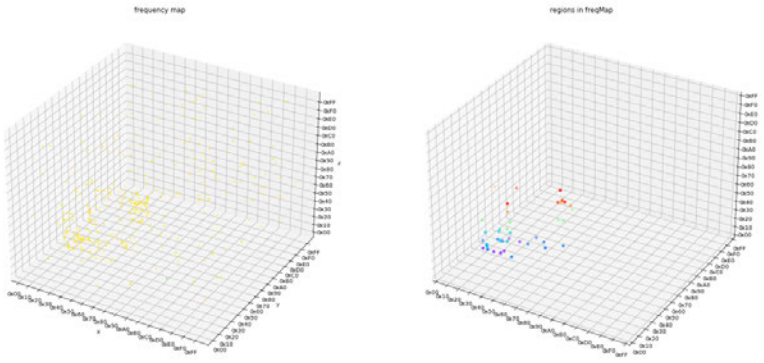


(c) Sektor 3

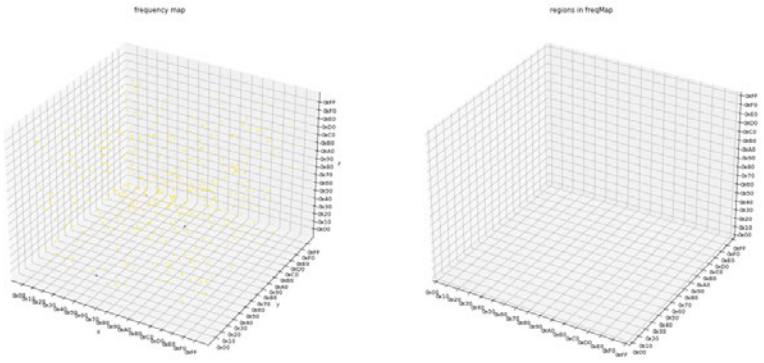
Abbildung C-10: Trigramm- und Regionendarstellung der Datei 0003-pdf.pdf [67]



(d) Sektor 4



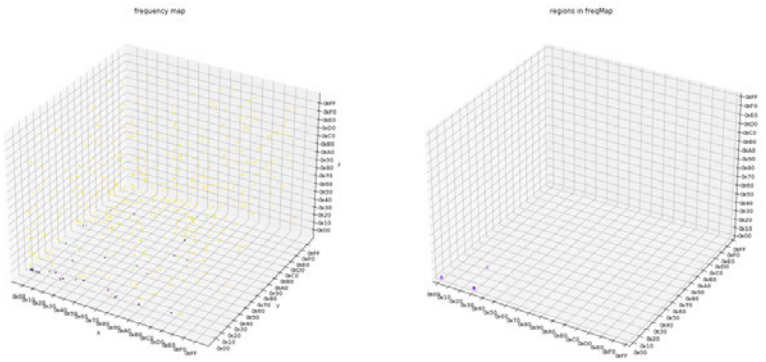
(e) Sektor 5



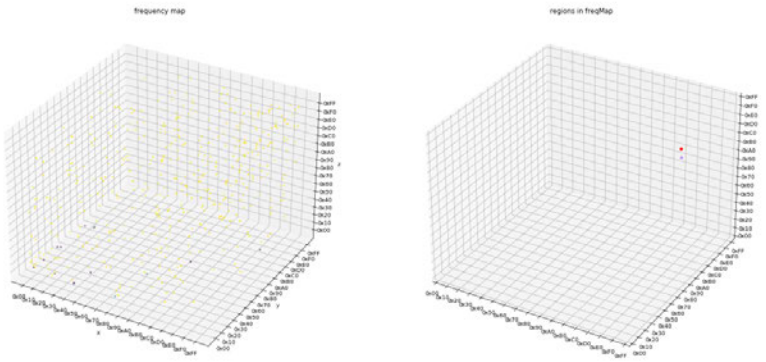
(f) Sektor 6

Abbildung C-10: Trigramm- und Regionendarstellung der Datei 0003-pdf.pdf [67], fortgesetzt

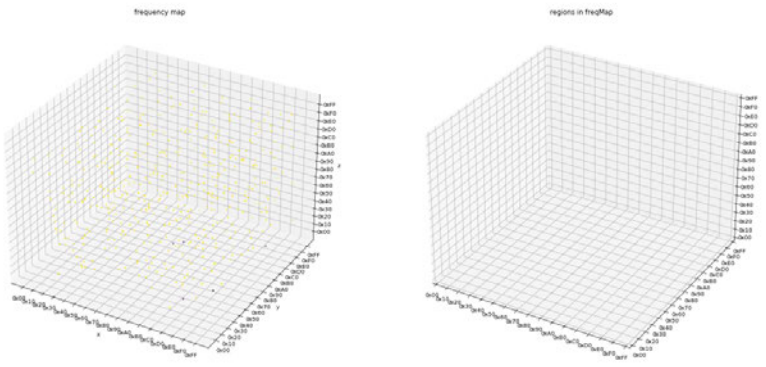
Datei 0003-png-c9.png



(a) Sektor 1

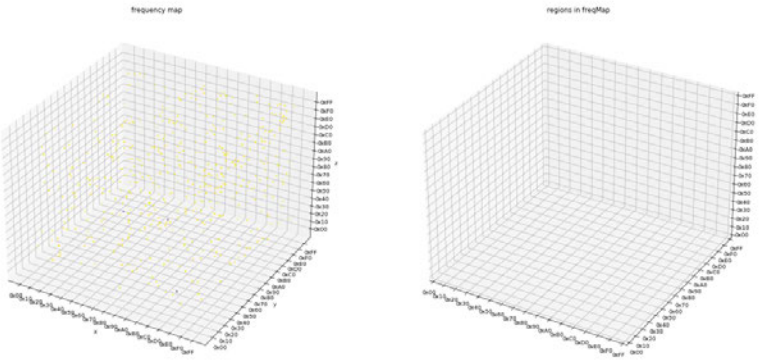


(b) Sektor 2

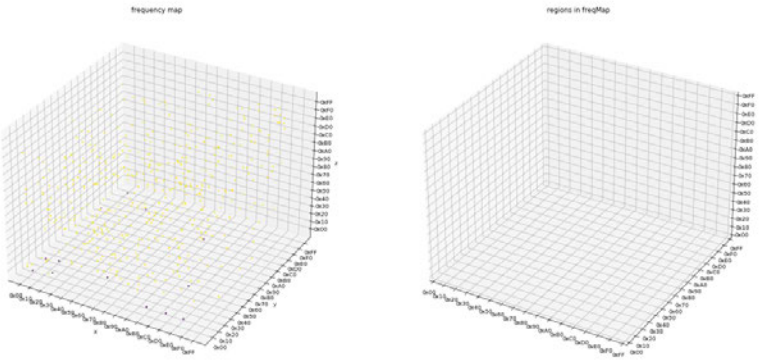


(c) Sektor 3

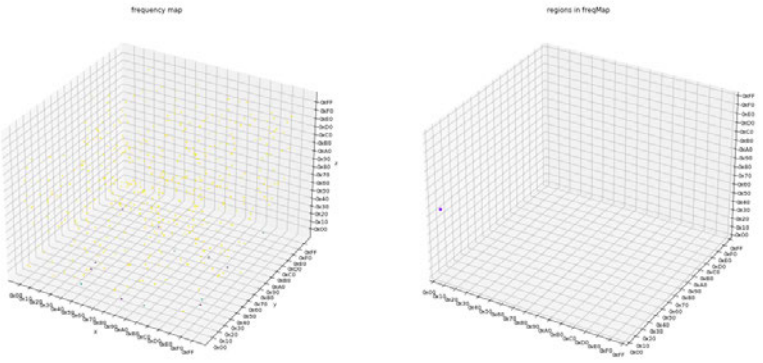
Abbildung C-11: Trigramm- und Regionendarstellung der Datei 0003-png-from-web.png [67]



