IT-Sicherheit mit Assoziativmatrizen

H.-J. Bentz (*), J. Braun (&), A. Dierks (*), U. Dobbratz (*), O. Festerling (*), M. Glockemann (*), G. Palm (§), M. Miyamoto (#), D. Romberger (#), F. Rosenschein (*), J. Weber (#)
[* imbit.net, # Hochschule Hannover, & Ludwig–Maximilians–Universität München, § Universität Ulm]

Der Einsatz einer besonderen Art von Assoziativmatrizen erlaubt ein informationstechnisch sicheres Ablegen und Übertragen von Daten.[1] Die durchzuführenden Matrixoperationen werden durch elektronische Schaltungen unterstützt, die in einem eigens dafür entwickelten Baustein untergebracht sind. Für unterschiedliche Anwendungsfelder entstanden Gerätemodelle, die hinsichtlich der Matrixgröße skalierbar und bezüglich der Sicherheitserfordernisse erweiterbar sind.

Inhalt

Einordnung

Gegenstand unserer Untersuchungen sind die Eigenschaften und Anwendungsgebiete von Matrizen, deren Einträge nur aus Nullen und Einsen bestehen. Die Verteilung dieser Nullen und Einsen kann durch Lernregeln bestimmt werden, wie sie bei Palm [1982], Steinbuch [1965] oder Bentz and Dierks [2013] beschrieben sind.[2] Diese Matrizen dienen der Mustererkennung, -vervollständigung oder -extraktion, dem Assoziativen Rechnen oder der Assoziativen Programmierung durch ihre Fähigkeit, mit Fragen mittels der von ihnen gelernten Frage–Antwort–Paare tolerant Antworten zu assoziieren.[3] Daher nennt man diese Matrizen *Assoziativmatrizen.* Ihr Zusammenwirken in komplexeren Strukturen führt zu robusten, frei programmierbaren *Assoziativmaschinen.*[4]

Im Folgenden geht es jedoch um Eigenschaften von Assoziativmatrizen, bei denen das Eintragen der Nullen und Einsen **nicht** per Lernregel und durch vorgegebene Frage–Antwort–Paare, sondern **zufällig** erfolgt, und deren Zeilen- und Spaltenanzahl n gleich groß ist, also um $n \times n$–Matrizen. Das liefert einen informationstechnischen Nutzen, der hier unser zentrales Anliegen ist.

Der ursprüngliche Ansatz, Assoziativmatrizen zur Modellierung synaptischer Verbindungen von waagerecht und senkrecht angeordneten Neuronen zu nutzen, wird durch die herkömmlichen Lernregeln deutlich.[5] Mit dem im Folgenden vorgestellten *Lernen durch den Zufall*, verlassen wir das Anwendungsgebiet der künstlichen neuronalen Netze, in welchem es um das Trainieren und Erkennen von Mustern geht.[6]

Die nachstehende Abb. 1 veranschaulicht, wie eine Matrix zur Beschreibung der Lage von Verbindungsstellen eingesetzt werden kann. Beide Darstellungen geben das gleiche Verbindungsmuster wieder. Die linke Darstellung ist in einem schaltungstechnischen, die rechte in einem mathematischen Umfeld nützlich.

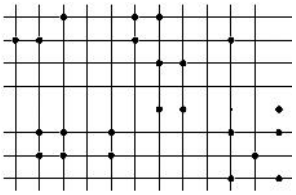


Abb. 1: Die Verbindungen zwischen waage- und senkrecht verlaufenden Neuronen sind als Punkte markiert.

$$\begin{pmatrix} 0&0&1&0&0&1&1&0&0&0&0&0 \\ 1&1&0&0&0&1&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&1&0&0&0&1 \\ 0&1&1&0&1&0&0&0&0&1&0&1 \\ 0&1&1&0&1&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&1&0&1 \end{pmatrix}$$

Abb. 2: Die Orte, an denen Zeilen und Spalten verbunden sind, werden in der Matrix als 1 notiert, die anderen als 0.

Nunmehr lassen wir den Zufall oder einen Zufallsgenerator die Verbindungsstellen zwischen Zeilen und Spalten bestimmen. Füllt man alle Spalten einer $n \times n$–Matrix, in der vorher nur Nullen stehen (*Nullmatrix*), zufällig mit Einsen, bis eine vorgegebene Anzahl p an Einsen erreicht ist, mit $p \leq n$, nennt man diese Matrix *spaltenstarr* (siehe Abb. 3). Füllt man alle Zeilen einer $n \times n$–Matrix zufällig mit p Einsen, nennt man sie *zeilenstarr.* Werden in die Spalten **und** Zeilen einer Matrix zufällig jeweils eine feste Anzahl Einsen eingetragen, heißt sie *doppeltstarr* (siehe Abb. 4). Verteilt man eine feste Anzahl p von Einsen zufällig in einer $n \times n$–Matrix, wird sie *p–dicht* genannt, mit $p \leq n^2$. Man beachte, dass es einen Unterschied gibt zu Matrizen, in denen auf p zufälligen Positionen Einsen verteilt werden, denn das kann zu weniger als p Einsen in der Matrix führen.

$$\begin{pmatrix} 0&0&1&0&1&1&0&0 \\ 1&1&1&0&0&1&1&0 \\ 1&0&1&1&0&0&1&1 \\ 0&1&1&0&0&1&0&0 \\ 0&0&0&1&0&0&1&1 \\ 1&1&0&1&1&1&0&1 \\ 0&1&0&1&1&0&1&1 \\ 1&0&0&0&1&0&0&0 \end{pmatrix}$$

Abb. 3: Eine spaltenstarre 8×8–Matrix mit $p = \frac{1}{2}n$

$$\begin{pmatrix} 1&1&0&1&0&0&1&0 \\ 0&0&0&0&1&1&1&1 \\ 0&1&1&1&0&1&0&0 \\ 1&1&1&0&0&0&1&0 \\ 0&0&1&1&0&0&1&1 \\ 1&0&0&0&1&1&0&1 \\ 1&1&0&0&1&1&0&0 \\ 0&0&1&1&1&0&0&1 \end{pmatrix}$$

Abb. 4: Eine doppeltstarre 8×8–Matrix mit $p = \frac{1}{2}n$

Wir konzentrieren uns im Folgenden auf spaltenstarre $n \times n$–Matrizen.

Das *Abfragen* einer Matrix geschieht durch die übliche Multiplikation eines n–Tupels aus Nullen und Einsen mit der Matrix.[7] Dieses n–Tupel wird *Frage* genannt. Durch die Multiplikation entsteht ein n–Tupel mit Werten, die kleiner oder gleich n sind. Dieses Zwischenergebnis wird mit Hilfe eines von vielen möglichen Verfahren binarisiert, indem beispielsweise alle Werte, die geradzahlig sind, auf eine Null abgebildet werden, die anderen auf eine Eins. Durch diesen Vorgang entsteht eine *Antwort*, ein weiteres n–Tupel, das ebenfalls nur aus Nullen und Einsen besteht.

Sei beispielsweise das 8–Tupel $f = (1\,0\,1\,1\,0\,1\,0\,0)$ die Frage, dann ergäbe sich mit der Matrix aus der Abb. 3 folgende Rechnung.

$$\begin{pmatrix} 1\,0\,1\,1\,0\,1\,0\,0 \end{pmatrix} \cdot \begin{pmatrix} 0\,0\,1\,0\,1\,1\,0\,0 \\ 1\,1\,1\,0\,0\,1\,1\,0 \\ 1\,0\,1\,1\,0\,0\,1\,1 \\ 0\,1\,1\,0\,0\,1\,0\,0 \\ 0\,0\,0\,1\,0\,0\,1\,1 \\ 1\,1\,0\,1\,1\,1\,0\,1 \\ 0\,1\,0\,1\,1\,0\,1\,1 \\ 1\,0\,0\,0\,1\,0\,0\,0 \end{pmatrix}$$

$$= \begin{pmatrix} 2\,2\,3\,2\,2\,3\,1\,2 \end{pmatrix}$$

Abb. 5: Multiplikation einer Frage mit einer 8×8–Matrix

Wenden wir auf das Zwischenergebnis $(2\,2\,3\,2\,2\,3\,1\,2)$ nun die Regel an, alle geradzahligen Werte auf Null und die anderen auf Eins abzubilden, erhalten wir eine Antwort a auf die gestellte Frage f. Die Antwort hat das gleiche Format wie die Frage.

$$\begin{pmatrix} 2\,2\,3\,2\,2\,3\,1\,2 \end{pmatrix} \xrightarrow{\text{LSB}} \begin{pmatrix} 0\,0\,1\,0\,0\,1\,1\,0 \end{pmatrix}$$

Das Verfahren, welches hier alle geradzahligen Werte auf Null und die anderen auf Eins abbildet, heißt *LSB*–Verfahren[8] und wird mit $\xrightarrow{\text{LSB}}$ notiert. Ein weiteres Verfahren zur Binarisierung wäre folglich das umgekehrte Verfahren $\overline{LSB}$, mit dem man alle geradzahligen Werte auf Eins und die anderen auf Null abbildet.

$$\begin{pmatrix} 2\,2\,3\,2\,2\,3\,1\,2 \end{pmatrix} \xrightarrow{\overline{\text{LSB}}} \begin{pmatrix} 1\,1\,0\,1\,1\,0\,0\,1 \end{pmatrix}$$

Weitere, komplexere Verfahren zur Binarisierung werden im Kapitel „Binarisierungsverfahren“ vorgestellt, nämlich *MINA*, *Sigma**, *MEAN*, *MILO* und *SILO*, die alle eine ihnen eigene Charakteristik zeigen.