(d) Sektor 4



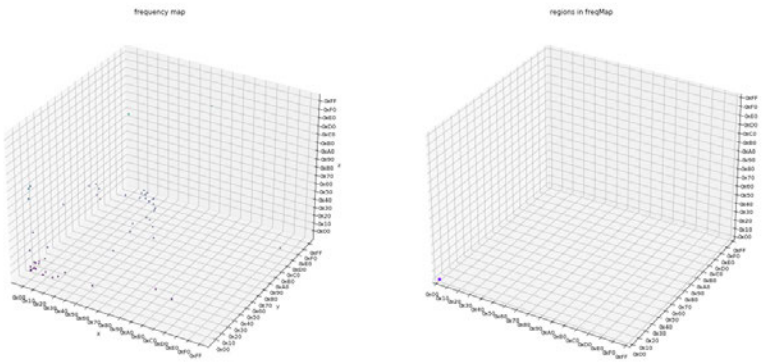
(e) Sektor 5



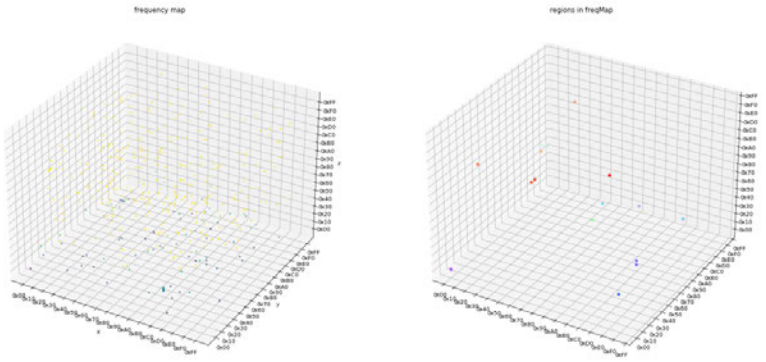
(f) Sektor 6

Abbildung C-11: Trigramm- und Regionendarstellung der Datei 0003-png-from-web.png [67], fortgesetzt

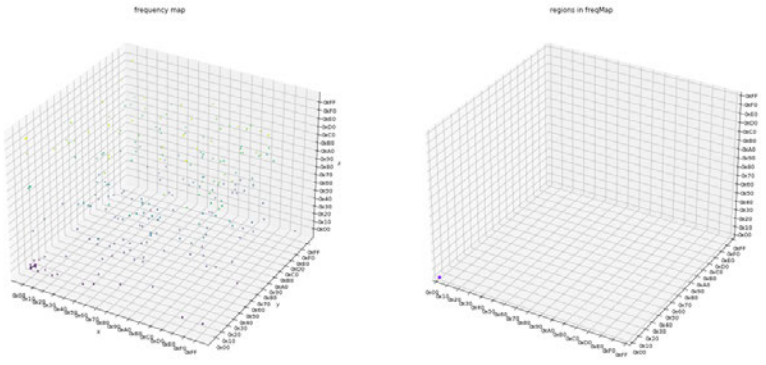
Datei 0003-pptx.pptx



(a) Sektor 1

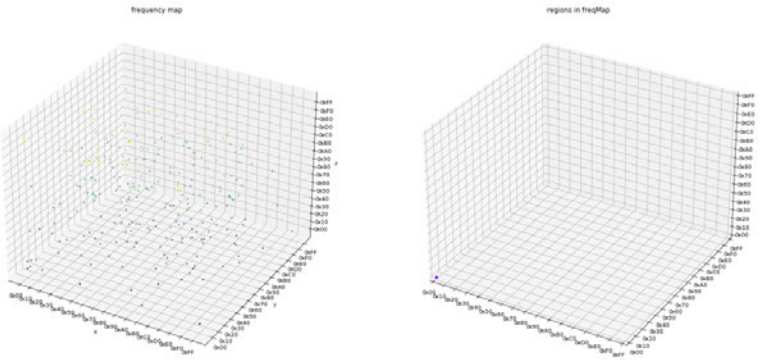


(b) Sektor 2

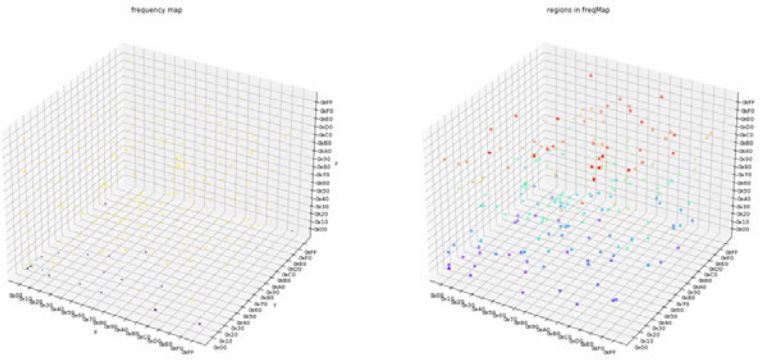


(c) Sektor 3

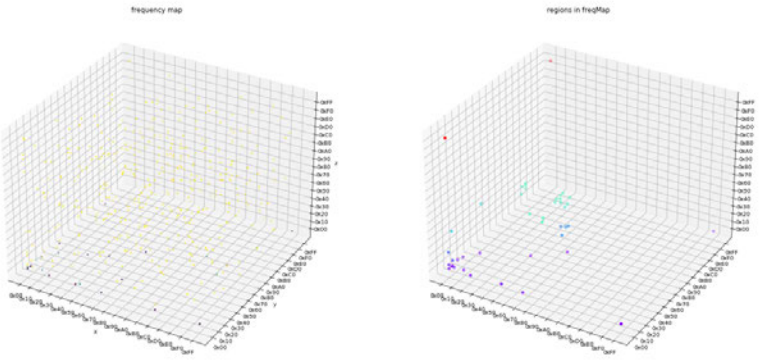
Abbildung C-12: Trigramm- und Regionendarstellung der Datei 0003-pptx.pptx [67]



(d) Sektor 4



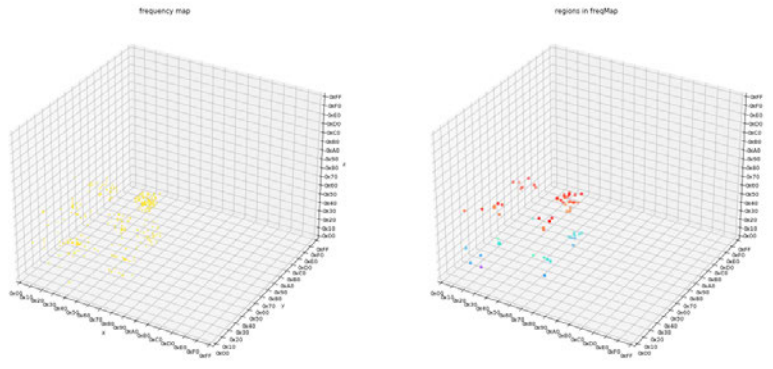
(e) Sektor 5



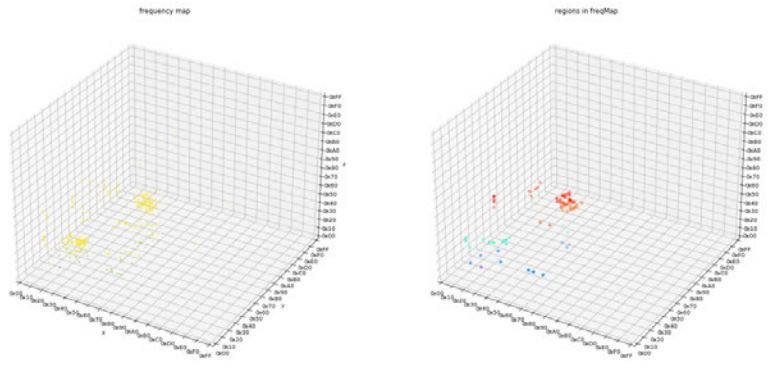
(f) Sektor 6

Abbildung C-12: Trigramm- und Regionendarstellung der Datei 0003-pptx.pptx [67], fortgesetzt

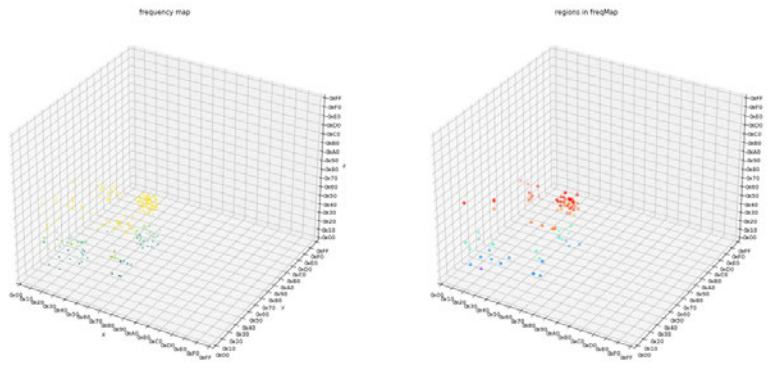
Datei 0003-svg-from-web.svg



(a) Sektor 1

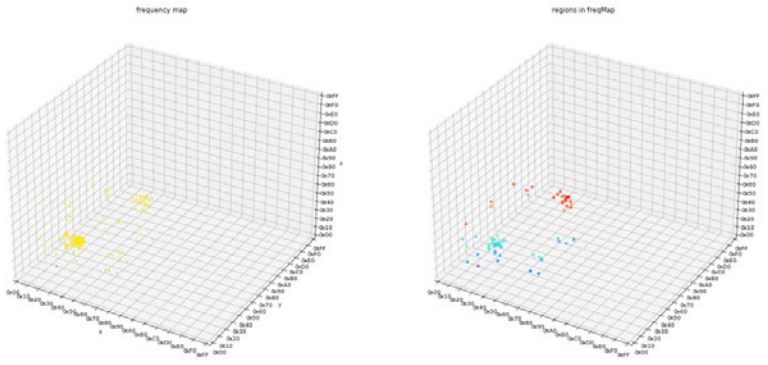


(b) Sektor 2

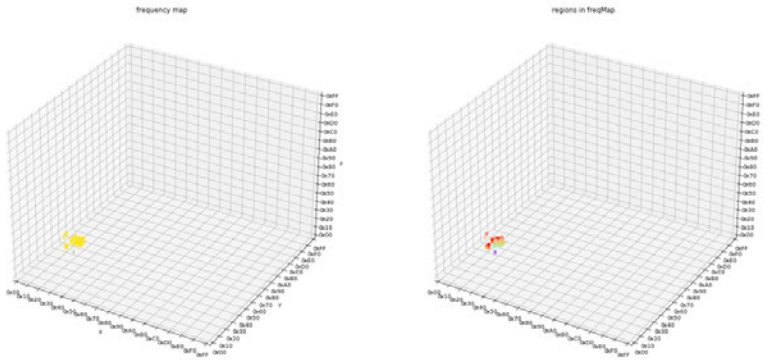


(c) Sektor 3

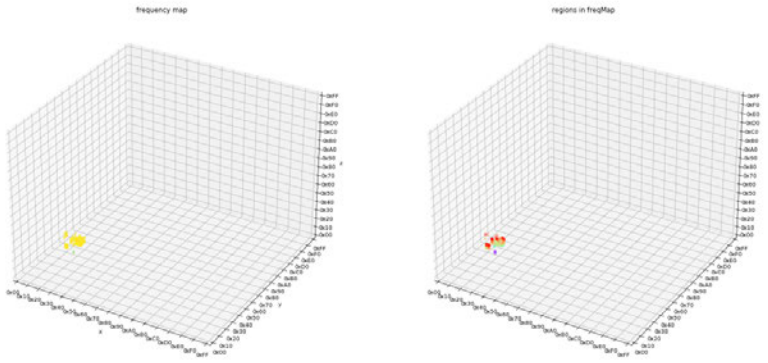
Abbildung C-13: Trigramm- und Regionendarstellung der Datei 0003-svg-from-web.svg [67]



(d) Sektor 4



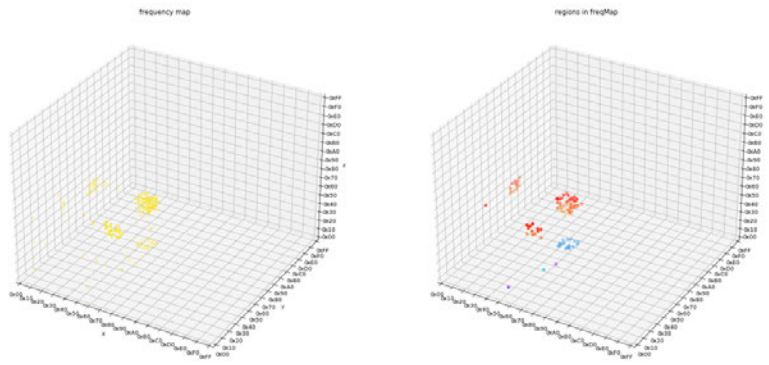
(e) Sektor 5



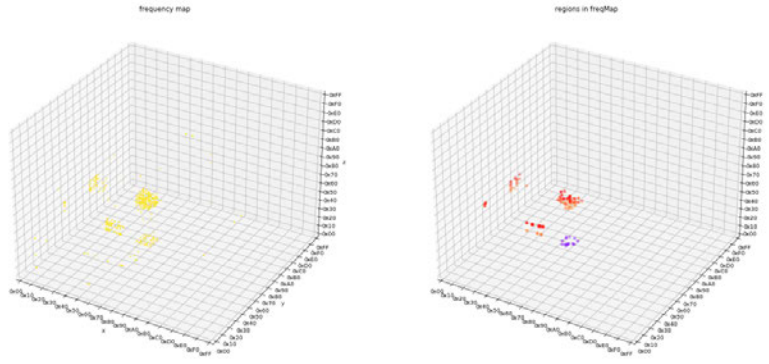
(f) Sektor 6

Abbildung C-13: Trigramm- und Regionendarstellung der Datei 0003-svg-from-web.svg [67], fortgesetzt