Erhält man zu einer ersten Frage f_0, der *Startfrage*, eine Antwort a_1, dann lässt sich diese als nächste Frage f_1 einsetzen, da sie das passende Format einer Frage besitzt, die Antwort a_2 wiederum als Frage f_2 und so fort. Es entsteht eine Folge von Antworten, die wegen des beschriebenen Iterationsverfahren als *Iteriertenfolge* oder *Iteriertenbahn* bezeichnet wird. Anders gesagt wird mit Hilfe der Matrix eine *Assoziationskette* gebildet, eine Iteriertenkette, wie sie auch in der *Assoziativen Programmierung* Verwendung findet, um einen Programmablauf darzustellen und abzuwickeln.

Auch Zeichenketten lassen sich als Assoziationsketten auffassen und in Assoziativmatrizen ablegen, indem zum Beispiel ein Wort wie `KUPFERKESSEL` durch einen Repräsentanten für das `K` mit einem für das `U`, einem für das `U` mit einem für das `P` und so fort verbunden wird. Es genügt dann, sich an den ersten Repräsentanten zu erinnern und die Matrix damit abzufragen: das gesamte Wort taucht bei sorgfältiger Zuweisung der Repräsentanten wieder auf. Dabei ist zu bedenken, dass jeder der Repräsentanten genau nur eine Antwort liefern darf, damit man das Ende der Zeichenkette sicher erreicht. Hier müsste sich also der Repräsentant für das erste `K` von dem für das zweite `K` unterscheiden, der für das erste `E` von dem für das zweite `E` und so weiter. Die Assoziationskette darf zudem nicht mittendrin abbrechen (*Nullaktivität*), sich in einer Endlosschleife verfangen (*Endloskette*) oder in einer Antwortschwemme münden (*Überreizung*).[9] Diese Erfordernisse sind auch für die im nächsten Kapitel „Iteriertenbahnen“ vorgestellten Assoziationsketten von Bedeutung, die man im gleichen Sinne wie Zeichenketten auffassen kann, bei denen auf jedes Zeichen genau nur ein bestimmtes Zeichen folgt, auf jede Iterierte genau nur eine nachfolgende Iterierte.

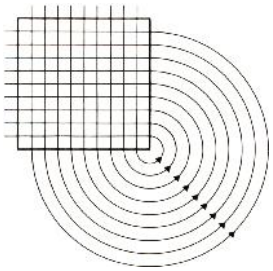


Abb. 6: Eine erste Darstellung einer Fortsetzungsassoziation stammt aus Palm [1982], S. 50.[10]

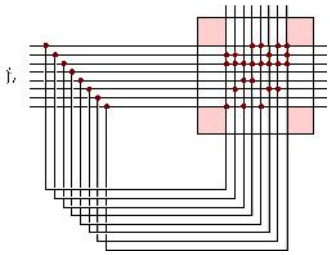


Abb. 7: Technisch gesehen legt man die Antwortleitungen im nächsten Takt wieder auf die Frageleitungen f_i.

Anmerkungen

[1] Als Maßnahmen zur informationstechnischen Sicherheit, der IT–Sicherheit, sind hier solche im Blick, die eine missbräuchliche Nutzung von Daten oder deren Manipulation verhindern können.

[2] s. Palm [1982], S. 193f, oder Steinbuch [1965], S. 220ff

[3] *lat.* associare — in Verbindung bringen, miteinander vereinigen

[4] vgl. Bentz and Dierks [2013], S. 1f

[5] Man denke zum Beispiel an das Kurz- und Langzeitlernen in Bentz and Dierks [2013], S. 40ff.

[6] Für eine Übersicht theoretischer Untersuchungen zu einfachen neuronalen Netze s. Palm and Bentz [2021].

[7] n–Tupel — verallgemeinert aus dem Lateinischen von quintuplus „fünffach“, septuplus „siebenfach“, centuplus „hundertfach“ für eine endlich lange Liste von n Objekten, hier von Nullen und Einsen; zur Multiplikation s. Bentz and Dierks [2013], S. 18

[8] Abk.: LSB — least significant bit

[9] s. Bentz and Dierks [2013], S. 140

[10] Die Leitungswege zur Fortsetzungsassoziation werden in Palm [1982] *additional feedback connections* genannt, womit die Idee einer Rückkopplung zum Ausdruck kommt, womöglich zur Verbesserung einer Erkennungsleistung.

Iteriertenbahnen

Schreibt man die durch Fortsetzungsassoziation gewonnenen Iterierten untereinander, erhält man einen *Iteriertenteppich* wie er in den Abb. 8 und 9 in Teilen abgebildet ist. Man erkennt, dass diese Teppiche schon dem Augenschein nach vom gewählten Binarisierungsverfahren abhängen. In Abb. 8 wurde er durch *LSB* und in Abb. 9 durch *SILO* auf dieselbe Startfrage hin erzeugt.

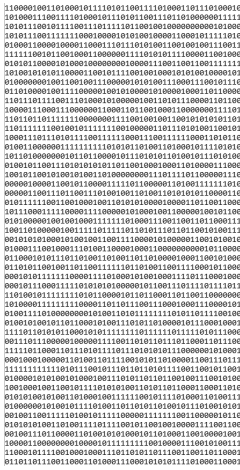


Abb. 8: Ausschnitt aus einem durch LSB erzeugten Iteriertenteppich

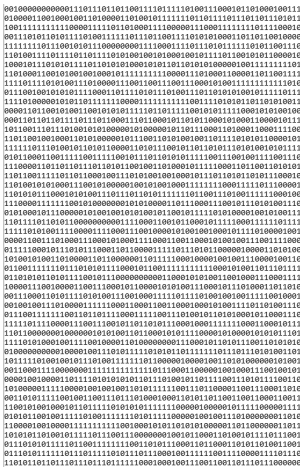


Abb. 9: Ausschnitt aus einem durch SILO erzeugten Iteriertenteppich

Eine weitere Methode zur Veranschaulichung und Erläuterung, wie unterschiedlich mögliche Iteriertenfolgen aussehen können, ist die Darstellung als Graph der Iteriertenbahnen wie in Abb. 10 geschehen.

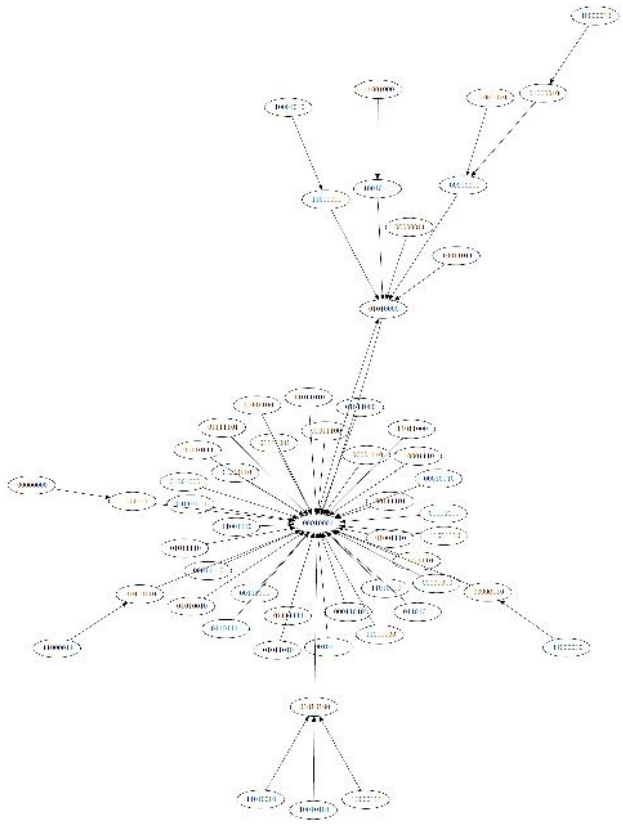


Abb. 10: Als Graph dargestellte Iteriertenbahnen

Hier war eine zeilenstarre 8×8–Matrix (s. Abb. 12) der Anlass für die Aufgabenstellung, alle möglichen Iteriertenbahnen bildlich zu erfassen. Die Abb. 10 zeigt nur einen der beiden sich ergebenden Teilgraphen, Abb. 11 zeigt den zweiten.

Abb. 11: Zweiter Teilgraph mit den weiteren Iteriertenbahnen zu denen aus Abb. 10

Diese Graphen der Iteriertenbahnen werden kurz *Vektographen* genannt.[11] Den Vektographen entnimmt man eine Reihe von Beobachtungen, die man hinsichtlich informationstechnischer Vor- und Nachteile bewertet. Folgende Fälle erkennt man:

a) Eine Iteriertenbahn bildet einen zyklischen (Teil-) Graphen. Wenn die Bahn so lang ist, dass sich alle unsere Daten damit wiederholungsfrei bearbeiten lassen, dann käme sie für unsere Zwecke gut in Frage.

b) Eine Iteriertenbahn stellt sich im Vektographen als Einzelknoten dar. Die Iteriertenbahn erweist sich damit für unsere Belange als unbrauchbar. Die *Nullfrage* ergibt immer einen Einzelknoten.

c) Eine Iteriertenbahn bildet zwar keinen zyklischen Graphen, ist aber verzweigungsfrei und lang genug, so dass alle Daten wie in a) verarbeitet werden können.

d) Als *End*–Knoten werden solche Knoten bezeichnet, die keinen Nachfolger haben. Eine Bahn, die in einem Endknoten aufhört, müsste wie in c) lang genug sein, um unsere Zwecke zu erfüllen.