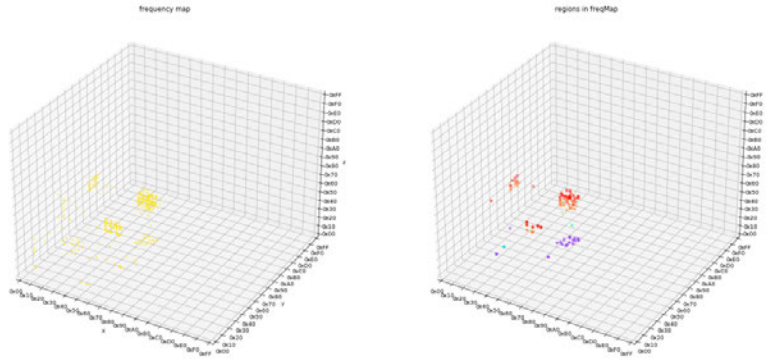
Datei 0003-txt.txt



(a) Sektor 1

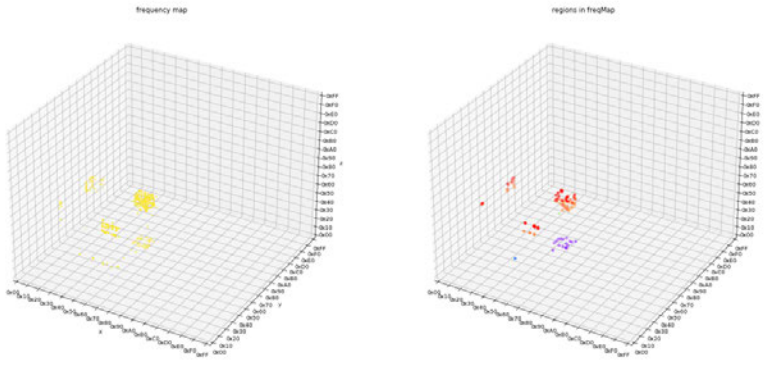


(b) Sektor 2

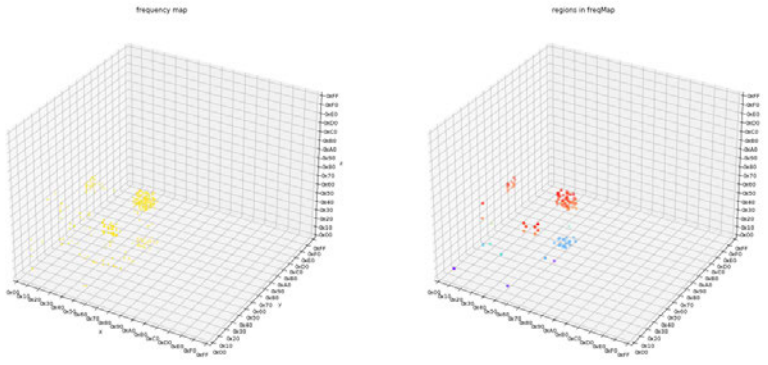


(c) Sektor 3

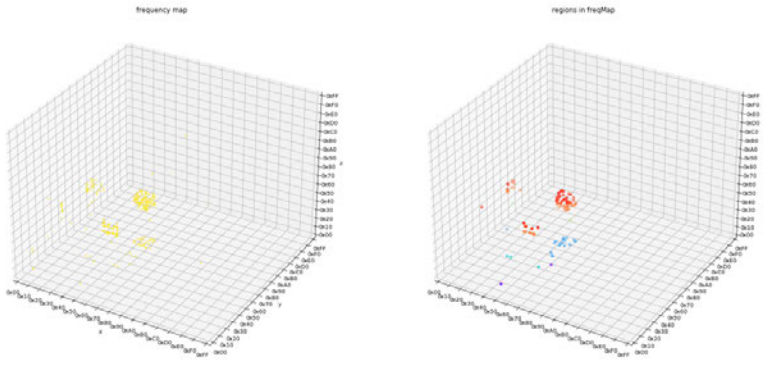
Abbildung C-14: Trigramm- und Regionendarstellung der Datei 0003-txt.txt [67]



(d) Sektor 4



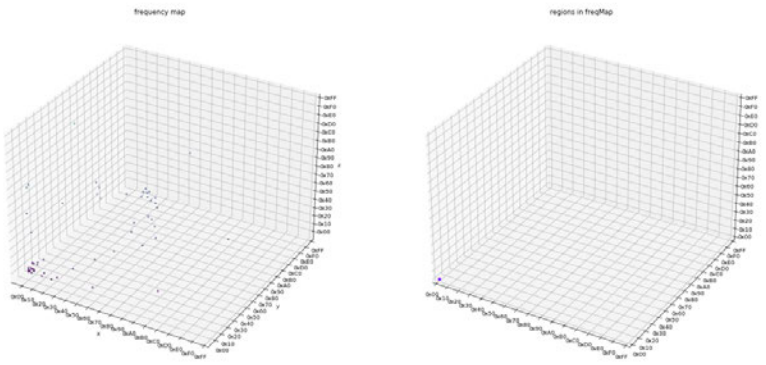
(e) Sektor 5



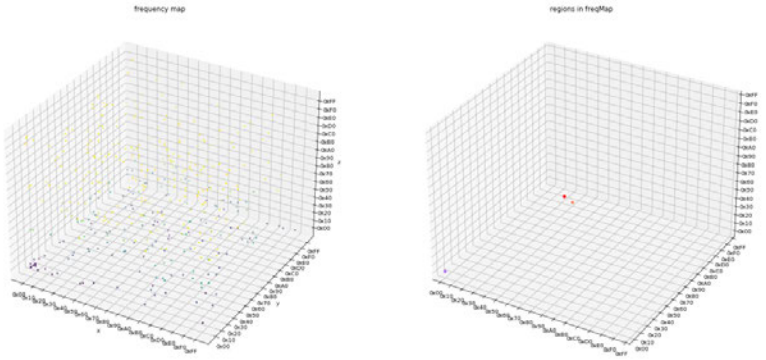
(f) Sektor 6

Abbildung C-14: Trigramm- und Regionendarstellung der Datei 0003-txt.txt [67], fortgesetzt

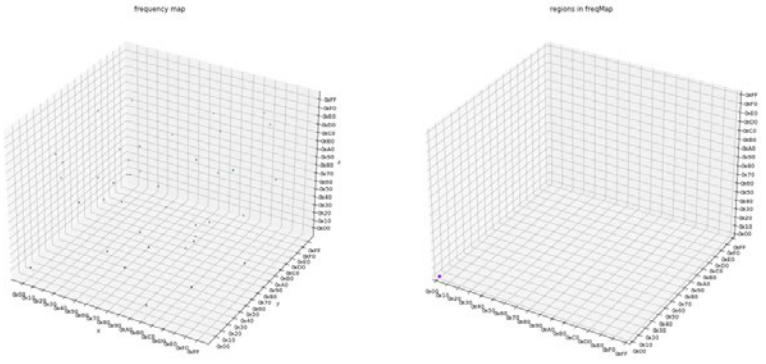
Datei 0003-xlsx.xlsx



(a) Sektor 1

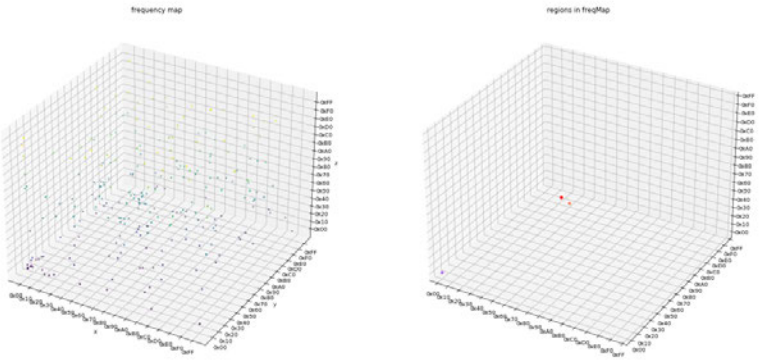


(b) Sektor 2

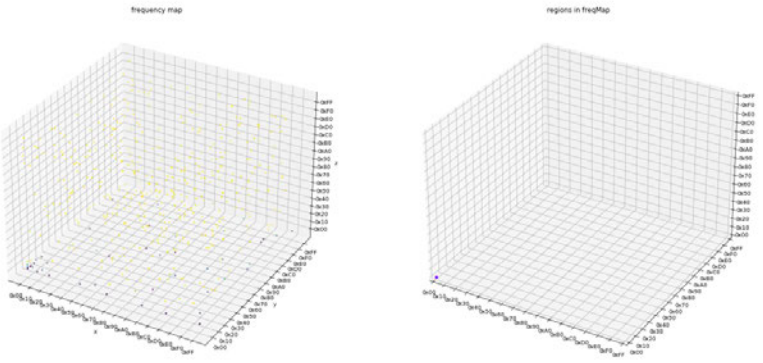


(c) Sektor 3

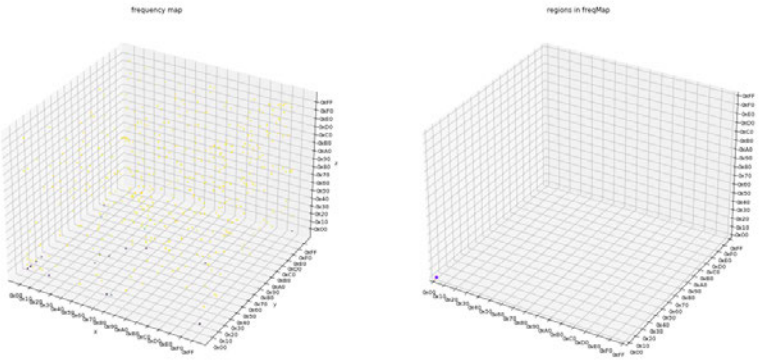
Abbildung C-15: Trigramm- und Regionendarstellung der Datei 0003-xlsx.xlsx [67]



(d) Sektor 4



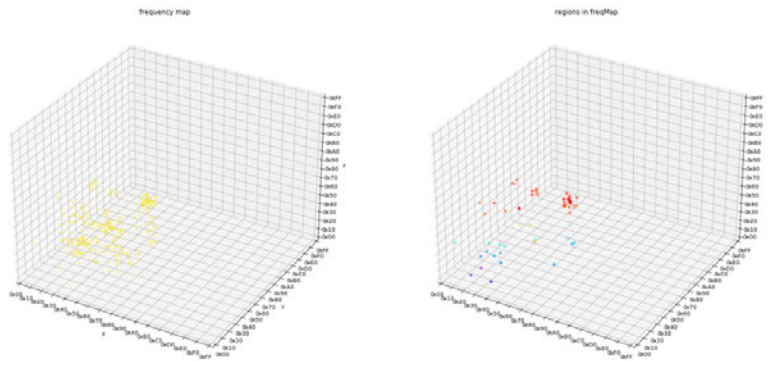
(e) Sektor 5



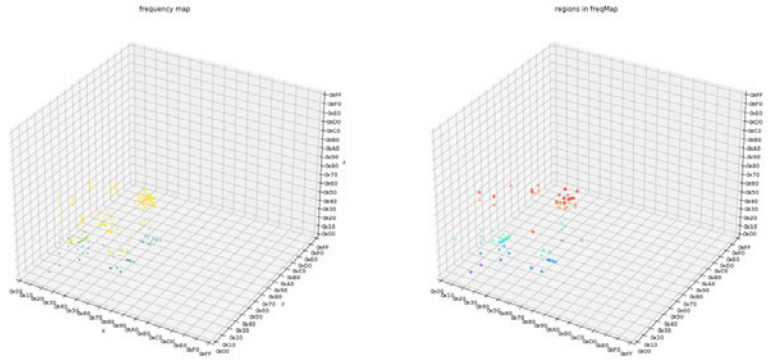
(f) Sektor 6

Abbildung C-15: Trigramm- und Regionendarstellung der Datei 0003-xlsx.xlsx [67], fortgesetzt

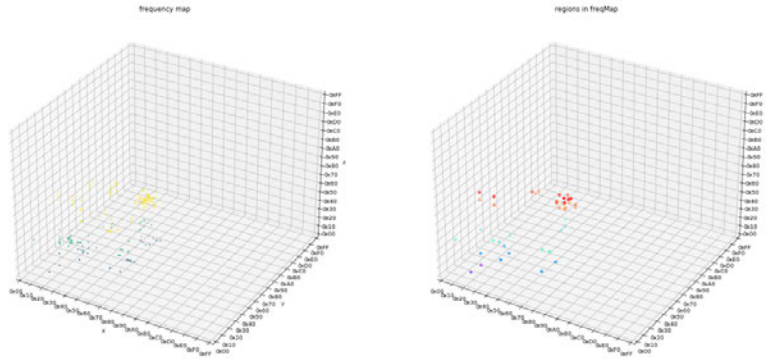
Datei 0003-xml.xml



(a) Sektor 1

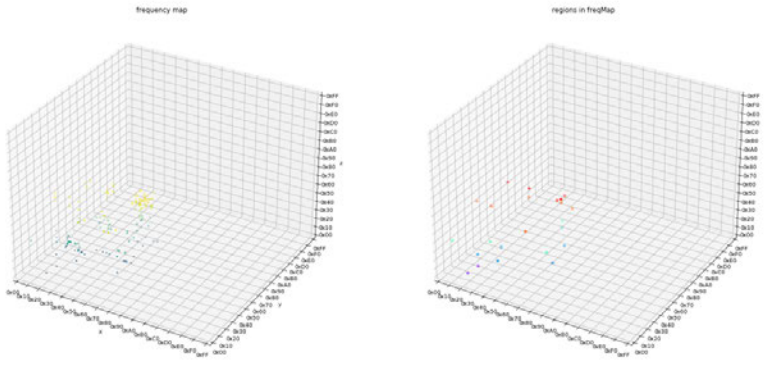


(b) Sektor 2

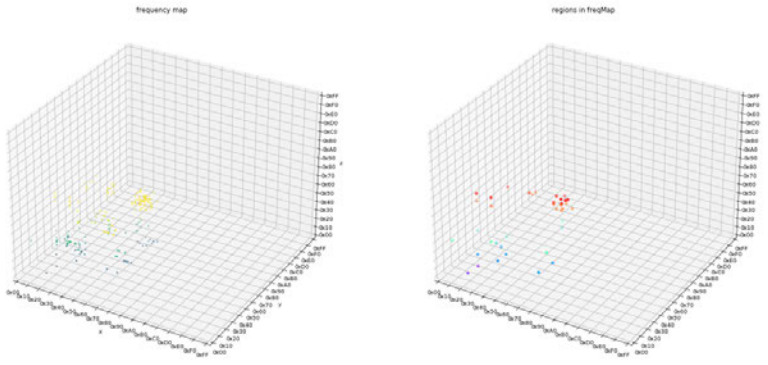


(c) Sektor 3

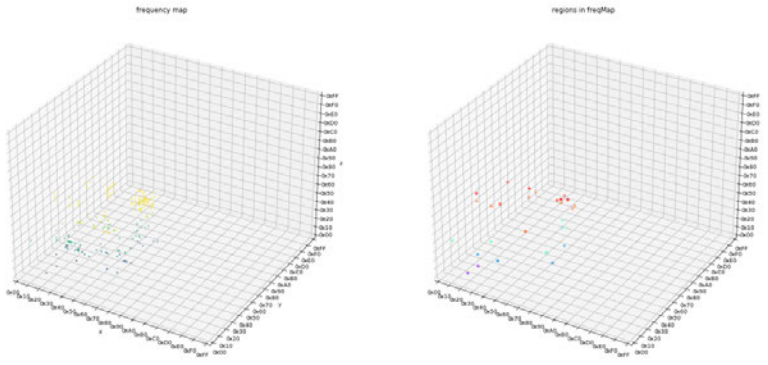
Abbildung C-16: Trigramm- und Regionendarstellung der Datei 0003-xml.xml [67]



(d) Sektor 4



(e) Sektor 5



(f) Sektor 6

Abbildung C-16: Trigramm- und Regionendarstellung der Datei 0003-xml.xml [67], fortgesetzt

Anhang D: Konfusionsmatrizen

Auf der folgenden Doppelseite sind die in Abschnitt 5.4.3 besprochenen Konfusionsmatrizen eingefügt. Darin lässt sich ablesen, wie die Sektoren einer bestimmten Klasse klassifiziert wurden. Auch umgekehrt ist erkenntlich, um was für Dateitypen es sich bei den einem Dateityp zugeordneten Sektoren tatsächlich handelt. Dabei sind zeilenweise die tatsächlichen Klassenzugehörigkeiten angegeben und spaltenweise die vom Klassifikator erkannten Dateitypen. Für einen besseren Überblick wurden in der Zelle links oben die Markierungen cc und pc eingefügt. Dabei steht cc für correct class und das pc für predicted class.