Die zeilenstarre Matrix, die den in den vorstehenden Abbildungen gezeigten Vektographen erzeugt, ist in Abb. 12 angegeben. In diesem Beispiel ist das 8–Tupel $f = (1\,0\,0\,0\,0\,0\,0\,1)$ die Startfrage. Damit folgt die Rechnung:

$$\begin{pmatrix} 1\,0\,0\,0\,0\,0\,0\,1 \end{pmatrix} \cdot \begin{pmatrix} 1\,1\,0\,0\,0\,1\,1\,0 \\ 0\,0\,0\,1\,0\,1\,1\,1 \\ 0\,1\,1\,0\,0\,1\,0\,1 \\ 0\,1\,0\,1\,1\,0\,0\,1 \\ 1\,0\,0\,1\,1\,0\,0\,1 \\ 0\,0\,0\,1\,1\,1\,0\,1 \\ 0\,1\,0\,1\,0\,0\,1\,1 \\ 1\,1\,0\,1\,0\,1\,0\,0 \end{pmatrix}$$

$$= \begin{pmatrix} 2\,2\,0\,1\,0\,2\,1\,0 \end{pmatrix}$$

Abb. 12: Multiplikation einer Frage mit einer zeilenstarren 8×8–Matrix.

Zur Binarisierung sei dieses Mal das MAX–Verfahren gewählt, womit alle Werte auf Eins abgebildet werden, die gleich dem größten Wert im Zwischenergebnis sind.[12] Der größte Wert ist hier die 2, also:

$$\begin{pmatrix} 2\,2\,0\,1\,0\,2\,1\,0 \end{pmatrix} \xrightarrow{\text{MAX}} \begin{pmatrix} 1\,1\,0\,0\,0\,1\,0\,0 \end{pmatrix}$$

Nun fragt man in derselben Weise die Matrix mit $\begin{pmatrix} 1\,1\,0\,0\,0\,1\,0\,0 \end{pmatrix}$ ab und erhält $\begin{pmatrix} 0\,0\,0\,0\,0\,1\,0\,0 \end{pmatrix}$ und so fort.

Insgesamt ergibt sich folgende Iteriertenbahn:

$$\begin{aligned} f_0 &= \begin{pmatrix} 1\,0\,0\,0\,0\,0\,0\,1 \end{pmatrix} \\ f_1 &= \begin{pmatrix} 1\,1\,0\,0\,0\,1\,0\,0 \end{pmatrix} \\ f_2 &= \begin{pmatrix} 0\,0\,0\,0\,0\,1\,0\,0 \end{pmatrix} \\ f_3 &= \begin{pmatrix} 0\,0\,0\,1\,1\,1\,0\,1 \end{pmatrix} \\ f_4 &= \begin{pmatrix} 0\,0\,0\,1\,0\,0\,0\,0 \end{pmatrix} \\ f_5 &= \begin{pmatrix} 0\,1\,0\,1\,1\,0\,0\,1 \end{pmatrix} \\ f_6 &= f_4 \end{aligned}$$

Eine Ausschnittsvergrößerung des Teilgraphen aus Abb. 11 zeigt diese Iteriertenbahn in Abb. 13, die von rechts nach links verläuft und farblich hervorgehoben wurde:

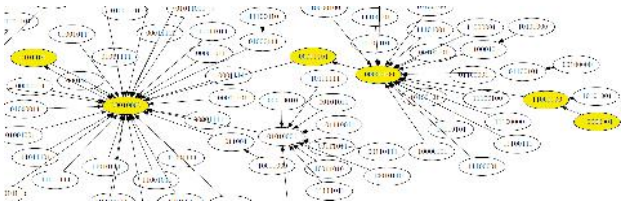


Abb. 13: Iteriertenbahn im Vektographen aus Abb. 11

Was die Vektographen sehr verdeutlichen, das ist, dass jemand, der die Startfrage nicht kennt, die Iteriertenbahn aber dennoch durch Ausprobieren erfahren möchte, vor einer Vielzahl von Möglichkeiten steht, in „falsche“ Bahnen einzusteigen. Zu seinem Nachteil gehört auch die Möglichkeit, zwar die richtige Bahn erwischt zu haben, aber nicht deren Startfrage.

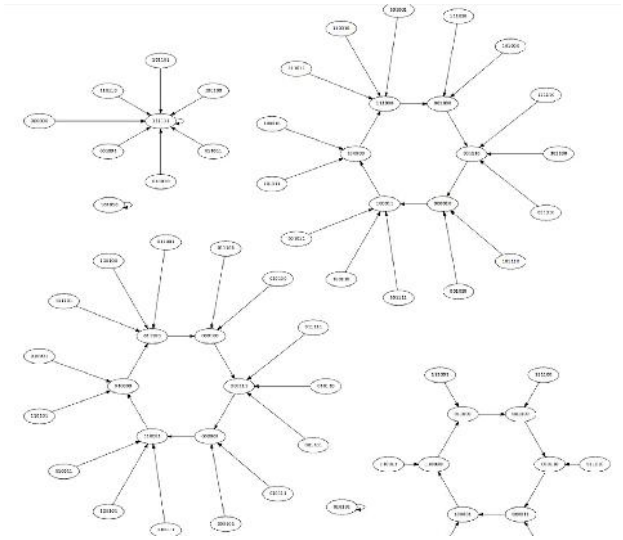


Abb. 14: Vektograph zu einer 6×6–Matrix

Die Vielfalt an Vektographen–Formen zeigt sich in der voran- und der nachstehenden Abbildung. Mal bilden sich wenige mal viele Teilgraphen.

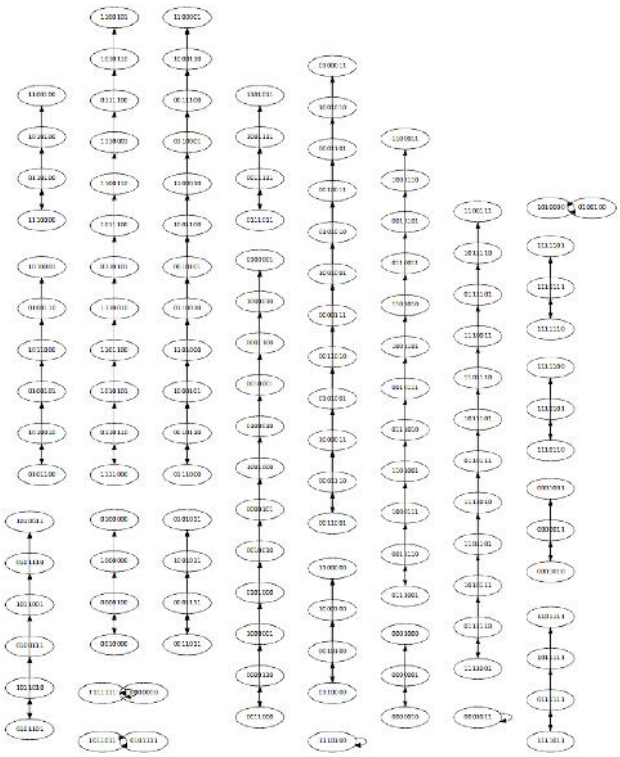


Abb. 15: Vektograph zu einer 7×7–Matrix

Auch schon am Beispiel der zeilenstarren 8×8–Matrix aus Abb. 12 wird deutlich, wie die Form sich wandelt, wenn man bei ihr nur statt des MAX–Verfahrens das LSB–Verfahren anwendet. Der Vektograph nimmt trotz gleicher Matrix die Gestalt an, die in Abb. 16 zu sehen ist.

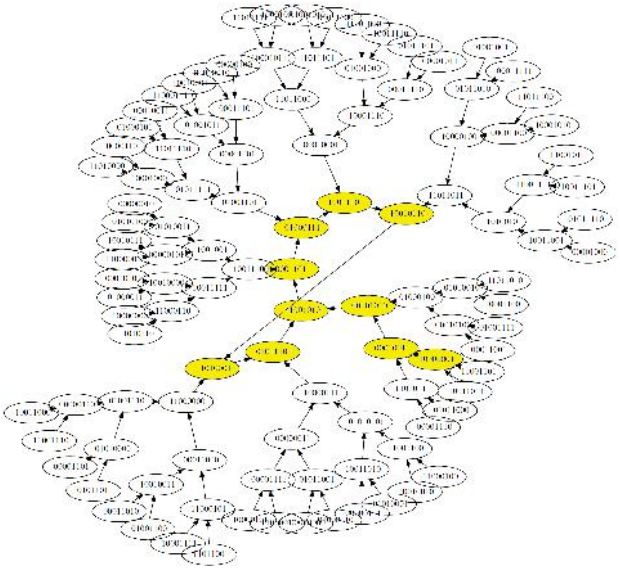


Abb. 16: Vektograph einer LSB–Iteriertenbahn zur Matrix aus Abb. 12

Trotz gleicher Startfrage und gleicher Matrix besteht die Bahn nunmehr aus den Iterierten:

$$\begin{aligned} f_0 &= \begin{pmatrix} 1\,0\,0\,0\,0\,0\,0\,1 \end{pmatrix} \\ f_1 &= \begin{pmatrix} 0\,0\,0\,1\,0\,0\,1\,0 \end{pmatrix} \\ f_2 &= \begin{pmatrix} 0\,0\,0\,0\,1\,0\,1\,0 \end{pmatrix} \\ f_3 &= \begin{pmatrix} 1\,1\,0\,0\,1\,0\,1\,0 \end{pmatrix} \\ f_4 &= \begin{pmatrix} 0\,0\,0\,1\,1\,0\,1\,1 \end{pmatrix} \\ f_5 &= \begin{pmatrix} 0\,1\,0\,0\,0\,1\,1\,1 \end{pmatrix} \end{aligned}$$

$$
\begin{aligned}
f_6 &= \begin{pmatrix} 1\,0\,0\,0\,1\,1\,0\,1 \end{pmatrix} \\
f_7 &= \begin{pmatrix} 1\,0\,0\,1\,0\,1\,1\,0 \end{pmatrix} \\
f_8 &= \begin{pmatrix} 1\,1\,0\,1\,0\,0\,0\,1 \end{pmatrix} \\
f_9 &= \begin{pmatrix} 0\,1\,0\,1\,1\,1\,0\,0 \end{pmatrix} \\
f_{10} &= f_3
\end{aligned}
$$

Diese Bahn mit ihrem typischen Rücksprung von der Iterierten f_{10} zur Iterierten f_3 wurde in Abb. 16 farblich hervorgehoben.