Tabelle D-1: Konfusionsmatrix für die Digrammanalyse ohne vorherige Sortierung

cc \ pc	apk	bmp	docx	elf	epub	exe	gif	jpg	mp4	pdf	png	pptx	svg	txt	xlsx	xml
apk	589	0	1	13	21	4	9	14	141	246	11	5	0	0	0	0
bmp	112	1492	58	0	10	0	60	25	56	113	129	18	0	0	0	0
docx	91	1	230	13	27	3	3	69	59	178	4	63	0	0	1	0
elf	123	4	241	124	5	3	28	0	212	10	31	14	0	0	0	0
epub	36	0	0	0	126	0	0	30	88	170	3	1	0	0	0	0
exe	72	6	31	2	3	181	68	4	43	6	0	3	0	0	0	0
gif	0	0	0	0	5	0	356	6	78	26	4	0	0	0	0	0
jpg	11	0	0	0	5	0	0	381	9	22	2	1	0	0	0	0
mp4	4	0	2	2	1	1	0	0	1722	6	0	1	0	0	0	0
pdf	79	0	2	0	23	7	15	25	193	1082	14	6	0	23	0	0
png	12	0	0	0	4	0	0	1	19	63	960	1	0	0	0	0
pptx	27	0	5	0	17	0	2	63	82	95	10	150	0	0	0	0
svg	2	1	0	0	0	0	0	0	0	1	0	0	454	0	0	0
txt	33	0	0	0	0	0	1	0	0	3	0	0	0	382	0	2
xlsx	22	0	0	0	2	0	0	0	40	65	6	3	0	0	335	0
xml	0	2	1	0	0	0	0	0	0	7	0	0	0	0	0	339

Tabelle D-2: Konfusionsmatrix für die Digrammanalyse mit vorheriger Sortierung entlang der Hilbertkurve

cc \ pc	apk	bmp	docx	elf	epub	exe	gif	jpg	mp4	pdf	png	pptx	svg	txt	xlsx	xml
apk	455	0	7	6	34	1	6	26	245	160	97	13	0	0	3	1
bmp	52	1718	168	0	2	0	51	2	32	13	32	2	0	0	1	0
docx	106	8	253	1	19	2	2	25	114	123	66	22	1	0	0	0
elf	4	3	335	95	0	1	45	0	300	1	0	8	0	0	0	3
epub	66	0	0	4	87	0	0	23	114	100	47	12	0	0	1	0
exe	39	7	64	5	2	161	67	1	55	7	0	4	0	0	1	6
gif	0	0	0	0	1	0	322	3	121	17	11	0	0	0	0	0
jpg	40	0	1	0	20	0	1	160	108	63	30	8	0	0	0	0
mp4	9	0	4	2	3	0	0	1	1702	5	11	2	0	0	0	0
pdf	117	0	2	0	30	0	16	52	324	755	152	6	4	1	1	9
png	25	0	0	0	4	0	7	2	44	30	947	1	0	0	0	0
pptx	79	1	6	0	11	0	5	28	144	52	59	65	0	0	1	0
svg	2	17	0	0	0	0	0	0	0	5	0	0	434	0	0	0
txt	37	2	0	0	0	1	0	0	2	15	0	0	3	313	0	48
xlsx	24	0	1	0	5	0	0	9	75	35	17	4	0	0	303	0
xml	2	3	0	0	0	0	0	0	0	20	0	0	2	0	0	322

Literaturverzeichnis

[1] Wikipedia, *Office Open XML*, en, Page Version ID: 1177350506, Sep. 2023. Adresse: https://en.wikipedia.org/w/index.php?title=Office_Open_XML&oldid=1177350506 (besucht am 11. 10. 2023).

[2] A. Brandt, *Visualization tools may cut through the security logjam*, en, Aug. 2008. Adresse: https://www.infoworld.com/article/2651999/visualization-tools-may-cut-through-the-security-logjam.html (besucht am 10. 07. 2023).

[3] Adrian Crenshaw, *4 2 1 Christopher Domas The future of RE Dynamic Binary Visualization*, en, Okt. 2012. Adresse: https://www.youtube.com/watch?v=4bM3Gut1hIk (besucht am 21. 04. 2023).

[4] Battelle Insider, *Battelle Publishes Open Source Binary Visualization Tool*, en, Juli 2020. Adresse: https://inside.battelle.org/blog-details/battelle-publishes-open-source-binary-visualization-tool (besucht am 21. 04. 2023).

[5] L. Fahrmeir, C. Heumann, R. Künstler, I. Pigeot und G. Tutz, *Statistik* (Springer-Lehrbuch), de. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, ISBN: 978-3-662-50371-3 978-3-662-50372-0. DOI: 10.1007/978-3-662-50372-0. Adresse: http://link.springer.com/10.1007/978-3-662-50372-0 (besucht am 19. 07. 2023).

[6] P. Planing, „Kennwerte“, de, in *Statistik Grundlagen*, Sep. 2022. Adresse: https://statistikgrundlagen.de/ebook/chapter/kennwerte/ (besucht am 19. 07. 2023).

[7] X. Wang, X. Zhang, M. Gao, Y. Tian, C. Wang und H. H.-C. Iu, „A Color Image Encryption Algorithm Based on Hash Table, Hilbert Curve and Hyper-Chaotic Synchronization“, en, *Mathematics*, Jg. 11, Nr. 3, S. 567, Jan. 2023, ISSN: 2227-7390. DOI: 10.3390/math11030567. Adresse: https://www.mdpi.com/2227-7390/11/3/567 (besucht am 03. 08. 2023).

[8] S. Fröhlich, *Hilbertkurve*, de. Adresse: http://www.geometrie-und-logik.de/galerie/flaechenf%C3%BCllend/hilbertkurve/ (besucht am 15. 09. 2023).

[9] J.-M. Wierum, „Anwendung diskreter raumfüllender Kurven: Graphpartitionierung und Kontaktsuche in der Finite-Elemente-Simulation“, de, Diss., Universität Paderborn, Fakultät für Elektrotechnik, Informatik und Mathematik, Paderborn, Okt. 2003. Adresse: https://d-nb.info/971574472/34 (besucht am 19. 09. 2023).

[10] P. Oberhofer, *Anwendung raumfüllender Kurven und Parallelisierung*, de, Juni 2003. Adresse: https://www5.in.tum.de/lehre/seminare/oktal/SS03/ausarbeitungen/oberhofer.pdf (besucht am 21. 04. 2023).

[11] R. Adams, *numpy-hilbert-curve*, original-date: 2020-11-04T19:34:15Z, Nov. 2020. Adresse: https://github.com/PrincetonLIPS/numpy-hilbert-curve (besucht am 03. 08. 2023).

[12] A. Cortesi, *Portrait of the Hilbert curve*, en, Jan. 2010. Adresse: https://corte.si/posts/code/hilbert/portrait/ (besucht am 27. 06. 2023).

[13] A. Lesne, „Shannon entropy: a rigorous notion at the crossroads between probability, information theory, dynamical systems and statistical physics“, en, *Mathematical Structures in Computer Science*, Jg. 24, Nr. 3, Juni 2014, ISSN: 0960-1295, 1469-8072. DOI: 10.1017/S0960129512000783. Adresse: https://www.cambridge.org/core/product/identifier/S0960129512000783/type/journal_article (besucht am 24. 07. 2023).

[14] J. Brownlee, *A Gentle Introduction to Information Entropy*, en, Okt. 2019. Adresse: https://machinelearningmastery.com/what-is-information-entropy/ (besucht am 15. 08. 2023).

[15] A. Cortesi, *Visualizing entropy in binary files*, en, Jan. 2012. Adresse: https://corte.si/posts/visualisation/entropy/index.html (besucht am 03. 07. 2023).

[16] J. Brownlee, *How to Develop a Naive Bayes Classifier from Scratch in Python*, en, Jan. 2020. Adresse: https://machinelearningmastery.com/classification-as-conditional-probability-and-the-naive-bayes-algorithm/ (besucht am 23. 08. 2023).

[17] A. A. Awan und A. Navlani, *Naive Bayes Classifier Tutorial: with Python Scikit-learn*, en, März 2023. Adresse: https://www.datacamp.com/tutorial/naive-bayes-scikit-learn (besucht am 27. 09. 2023).

[18] J. Aßfalg u. a., *Kapitel 3: Klassifikation*, de, Skript zur Vorlesung Knowledge Discovery in Databases im Wintersemester 2007/2008, 2007. Adresse: https://www.dbs.ifi.lmu.de/Lehre/KDD/WS0708/skript/kdd-3-klassifikation.pdf (besucht am 11. 09. 2023).

[19] J. Brownlee, *Supervised and Unsupervised Machine Learning Algorithms*, en, Aug. 2020. Adresse: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ (besucht am 28. 09. 2023).

[20] T. Kühn, „Implementierung und Evaluation ergänzender Korrekturmethoden für statistische Lernverfahren bei unbalancierten Klassifikationsproblemen“, de, Okt. 2014, Herausgeber: Institut für Statistik, Ludwig Maximilians Universität München. DOI: 10.5282/UBM/EPUB.25525. Adresse: https://epub.ub.uni-muenchen.de/id/eprint/25525 (besucht am 11. 09. 2023).

[21] webmaster@saracus.com, *Performance-Metriken für Klassifikationsprobleme*, de, Nov. 2018. Adresse: https://saracus.com/synvert-saracus-blog/performance-metriken-klassifikation-2-2/ (besucht am 15. 09. 2023).

[22] F. Pedregosa u. a., *3.3. Metrics and scoring: quantifying the quality of predictions*, en. Adresse: https://scikit-learn/stable/modules/model_evaluation.html (besucht am 11. 09. 2023).

[23] T. Montanari, *Color Model vs Color Space: A Beginner's Guide [1/2]*, en, Apr. 2023. Adresse: https://www.tobiamontanari.com/color-model-vs-color-space-a-beginners-guide/ (besucht am 28. 07. 2023).

[24] S. Süsstrunk, R. Buckley und S. Swen, „Standard RGB Color Spaces“, en, in *Proc. IS&T/SID 7th Color Imaging Conference*, Bd. 7, 1999, S. 127–134. Adresse: http://infoscience.epfl.ch/record/34089.

[25] V. Kalist, P. Ganesan, B. Sathish, J. M. M. Jenitha und K. Basha.shaik, „Possiblistic-Fuzzy C-Means Clustering Approach for the Segmentation of Satellite Images in HSL Color Space“, en, *Procedia Computer Science*, Jg. 57, S. 49–56, 2015, ISSN: 18770509. DOI: 10.1016/j.procs.2015.07.364. Adresse: https://linkinghub.elsevier.com/retrieve/pii/S1877050915018931 (besucht am 28. 07. 2023).

[26] A. Horchler, *Antwort auf "Drawing 3-D RGB cube model with Matlab"*, en, Apr. 2015. Adresse: https://stackoverflow.com/a/29956129 (besucht am 10. 10. 2023).

[27] J. Y. Bai und H. E. Ren, „An Algorithm of Leaf Image Segmentation Based on Color Features“, en, *Key Engineering Materials*, Jg. 474-476, S. 846–851, Apr. 2011, ISSN: 1662-9795. DOI: 10.4028/www.scientific.net/KEM.474-476.846. Adresse: https://www.scientific.net/KEM.474-476.846 (besucht am 10. 10. 2023).

[28] T. Montanari, *HSL and HSV Explained: Which Color Model Should You Use?*, en, Mai 2023. Adresse: https://www.tobiamontanari.com/hsl-and-hsv-explained-which-color-model-should-you-use/ (besucht am 28. 07. 2023).

[29] L. Vandevenne, *Light and Color*, en, 2004. Adresse: https://lodev.org/cgtutor/color.html (besucht am 07. 08. 2023).

[30] A. Guarino, „Digital Forensics as a Big Data Challenge“, en, in *ISSE 2013 Securing Electronic Business Processes*, H. Reimer, N. Pohlmann und W. Schneider, Hrsg., Wiesbaden: Springer Fachmedien Wiesbaden, 2013, S. 197–203, ISBN: 978-3-658-03370-5 978-3-658-03371-2. DOI: 10.1007/978-3-658-03371-2_17. Adresse: http://link.springer.com/10.1007/978-3-658-03371-2_17 (besucht am 26. 09. 2023).

[31] K. Barik, A. Abirami, K. Konar und S. Das, „Research Perspective on Digital Forensic Tools and Investigation Process“, en, in *Illumination of Artificial Intelligence in Cybersecurity and Forensics*, S. Misra und C. Arumugam, Hrsg., Bd. 109, Series Title: Lecture Notes on Data Engineering and Communications Technologies, Cham: Springer International Publishing, 2022, S. 71–95, ISBN: 978-3-030-93452-1 978-3-030-93453-8. DOI: 10.1007/978-3-030-93453-8_4. Adresse: https://link.springer.com/10.1007/978-3-030-93453-8_4 (besucht am 21. 04. 2023).

[32] B. Carrier, *File System Forensic Analysis*, en. Addison Wesley Professional, März 2005, ISBN: 0-321-26817-2.

[33] A. Geschonneck, *Computer-Forensik: Computerstraftaten erkennen, ermitteln, aufklären* (iX-Edition), de, 6., aktualisierte und erweiterte Auflage. Heidelberg: dpunkt.verlag, 2014, ISBN: 978-3-86491-489-8 978-3-86490-133-1.

[34] N. M. Karie und H. S. Venter, „Taxonomy of Challenges for Digital Forensics“, en, *Journal of Forensic Sciences*, Jg. 60, Nr. 4, S. 885–893, Juli 2015, ISSN: 0022-1198, 1556-4029. DOI: 10.1111/1556-4029.12809. Adresse: https://onlinelibrary.wiley.com/doi/10.1111/1556-4029.12809 (besucht am 26. 09. 2023).

[35] „Leitfaden „IT-Forensik““, de, Nr. Version 1.0.1, Bundesamt für Sicherheit in der Informationstechnik (BSI), Hrsg., März 2011. Adresse: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/Themen/Leitfaden_IT-Forensik.pdf?__blob=publicationFile&v=1 (besucht am 15. 05. 2023).

[36] G. Conti u. a., „A Visual Study of Primitive Binary Fragment Types“, en, Juli 2010.

[37] H. Reibold, *Android Forensik*, de. Saarbrücken: Brain Media, 2016, OCLC: 1033815272, ISBN: 978-3-95444-258-4.

[38] A. Fukami, S. Sheremetov, F. Regazzoni, Z. Geradts und C. De Laat, „Experimental Evaluation of e.MMC Data Recovery“, en, *IEEE Transactions on Information Forensics and Security*, Jg. 17, S. 2074–2083, 2022, ISSN: 1556-6013, 1556-6021. DOI: 10.1109/TIFS.2022.3176187. Adresse: https://ieeexplore.ieee.org/document/9777707/ (besucht am 21. 04. 2023).

[39] M. F. Abdillah und Y. Prayudi, „Data Recovery Comparative Analysis using Open-based Forensic Tools Source on Linux“, en, *International Journal of Advanced Computer Science and Applications*, Jg. 13, Nr. 9, 2022, ISSN: 21565570, 2158107X. DOI: 10.14569/IJACSA.2022.0130975. Adresse: http://thesai.org/Publications/ViewPaper?Volume=13&Issue=9&Code=IJACSA&SerialNo=75 (besucht am 21. 04. 2023).

[40] K. Tasdelen und A. A. Süzen, „Analysing and Carving MS Word and PDF Files from RAM Images on Windows“, en, *Tehnicki vjesnik - Technical Gazette*, Jg. 29, Nr. 5, Okt. 2022, ISSN: 13303651, 18486339. DOI: 10.17559/TV-20210218122046. Adresse: https://hrcak.srce.hr/281688 (besucht am 21. 04. 2023).

[41] X-Ways AG, *X-Ways Forensics: Integrierte Software für Computerforensik*, de. Adresse: http://www.x-ways.net/forensics/index-d.html (besucht am 22. 05. 2023).

[42] G. Conti, E. Dean, M. Sinda und B. Sangster, „Visual Reverse Engineering of Binary and Data Files“, en, Sep. 2008. Adresse: http://www.gregconti.com/publications/2008_VizSEC_FileVisualization_v53_final.pdf.

[43] S. Bratus und G. Conti, „Voyage of the Reverser: A Visual Study of Binary Species“, en, *Black Hat USA 2010, Las Vegas, Nevada, USA*, Juli 2010. Adresse: https://media.blackhat.com/bh-us-10/presentations/Bratus_Conti/BlackHat-USA-2010-Bratus-Conti-Voyage-of-a-Reverser-slides.pdf.

[44] A. Cortesi, *Visualizing binaries with space-filling curves*, en, Dez. 2011. Adresse: https://corte.si/posts/visualisation/binvis/ (besucht am 27. 06. 2023).