Vektographen können bezüglich des Zusammenhangs der Teilgraphen sehr verschiedenartig sein, wie die Abb. 17 und 18 zeigen.

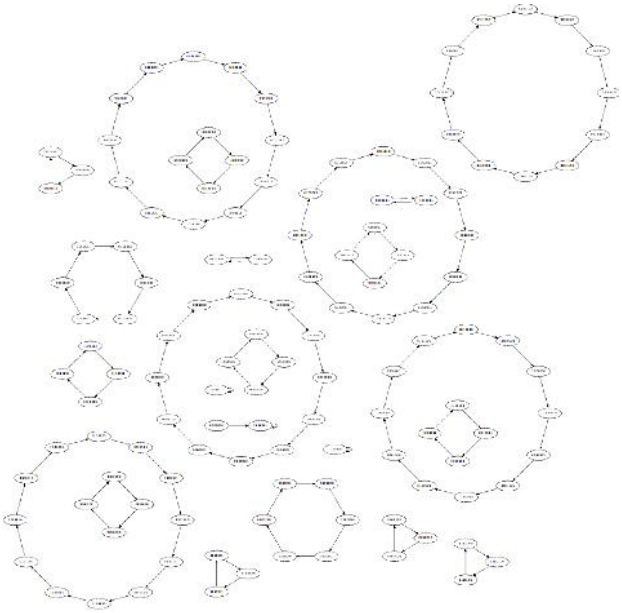


Abb. 17: Vektograph zu einer 7×7–Matrix, der aus vielen zyklischen Teilgraphen besteht

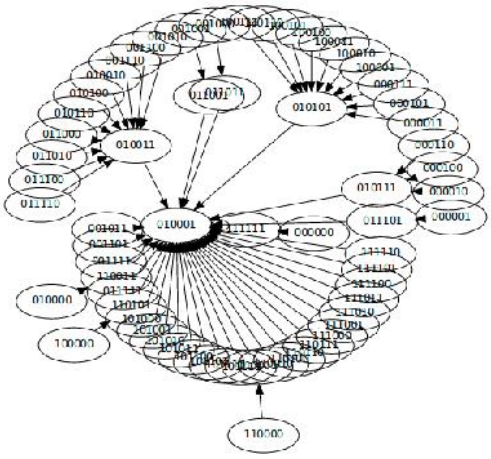


Abb. 18: Vektograph zu einer 6×6–Matrix, die nur eine einzige zusammenhängende Bahn erzeugt

Die Beobachtungen an Vektographen lassen sich mit den im vorherigen Kapitel beschriebenen Bedingungen verknüpfen, in denen vom Vermeiden von Nullaktivität oder Überreizung die Rede war. Die eingesetzten Verfahren zum Binarisieren sollten so konstruiert sein, dass die Iteriertenbahnen weder das eine noch das andere zulassen. Beim LSB–Verfahren zeigt sich eine Nullaktivität zum Beispiel, wenn man die Matrix aus Abb. 5 zunächst mit der Startfrage $\begin{pmatrix} 1\,0\,0\,0\,0\,0\,1\,1 \end{pmatrix}$ multipliziert:

$$
\begin{pmatrix} 1\,0\,0\,0\,0\,0\,1\,1 \end{pmatrix} \cdot \begin{pmatrix}
0\,0\,1\,0\,1\,1\,0\,0 \\
1\,1\,1\,0\,0\,1\,1\,0 \\
1\,0\,1\,1\,0\,0\,1\,1 \\
0\,1\,1\,0\,0\,1\,0\,0 \\
0\,0\,0\,1\,0\,0\,1\,1 \\
1\,1\,0\,1\,1\,1\,0\,1 \\
0\,1\,0\,1\,1\,0\,1\,1 \\
1\,0\,0\,0\,1\,0\,0\,0
\end{pmatrix}
$$

$$= \begin{pmatrix} 1\,1\,1\,1\,3\,1\,1\,1 \end{pmatrix}$$

Abb. 19: Zwischenergebnis mit ausschließlich ungeraden Werten

Das LSB–Verfahren liefert daraufhin lauter Einsen in die Antwort

$$\begin{pmatrix} 1\,1\,1\,1\,3\,1\,1\,1 \end{pmatrix} \xrightarrow{\text{LSB}} \begin{pmatrix} 1\,1\,1\,1\,1\,1\,1\,1 \end{pmatrix}$$

und das führt im nächsten Abfrageschritt wegen der spaltenstarren Matrix zur Nullaktivität. Die Iteriertenbahn erreicht ihr womöglich vorzeitiges Ende.

Für größere Matrizen ist es in der Praxis nicht möglich, den Vektographen zu zeichnen, um eine für uns geeignete Iteriertenbahn zu bestimmen. Bei einer Matrixgröße von $n = 128$, wie wir sie mindestens wählen, wären über 10^{38} Knoten darzustellen. Wir gehen dann so vor, dass wir nach Gutdünken eine Startfrage bestimmen, erzeugen oder aus einer Sammlung nehmen und damit die zu nutzende Bahn für die Verarbeitung der Datenblöcke festlegen. Unter *Verarbeitung der Datenblöcke* wird verstanden, dass sie für die informationstechnisch sichere Übertragung oder Ablage auf einem Datenträger mit den Iterierten einer Bahn geXORt werden.[13]

Weiter oben war bereits angenommen worden, es gäbe jemanden, der die Iteriertenbahn erfahren möchte, ohne die Startfrage zu kennen. Nehmen wir zusätzlich an, er hätte Kenntnis von der spaltenstarren Matrix aus Abb. 5 erhalten und er weiß, dass LSB genutzt wurde. Dann würden seine kombinierenden Versuche auch die Suche nach dem Vorgänger von $\begin{pmatrix} 2\,2\,3\,2\,2\,3\,1\,2 \end{pmatrix}$ beinhalten, der laut Abb. 5 die Frage $\begin{pmatrix} 1\,0\,1\,1\,0\,1\,0\,0 \end{pmatrix}$ war. Das Lösen des zugehörigen linearen Gleichungssystems führt ihn auf die folgende Treppenstufenform:[14]

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & | & 2 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & | & 2 \\
0 & 0 & 1 & -1 & 0 & 0 & 0 & -1 & | & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & | & 2 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & | & 2 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & | & -3 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & | & -\frac{5}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & | & 1
\end{pmatrix}
$$

Man erkennt an der letzten Zeile, dass dieses Gleichungssystem keine Lösung hat. Der Vorgänger kann nicht berechnet werden. Das ist generell von uns erwünscht, denn die Rückverfolgung innerhalb einer Bahn soll erschwert sein, wenn nicht sogar praktisch unmöglich werden.

Diese Eigenschaft wird auch genutzt, um die Startfragen (*Startschlüssel*) zu verwalten, die man verschiedenen Anwendern zur Verarbeitung ihrer Daten überlässt. Hierfür genügt es im Allgemeinen nicht, jedem Mitarbeiter einen Startschlüssel für seine eigene Iteriertenbahn zuzuweisen, sondern man sollte auch ermöglichen, dass es so etwas wie einen übergeordneten Schlüssel gibt, mit dem im Notfall die abgelegten Daten aller Mitarbeiter wieder zugänglich gemacht werden können. Also besitzt der Mitarbeiter einen Startschlüssel, mit dem er seine Daten verarbeitet, über den er aber nicht die Daten seiner Kollegen erreichen kann. Das gelingt jedoch mit dem übergeordneten Schlüssel, da sich der Startschlüssel des Mitarbeiters selbst wieder über eine Iteriertenbahn aus dem übergeordneten Schlüssel ableiten lässt. Umgekehrt ginge das nicht, da die Rückverfolgung seines Startschlüssels auf der Iteriertenbahn dem Mitarbeiter praktisch unmöglich ist.

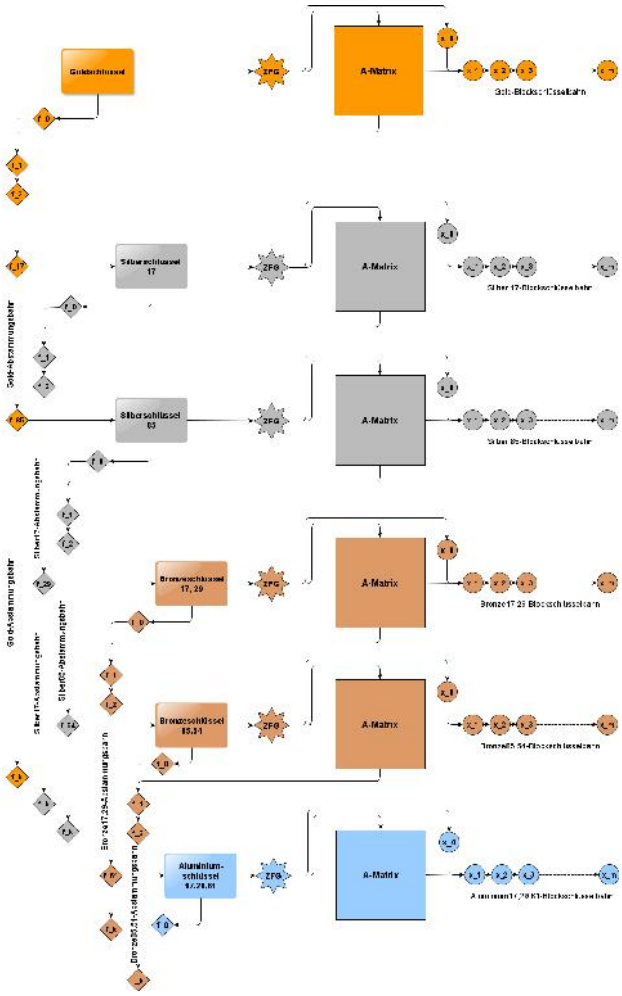


Abb. 20: Ein beliebig kaskadierbares Gold–Silber–Bronze–Konzept regelt die Schlüsselvergabe, also die Zuweisung von Iteriertenbahnen an die Mitarbeiter.

In Abb. 20 sind links die Iteriertenbahnen für Gold-, Silber-, Bronzeschlüssel und weiteren daraus ableitbaren Schlüsseln dargestellt. Rechts erkennt man die Iteriertenbahnen, die der jeweilige Mitarbeiter mit seinem Schlüssel zur Verarbeitung seiner Daten einsetzt.[15]

Für den Austausch von Schlüsseln über eine öffentliche Leitung wird der zunächst noch allbekannte Diffie–Hellmann–Schlüsselaustausch eingesetzt. Inwieweit das auf Grundlage der in diesem Kapitel vorgestellten Iteriertenbahnen möglich ist, wurde in folgender Weise angedacht, wobei Alice und Bob die beiden sind, die den Schlüssel austauschen wollen: „Nehmen wir [eine Iteriertenbahn], die auf einer Assoziativmaschine [...] erzeugt wurde. Dann könnte Alice in Analogie zum ECDH-Verfahren[16] d_A Schritte in der Kette vorwärts laufen bis zur Antwort Q_A und Bob d_B Schritte bis zur Antwort Q_B. Eine gemeinsame Startfrage f_G entspricht dann dem Generatorpunkt G. Damit wäre ein Diffie–Hellman-Verfahren für die Assoziativmaschine beschrieben (ASDH).“[17] Eine praktische Erprobung dieses Schlüsselaustauschsverfahrens steht noch aus. Da sich aber der Umgang mit einer Iteriertenbahn und einer elliptische Kurve in diesen Belangen ähnelt, liegt ein solcher Versuch von uns nahe.[18]

Anmerkungen

[11] Der Begriff Vektor wird in diesem Zusammenhang nicht im Sinne der Mathematik oder Physik, sondern wie in der Informatik als Bezeichnung für eine Liste oder Reihung benutzt, hier also einer Reihung von Nullen und Einsen.

[12] Das MAX–Verfahren ist dasjenige, welches üblicherweise bei Assoziativmatrizen und -maschinen angewandt wird (s. „Abfrageregel“ in Bentz and Dierks [2013], S. 43f).

[13] Die aussagenlogische Verknüpfung XOR (exklusives ODER) verknüpft die Bits des Datenblocks mit denen der Iterierten in einer Weise, dass bei einer abermaligen Verknüpfung mit derselben Iterierten wieder die ursprünglichen Daten entstehen.

[14] Treppenstufenform, engl. *row reduced echelon form* (rref), die hier durch das CAS *Maxima* errechnet wurde

[15] Die Einzelheiten dieses Konzepts entnehme man „Gold-, Silber-, Bronzeschlüssel“ in Dierks [02. Dezember 2019].

[16] ECDH — Diffie–Hellmann–Verfahren mit elliptischen Kurven

[17] vgl. Dierks [08. Mai 2019], S. 10

[18] s. S. 22

Zufallsgeneratoren

Es hat einen Einfluss auf die Iteriertenbahnen, wie die sie erzeugende Assoziativmatrix mit Einsen gefüllt wird. Die Unterschiede werden im zugehörigen Vektographen, falls man ihn zeichnen könnte, oder im Iteriertenteppich deutlich, wie man in den Abb. 21 und 22 mit bloßem Auge erkennt.

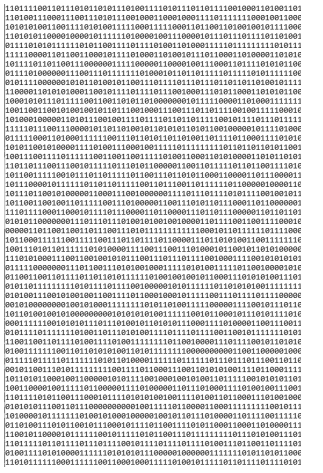


Abb. 21: MINA–Teppich einer spaltenstarren Matrix

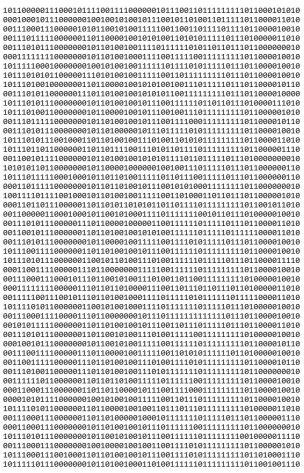


Abb. 22: MINA–Teppich einer zeilenstarren Matrix

Wir stellten fest, dass sich für unsere Zwecke spaltenstarre oder doppelstarre Matrizen eignen, da sie keine Iteriertenbahnen mit Auffälligkeiten wie in Abb. 22 liefern. Ob die Anzahl an verfügbaren Matrizen durch diese Einschränkung auf starre Matrizen empfindlich verkleinert wird, beantworten Canfield and McKay † [2004]: Für eine Matrixgröße $n = 30$ zum Beispiel gibt es bereits $7,5 \cdot 10^{221}$ doppeltstarre Matrizen mit $p = \frac{1}{2}n$. Wir halten zudem fest, dass Matrizen der Art

$$\begin{pmatrix} 1&1&0&0&0&0&1&1 \\ 1&1&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 1&1&0&0&0&0&1&1 \\ 1&1&0&0&0&0&1&1 \end{pmatrix}$$

zwar doppeltstarr mit $p = \frac{1}{2}n$ sind, aber dass deren Zeilen- und Spaltensummen nicht die benötigte Verteilung aufweisen und für uns daher nicht in Frage kommen.[19] Wir setzen darauf, dass Zufallsprozesse die Einsen in den Matrizen anwendungsgerecht verteilen. Daher wurde bereits in der Definition von Starrheit im Kapitel „Einordnung“ die zufällige Füllung der Matrixzeilen und -spalten genannt.

Zur Erzeugung starrer Matrizen hat man die Wahl zwischen dem Einsatz natürlicher Zufallsquellen, zum Beispiel durch *HotBits*[20], dazu später mehr, und dem Einsatz von Zufallsgeneratoren wie etwa besonderen Schieberegistern, deren Kombination oder deren algorithmische Nachbildung durch ein Programm.

In Schneier [1996], S. 383, wird eine Kombination aus drei verschieden langen, linear rückgekoppelten Schieberegistern vorgeschlagen. Sie sind in Abb. 24 mit den Abkürzungen LFSR-1, LFSR-2 und LFSR-3 eingetragen.[21] Bei einem rückgekoppelten Schieberegister handelt es sich um ein zusammengesetztes elektronisches Bauteil, an dessem einen Eingang ein Taktgeber den Zustand der bistabilen Kippstufen (Flipflops), aus denen das Schieberegister besteht, um jeweils einen Platz pro Takt weiterschieben kann. Einige der Flipflop–Zustände werden verknüpft auf den anderen Eingang zurückgekoppelt (feedback) wie in Abb. 23 gezeigt, damit auch dieser einen wohldefinierten Folgezustand einnehmen kann. Am Ausgang, in Abb. 23 mit Y beschriftet, liegt dann bei jedem Takt ein Bit einer Zufallsbitfolge an, dessen Periodenlänge man möglichst groß werden lässt. Um eine optimale Periodenlänge zu erreichen, also um die Anzahl an Takten, ab der sich die Zufallsbitfolge wiederholt, so groß wie möglich zu machen, nutzt man die mathematische Erkenntnis, dass die Rückkopplung bestimmter Flipflop–Zustände auf den Eingang die Periodenlänge bei einem Schieberegister aus m Flipflops auf $2^m - 1$ Takte optimiert.[22] Wir haben uns für ein LFSR-1 mit 61 Flipflops, ein LFSR-2 mit 73 Flipflops und ein LFSR-3 mit 89 Flipflops entschieden.[23]

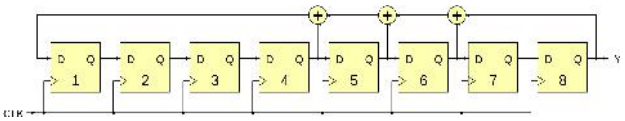


Abb. 23: Ein lineares, rückgekoppeltes Schieberegister aus acht Flipflops

Der in Abb. 24 gezeigte und von uns zur Matrixbefüllung eingesetzte Zufallszahlengenerator wird als alternierender Stop–and–go–Generator bezeichnet, weil das vorgeschaltete Schieberegister LFSR-1 bei jedem eingehenden Taktimpuls bestimmt, welches der beiden anderen Schieberegister LFSR-2 oder LFSR-3 weitergetaktet werden soll. Die Ausgänge von LFSR-2 und LFSR-3 werden mit XOR verknüpft und als Zufallsbit $b(t)$ geliefert.[24]

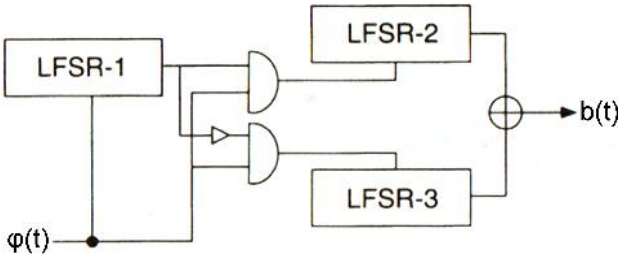


Abb. 24: Ein alternierender Stop–and–go–Generator aus drei linearen, rückgekoppelten Schieberegistern

Wichtig ist, dass sich weder in LFSR-1 noch in LFSR-2 noch in LFSR-3 alle Flipflops im Zustand 0 befinden. Wenn alle Flipflops eines der Schieberegisters den Zustand 0 haben, verlassen sie diesen Zustand nie wieder. Unser Zufallsgenerator muss zu Beginn seiner Nutzung mit einer *Saat* aus 30 Bytes versehen werden, die keines der Schieberegister ausschließlich mit Nullen befüllt. Von den 30 Bytes werden die ersten 8 Bytes für das LFSR-1, die nächsten 10 Bytes für das LFSR-2 und die restlichen 12 Bytes

für das LFSR-3 genommen.