[45] A. Cortesi, *binvis.io*, en, März 2015. Adresse: http://binvis.io (besucht am 03. 07. 2023).

[46] A. Cortesi, *binvis.io - a browser-based tool for visualising binary data*, en, März 2015. Adresse: https://corte.si/posts/binvis/announce/ (besucht am 27. 06. 2023).

[47] A. Cortesi, *binvis.io - Help*, en, März 2015. Adresse: http://binvis.io/#/view/local?modal=help (besucht am 03. 07. 2023).

[48] Statista, *Prognose zum Volumen der gespeicherten Datenmenge in Rechenzentren weltweit in den Jahren 2016 bis 2021 (in Exabyte)*, de, Feb. 2018. Adresse: https://de.statista.com/statistik/daten/studie/819487/umfrage/prognose-zum-weltweit-gespeicherten-datenvolumen-in-rechenzentren/ (besucht am 30. 06. 2023).

[49] M. Pytel, *Binary visualization explained*, en, Jan. 2017. Adresse: https://codisec.com/binary-visualization-explained/ (besucht am 21. 04. 2023).

[50] M. Pytel, *Binary data visualization*, en, Dez. 2016. Adresse: https://codisec.com/binary-data-visualization/ (besucht am 21. 04. 2023).

[51] B. Kerler, *Mobile Revelator*, Juni 2020. Adresse: https://github.com/bkerler/MR (besucht am 28. 04. 2023).

[52] B. Kerler, *Mobile Revelator 2.2.5 - Manual*, en, Feb. 2019. Adresse: https://github.com/bkerler/MR/blob/master/Manual.pdf (besucht am 11. 10. 2023).

[53] A. Singh, A. Handa, N. Kumar und S. K. Shukla, „Malware Analysis Using Image Classification Techniques“, en, in *Cyber Security in India*, S. K. Shukla und M. Agrawal, Hrsg., Bd. 4, Series Title: IITK Directions, Singapore: Springer Singapore, 2020, S. 33–38, ISBN: 9789811516757. DOI: 10.1007/978-981-15-1675-7_4. Adresse: http://link.springer.com/10.1007/978-981-15-1675-7_4 (besucht am 16. 05. 2023).

[54] A. Marwaha u. a., „Visualisation-based binary classification of android malware using vgg16“, en, *IET Software*, sfw2.12094, Jan. 2023, ISSN: 1751-8806, 1751-8814. DOI: 10.1049/sfw2.12094. Adresse: https://onlinelibrary.wiley.com/doi/10.1049/sfw2.12094 (besucht am 26. 04. 2023).

[55] Deutschlandfunk, *AI Act: Regelwerk für die künstliche Intelligenz*, de. Adresse: https://www.deutschlandfunk.de/ai-act-eu-kuenstliche-intelligenz-gefahr-regulierung-100.html (besucht am 11. 10. 2023).

[56] A. A. Solanke, „Explainable digital forensics AI: Towards mitigating distrust in AI-based digital forensics analysis using interpretable models“, en, *Forensic Science International: Digital Investigation*, Jg. 42, S. 301 403, Juli 2022, ISSN: 26662817. DOI: 10.1016/j.fsidi.2022.301403. Adresse: https://linkinghub.elsevier.com/retrieve/pii/S2666281722000841 (besucht am 21. 04. 2023).

[57] Europäisches Parlament, *2020/2016(INI) - 06/10/2021 - Artificial intelligence in criminal law and its use by the police and judicial authorities in criminal matters*, en, Okt. 2021. Adresse: https://oeil.secure.europarl.europa.eu/oeil/popups/summary.do?id=1678184&t=e&l=en (besucht am 11. 10. 2023).

[58] W. McKinney, *Python for data analysis: data wrangling with Pandas, NumPy, and Jupyter*, en, Third edition. Sebastopol, CA: O'Reilly, 2022, ISBN: 978-1-09-810403-0.

[59] M. A. Aaberge, *Everything You Need to Get Started With Python Programming*, en, Sep. 2020. Adresse: https://towardsdatascience.com/everything-you-need-to-get-started-with-python-programming-4a37a46e427b (besucht am 11. 10. 2023).

[60] *FILExt – Alles über Dateiarten und wie man Dateien online öffnen und ansehen kann*, de, 2023. Adresse: https://filext.com/de/ (besucht am 25. 07. 2023).

[61] Python, *Python 3.11.6 Documentation*, en, Okt. 2023. Adresse: https://docs.python.org/3.11/ (besucht am 13. 10. 2023).

[62] The Matplotlib development team, *Matplotlib documentation — Matplotlib 3.8.0 documentation*, en, 2023. Adresse: https://matplotlib.org/stable/index.html (besucht am 13. 10. 2023).

[63] Numpy Developers, *NumPy Reference — NumPy v1.24 Manual*, Dez. 2022. Adresse: https://numpy.org/doc/1.24/reference/index.html#reference (besucht am 13. 10. 2023).

[64] J. A. Clark, *Image Module*, en, 2023. Adresse: https://pillow.readthedocs.io/en/stable/reference/reference/Image.html (besucht am 13. 10. 2023).

[65] F. Pedregosa u. a., *User Guide*, en, 2023. Adresse: https://scikit-learn.org/stable/user_guide.html (besucht am 04. 07. 2023).

[66] The SciPy community, *Multidimensional image processing (scipy.ndimage) — SciPy v1.10.1 Manual*, en, 2023. Adresse: https://docs.scipy.org/doc/scipy/reference/ndimage.html#module-scipy.ndimage (besucht am 02. 06. 2023).

[67] S. R. Davies, R. Macfarlane und W. J. Buchanan, „NapierOne: A modern mixed file data set alternative to Govdocs1“, en, *Forensic Science International: Digital Investigation*, Jg. 40, S. 301 330, März 2022, ISSN: 26662817. DOI: 10.1016/j.fsidi.2021.301330. Adresse: https://linkinghub.elsevier.com/retrieve/pii/S2666281721002560 (besucht am 04. 05. 2023).

[68] K. Stratis, *How to Use Generators and yield in Python – Real Python*, en. Adresse: https://realpython.com/introduction-to-python-generators/ (besucht am 01. 06. 2023).

[69] IONOS, *Nibble – was ist das?*, de, Nov. 2021. Adresse: https://www.ionos.de/digitalguide/websites/web-entwicklung/was-ist-ein-nibble/ (besucht am 28. 08. 2023).

[70] The SciPy community, *scipy.ndimage.label — SciPy v1.11.3 Manual*, en, Sep. 2023. Adresse: https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.label.html#scipy.ndimage.label (besucht am 03. 10. 2023).

[71] larsaars, *Antwort auf "How to flatten 2d matrix (n x n) into 1d array in order of Hilbert Curve?"*, en, Jan. 2022. Adresse: https://stackoverflow.com/a/70856068 (besucht am 26. 07. 2023).

[72] A. A. Solanke und M. A. Biasiotti, „Digital Forensics AI: Evaluating, Standardizing and Optimizing Digital Evidence Mining Techniques“, en, *KI - Künstliche Intelligenz*, Jg. 36, Nr. 2, S. 143–161, Sep. 2022, ISSN: 0933-1875, 1610-1987. DOI: 10.1007/s13218-022-00763-9. Adresse: https://link.springer.com/10.1007/s13218-022-00763-9 (besucht am 21. 04. 2023).

Eidesstattliche Erklärung

Hiermit versichere ich – Susanna Juliane Beck – an Eides statt, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt oder anderweitig veröffentlicht.

Berlin, 17.10.2023

Ort, Datum

Susanna Juliane Beck

Nutzungs- und Verwertungsrechte

Ich übertrage zusätzliche Nutzungs- und Verwertungsrechte für die vorliegende Arbeit und allen damit in Zusammenhang stehenden Daten auf Grundlage der *Creative Commons Lizenz „CC0“* an alle genannten Betreuer dieser Arbeit.

Berlin, 17.10.2023

Ort, Datum

Susanna Juliane Beck