Die Saat für unseren Zufallsgenerator (s. Abb. 24) legt eindeutig fest, an welchen Plätzen die Matrix Einsen erhält, damit sie spaltenstarr wird. Durch eine gleiche Saat kann diese Matrix jederzeit wieder reproduziert werden. Die Saat muss also nicht abgespeichert werden, sondern bleibt ein (mittelbares) Geheimnis. Die Saat wird aus dem Startschlüssel des Anwenders abgeleitet (s. Kapitel „Iteriertenbahnen, Schlüsselkonzept“).

Möchte man die Assoziativmatrix nicht über eine Saat per Zufallsgenerator herstellen, sondern die Befüllung mit Einsen dem natürlichen Zufall überlassen oder einem anderen Verteilungsprinzip, dann wäre eine solche Matrix als Geheimnis aufzubewahren und bei Bedarf vom Anwender ins Gerät zu laden.

Der bekannte Programmierer John Walker wirbt für die Nutzung natürlichen Zufalls und kritisiert den Pseudozufall, der von Zufallsgeneratoren maschinell generiert wird.[25] Daher bietet er Zufallsdaten an, die einem radioaktiven Zerfallsprozess entnommen sind und fortlaufend werden.[26] Seinen Generator nennt er *HotBits*. Mit dessen Daten ließe sich die Matrix befüllen.

Vor diesem Hintergrund fiel auf, dass bei der Entwicklung einer unterbrechungsfreien Stromversorgung (USV) für unser Gerät ebenfalls Zufallsereignisse zu beobachten waren. Die Leuchtdioden (LED), die die Lade–Entlade–Phasen der vier Akkumulatoren anzeigten, blinkten in augenscheinlich zufälliger Weise (s. Abb. 25).[27]

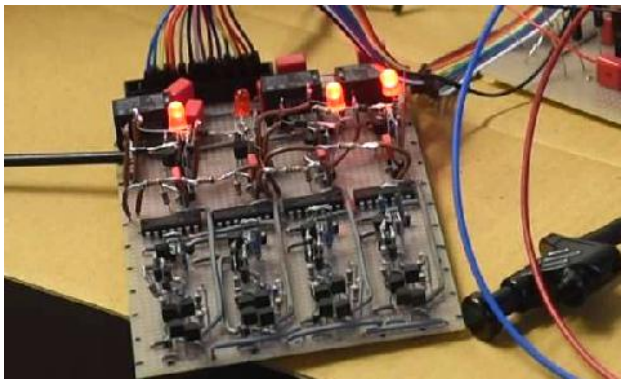


Abb. 25: Lade–Entlade–Anzeigen der Ladeeinrichtung der Akkumulatoren der USV

Zur Anwendung des HotBits–Verfahrens wäre hier das Blinken einer LED als ein Ereignis zu werten. Misst man die Zeit T_1, die zwischen zweien solcher Ereignisse verstreicht, und dann auch noch die Zeit T_2 zwischen den nächsten beiden Ereignissen, so liefert man als Zufallsbit eine 1, falls $T_1 > T_2$, eine 0, falls $T_1 < T_2$, und bei $T_1 = T_2$ liefert man nichts, sondern wartet auf das nächste Ereignispaar.

Zudem tauscht man nach jeder Abgabe eines Zufallsbits die Bedingungen für die 1 und die 0 um. Bei den nächsten zwei Ereignispaaren wird mit einer 1 geantwortet, falls $T_1 < T_2$ und mit einer 0, falls $T_1 > T_2$ ist. Damit verhindert man Walker zufolge nichtzufällige Einflüsse durch Versuchsaufbau oder Messgerät. Eine praktische Erprobung des Einsatzes dieses Verfahrens zur Befüllung unserer Matrix steht noch aus.

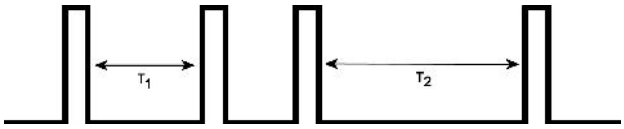


Abb. 26: HotBits–Verfahren mit der Messung von Zeitabständen T_1, T_2 zwischen je zwei Ereignissen

Wie auch immer die Matrix befüllt wurde, durch einen Zufallsgenerator oder durch natürlichen Zufall, das Ergebnis für die späteren Antworten, die die Matrix liefert, hängt vornehmlich davon ab, ob man damit eine spaltenstarre, zeilenstarre, doppeltstarre oder „freie“ Matrix erzeugt hat. Die folgenden Abbildungen veranschaulichen, wie sich die Ergebnisse für das Aufeinandertreffen von starr und frei befüllten Fragen auf starr und frei befüllten Matrixspalten unterscheiden.

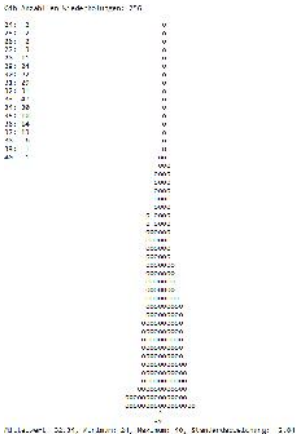


Abb. 27: 256 starre Fragen treffen auf starre Matrixspalten.[28]

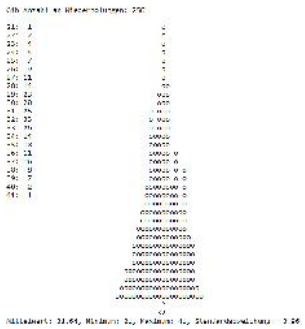


Abb. 28: 256 starre Fragen treffen auf frei befüllte Matrixspalten.

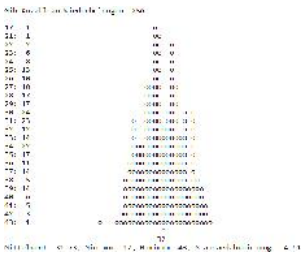


Abb. 29: 256 frei befüllte Fragen treffen auf frei befüllte Matrixspalten.[29]

Die Fragen und Matrixspalten sind bei den Experimenten aus den Abb. 27, 28 und 29 als 128–Tupel angesetzt worden, als Fragen und Matrixspalten, die zur Hälfte zufällig mit Einsen befüllt wurden. Bei starren Fragen und Matrixspalten sind es also genau 64 Einsen, bei freien wahrscheinlich weniger.

Das Aufeinandertreffen von Fragen und Matrixspalten ist mathematisch gesehen als Skalarmultiplikation auszuführen. Die Unterschiede in der Häufigkeitsverteilung der Skalarprodukte begründen die Unterschiede in deren Häufigkeit und Streuung. Die Verteilung der Produkte ist in dem Fall aus Abb. 27 hypergeometrisch, im Falle von Abb. 29 binomial. Die Verteilung aus Abb. 28 liegt dazwischen und entspricht auch dem

umgekehrten Fall, dass eine frei befüllte Frage auf eine starre Matrixspalte trifft.

Die Abb. 27 bis 29 lassen im Vergleich erkennen, dass beim Abfragen einer spaltenstarren Matrix der häufigste Wert deutlich häufiger vorkommt als der häufigste Wert bei einer frei befüllten Matrix. Zudem ist die Streuung der Produkte beim Abfragen spaltenstarrer Matrizen merklich geringer.

	starr | starr	starr | frei	frei | frei
Streuung	$\approx 2,8$	$\approx 4,0$	$\approx 4,9$

Die Streuung ist in dieser Tabelle durch die beobachtete Standardabweichung angegeben.

Die folgende Abb. 30 stellt die Unterschiede im theoretischen Idealfall dar. Die hellblau markierte Verteilung steht für das Aufeinandertreffen freier Fragen auf freie Matrixspalten, die rötlich markierte gibt den entsprechenden starren Fall wieder. Die Rechtsachse ist hier mit „Zählerstand“ statt mit „Skalarprodukt“ beschriftet, weil in der technischen Umsetzung ein Digitalzähler das Ereignis registriert, dass eine Eins der Frage auf eine Eins der Matrixspalte trifft.[30]

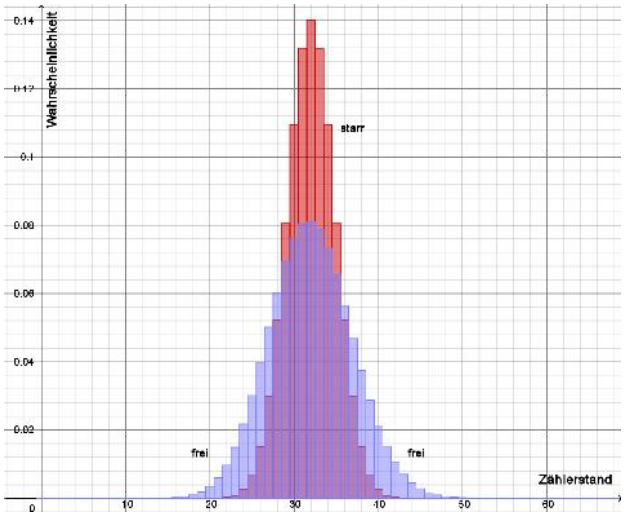


Abb. 30: Vergleich der Verteilungen

Die gezeigten Unterschiede haben Auswirkungen auf einige Binarisierungsverfahren, die im nächsten Kapitel vorgestellt werden, und das wiederum beeinflusst die davon erzeugten Iteriertenbahnen.

Anmerkungen

[19] Es wird erwartet, dass die häufigsten Spalten- beziehungsweise Zeilensummen bei 2 liegen und nicht bei 0 und 4.

[20] s. Walker [2017]

[21] LFSR — linear feedback shift register

[22] Zur geeigneten Auswahl der rückzukoppelnden Flipflops s. „LFSR für `plumChip` v2.0“ in Dierks [20. Oktober 2019].

[23] vgl. „Zufallsgenerator des `plumChips` in der Version v2.4“ in Dierks [05. August 2020]

[24] Schneier [1996] schreibt dazu auf S. 383: „This generator has a long period and large linear complexity. The authors found a correlation attack against LFSR-1, but it does not substantially weaken the generator.“

[25] s. Walker [2017]: „But numbers calculated by a computer through a deterministic process, cannot, by definition, be random. [...] HotBits is an Internet resource that brings genuine random numbers, generated by a process fundamentally governed by the inherent uncertainty in the quantum mechanical laws of nature, [...]“.

[26] Es werden die Zerfallsereignisse des Isotops Krypton–85 über ein Geiger–Müller–Zählrohr registriert. Jedes Ereignis wird über das Internet an Interessierte als Knackton hörbar weitergeleitet.

[27] s. Jan Weber. *Ladeeinrichtung für Akkumulatoren.* imbit.net, 04.12.2019

[28] Die experimentell bestimmte Standardabweichung von 2,84 stimmt mit dem für eine hypergeometrische Verteilung zu erwartenden Wert $\sigma = \sqrt{64 \cdot \frac{64}{128} \cdot (1 - \frac{64}{128}) \cdot \frac{128-64}{128-1}} \approx 2,84$ gut überein.

[29] Die experimentell bestimmte Standardabweichung von 4,91 stimmt mit dem für eine Binomialverteilung zu erwartenden Wert $\sigma = \sqrt{128 \cdot \frac{1}{4} \cdot \frac{3}{4}} \approx 4,899$ nahezu überein.

[30] Zu technischen Einzelheiten s. Dierks [2005].

Binarisierungsverfahren

Bisher wurden die Binarisierungsverfahren LSB, $\overline{LSB}$ und MAX vorgestellt.[31] Eines der wesentlichen Absichten der Binarisierungsverfahren ist, die Antworten möglichst zur Hälfte mit Einsen zu befüllen. Der Grund dafür liegt in der Überlegung, dass die Anzahl an Möglichkeiten, auf n Plätzen der Antwort $\frac{n}{2}$ Einsen unterzubringen, größer ist, als weniger oder mehr Einsen als $\frac{n}{2}$ in der Antwort zu verteilen. Nimmt man als Beispiel wieder eine 128×128–Matrix, dann geben die Binomialkoeffizienten $\binom{128}{k}$ über die Anzahl an Möglichkeiten Auskunft, wenn man k Einsen verteilen möchte. Es sind $\binom{128}{64} = 23.951.146.041.928.082.866.135.587.776.380.551.750$, also etwa 24 Sextillionen. Verteilt man hingegen nur ein Viertel an Einsen auf die 128 Plätze, ergeben sich $\binom{128}{32} = 1.477.806.921.502.280.666.682.474.774.300$, also „nur noch“ rund anderhalb Quintillionen Möglichkeiten. In der Nähe des Maximums stehen für $\binom{128}{63} = \binom{128}{65}$ gut 23,5 Sextillionen Möglichkeiten zur Verfügung, für $\binom{128}{60} = \binom{128}{68}$ sind noch knapp 18,7 Sextillionen Varianten möglich.

Damit kann man sich vorstellen, welchen Aufwand es für jemanden bedeutet, einen mit so einer Antwort/Iterierten verschlüsselten Datenblock zu untersuchen und auszuprobieren, ob er die Iterierte bestimmen kann. Der Aufwand steigt bei größeren Matrizen, zum Beispiel bei einer 256×256–Matrix auf $\binom{256}{128} \approx 5,77 \cdot 10^{75}$ Möglichkeiten, bei einer 512×512–Matrix auf $\binom{512}{256} \approx 4,73 \cdot 10^{152}$ und bei einer 1024×1024–Matrix auf $\binom{1024}{512} \approx 4,48 \cdot 10^{306}$ Möglichkeiten.[32]

Mit dem Ziel vor Augen, durch die Binarisierung möglichst jede Iterierte genau zur Hälfte mit Einsen zu füllen, stelle man sich das Zwischenergebnis der Abfrage einer 128×128–Matrix wie in Abb. 31 bildlich vor. Jeder Wert des Zwischenergebnisses wird als Balken gezeigt, dessen Fläche gleich dem Wert ist. Je höher der Balken, desto größer der Wert.

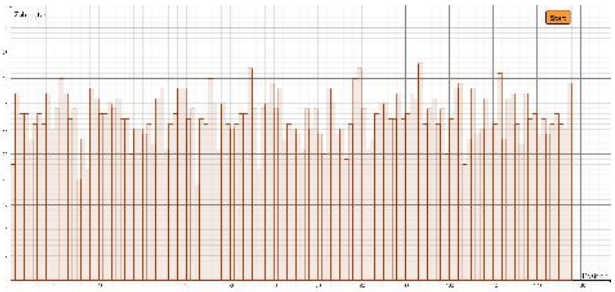


Abb. 31: Im Zwischenergebnis finden sich Werte zwischen 19 und 43.

Man erinnere sich, wie es zu einem Zwischenergebnis kommt (s. Abb. 5, Abb. 12 oder Abb. 19). Dieses soll nun durch ein möglichst wenig aufwendiges Verfahren wie gewünscht binarisiert werden. Die Werte (Zählerstände) des Zwischenergebnisses einer Abfrage mit einer zur Hälfte aus Einsen bestehenden Frage werden wie in Abb. 31 als Balkendiagramm dargestellt. Sie streuen hier etwa um 32. Der kleinste Wert ist 19, der größte 43.[33]

Führen wir nun einen, in den Abb. 32 bis 34 durch eine rote Waagerechte gezeigten *Schwellwert* s ein.[34] Diejenigen Balken, die höher oder gleich dem Schwellwert sind, bestimmen den Ort, in denen in die Antwort eine Eins eingetragen wird, sie werden farblich hervorgehoben, an den anderen Positionen wird eine Null gesetzt. Beginnen wir mit einem Schwellwert in Höhe des höchsten Balkens. Dann wird im in folgenden Abbildungen gezeigten Beispiel nur eine einzige Eins in die Antwort geschrieben. Also senken wir den Schwellwert s ab und erhalten bei $s = 38$ bereits 22 Einsen (s. Abb. 32), bei $s = 34$ sind es 58 Einsen (s. Abb. 33), womit das Ziel, nämlich 64 Einsen zu bestimmen, fast erreicht ist.

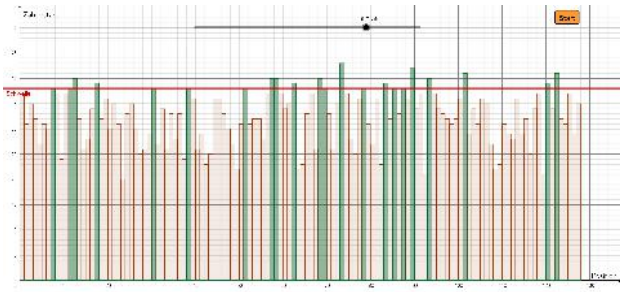


Abb. 32: *MINA*: Bei einem Schwellwert von $s = 38$ ergeben sich die Positionen von 22 Einsen.

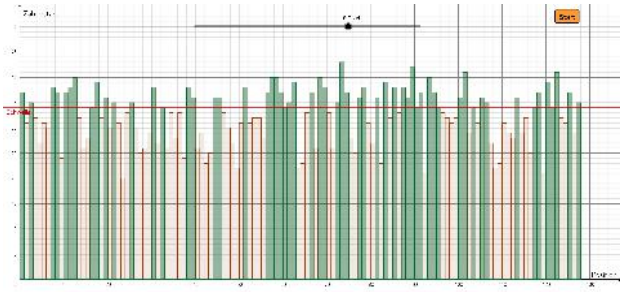


Abb. 33: *MINA*: Bei einem Schwellwert von $s = 34$ ergeben sich die Positionen von 58 Einsen.

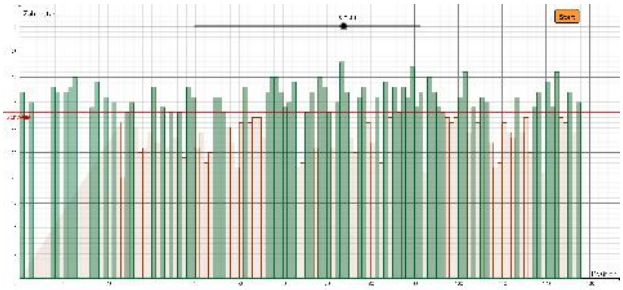


Abb. 34: *MINA*: Bei einem Schwellwert von $s = 33$ ergeben sich die Positionen von 66 Einsen.

Also senken wir den Schwellwert auf $s = 33$ (s. Abb. 34) und erhalten 66 Einsen. Das sind zwar zwei Einsen zu viel, aber wir sind dem Ziel, die Antwort zur Hälfte mit Einsen zu versorgen, ausreichend nahe gekommen.

Das soeben beschriebene und veranschaulichte Binarisierungsverfahren heißt *MINA*.[35]

Auch das schon erläuterte Binarisierungsverfahren *LSB*[36] kann mit den gleichen grafischen Werkzeugen erklärt werden (s. Abb. 35).

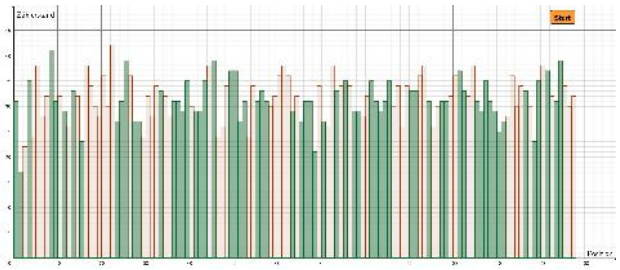


Abb. 35: *LSB*: Die Balken, die ungeradzahlige Werte anzeigen, liefern die Positionen für die Einsen in der Antwort.

Ein bisher noch nicht genanntes Binarisierungsverfahren ist das *Sigma*–Verfahren.[37] Es fußt auf den Sigma–Regeln für Normalverteilungen, denen zufolge in einer Umgebung von 0,674 Standardabweichungen um den Erwartungswert der Verteilung herum die Hälfte aller Zählerstände erwartet werden können.

Dieser Gedanke wird wie in Abb. 36 umgesetzt, indem man den Erwartungswert ermittelt, hier ist es 32 (grüne Waagerechte), und dann einen unteren und einen oberen Prüfwert (rote und blaue Waagerechte) in ungefähr einem Abstand von $0,674 \cdot \sigma = 0,674 \cdot \sqrt{128 \cdot \frac{1}{4} \cdot \frac{3}{4}} \approx 3$ bringt. Der obere Prüfwert ist somit 35 (blau) und der untere 29 (rot). Abb. 36 zeigt, dass man auf diesem Wege 67 Einsen für die Antwort erhält.

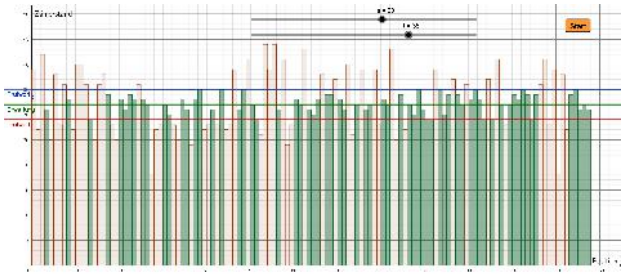


Abb. 36: Die Balken, deren Höhe zwischen einem oberen und einem unteren Prüfwert liegt, liefern die Positionen für die Einsen in der Antwort.

Das sind drei Einsen mehr als die angepeilten 64 Einsen für die Antwort. Das wäre nicht erheblich, aber da die Antwort als Iterierte wieder als nächste Frage gestellt wird, erhält man einen anderen Erwartungswert als vorher, an den man sich anpassen muss. Und überhaupt streut der Erwartungswert in der Praxis oft so sehr, dass das Sigma–Verfahren nicht genügend Einsen liefert und die Iteriertenbahn wegen Nullaktivität endet. Der technische Vorteil des Sigma–Verfahrens, nur vergleichweise wenige Zählerstände abprüfen zu müssen und nicht wie bei MINA ständig überwachen zu müssen, ob man schon genügend Einsen beisammen hat, wird durch die Vorkehrungen wieder zunichte gemacht, eine drohende Nullaktivität zu verhindern.

Folglich wurde das *Sigma*–Verfahren dahingehend erweitert, dass der obere Prüfwert (blaue Waagerechte) so lange angehoben wird, bis genügend Einsen erzielt sind. Das geänderte Verfahren wird mit einem Sternchen markiert und heißt also *Sigma**–Verfahren.

Um zu erläutern, welchen Vorteil *Sigma** dennoch verspricht, wechseln wir die Darstellungsform. Nunmehr wird die Verteilung der Häufigkeit der Zählerstände aufgezeichnet. Anders gesagt, es wird nun gezählt, welche Balkenhöhe wie oft vorkommt. Das wird wie in Abb. 37 in einem Häufigkeitsdiagramm dargestellt.

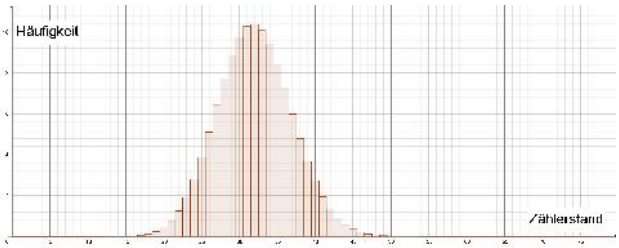


Abb. 37: Die Anzeige der Häufigkeitsverteilung der Zählerstände fördert Ideen für weitere Binarisierungsverfahren.

Der Vorteil dieser Darstellungsform ist, dass man sogleich sehen kann, welcher Zählerstand besonders häufig zu erwarten ist und daher besonders viele Einsen zur Antwort beisteuert, wenn ihn das Binarisierungsverfahren dafür auswählt.

Damit lässt sich für *Sigma** verdeutlichen, dass wenn dieses Binarisierungsverfahren mit der Auswahl seiner Zählerstände ein Stück vor dem häufigsten Zählerstand beginnt, es beim Einsammeln gegen Ende Zählerstände erreicht, die nicht so viele Einsen liefern. Dadurch gelangen im Mittel nicht so viele überzählige Einsen in die Antwort. Im in Abb. 38 gezeigten Fall wären es nur zwei.

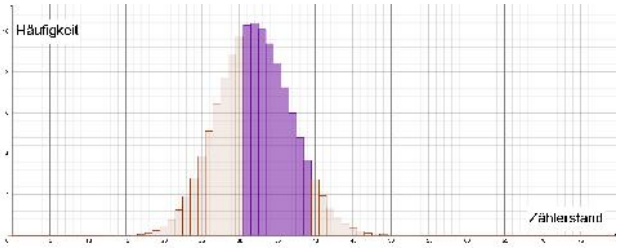


Abb. 38: Von links nach rechts: Das *Sigma**–Verfahren sammelt zum Schluss nur noch wenige Einsen ein.

Wenn die Balken der häufigsten Zählerstände wegen der überzähligen Einsen in den Iterierten weiter nach rechts streuen, passt sich das Verfahren dem an.

Das Binarisierungsverfahren *MEAN* erhielt seinen Namen, weil zunächst der Mittelwert aller Zählerstände berechnet wird.[38] [39]

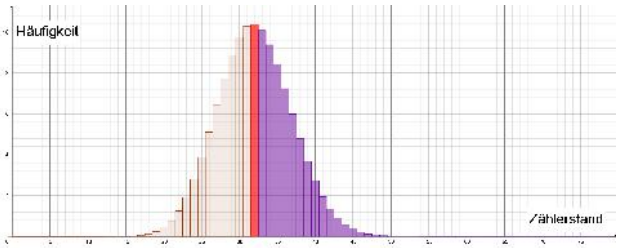


Abb. 39: *MEAN* sammelt Einsen rechts vom Mittelwert.

Als *MINA*–Variante vermeidet es *MILO*,[40] diejenigen Einsen einzusammeln, die zum letzten *MINA*–Schritt gehören. Der letzte *MINA*–Schritt wird oft einen der Balken mit den häufigsten Zählerständen auswerten, die in der Mitte der Verteilung liegen, wodurch zu erwarten ist, dass damit auch viele überzählige Einsen geliefert werden. Daher macht das *MILO*–Verfahren den letzten *MINA*–Schritt rückgängig und beginnt stattdessen die Balken von der anderen Seite her, in Abb. 40 also von links her, zu verarbeiten.

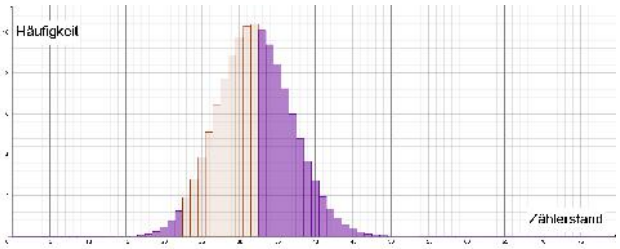


Abb. 40: *MILO* beginnt wie *MINA* von rechts her, macht den letzten Schritt aber rückgängig und setzt dann von links fort.

Als letztes hier zu beschreibendes Verfahren setzt das SILO–Binarisierungsverfahren bei einem Schwachpunkt des ansonsten effektiveren MILO–Verfahrens an, nämlich dem Rückwärtsschritt. Dieser kostet Zeit und Schaltungsaufwand. Daher hört SILO mit dem Erfassen der Einsen schon in einem Abstand von gut einer Standardabweichung[41] vor dem Erwartungswert der Verteilung auf und beginnt also eher mit der Auswertung der kleineren Balken von der anderen Seite her.

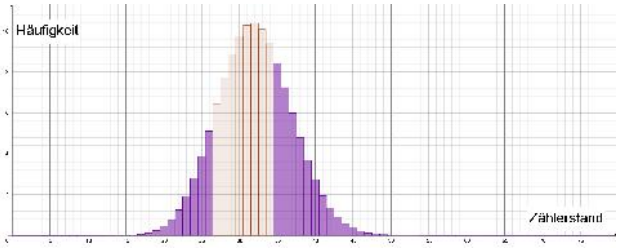


Abb. 41: *SILO* hört im Unterschied zu *MILO* von rechts kommend eher auf und setzt dann ebenfalls von links fort.

Alle genannten Binarisierungsverfahren weisen Vor- und Nachteile auf. Bei *LSB* stehen die Vorteile der im Mittel nicht zu erwartenden überzähligen Einsen und der raschen Durchführbarkeit, dem Nachteil einer eventuellen Nullaktivität gegenüber (s. S. 5). *MINA* punktet mit seiner algorithmischen Klarheit, liefert aber im Vergleich zu den anderen Verfahren die meisten überzähligen Einsen. Außerdem erlaubt der Umstand, dass *MINA* immer nur die Einsen der größten Zählerstände einsammelt, eine Rekonstruktion der Matrix, wie im nächsten Kapitel „Statistische Untersuchungen“ noch erläutert werden wird. *Sigma** liefert zwar weniger überzählige Einsen, leidet in der Praxis bei kleinen Matrizen aber unter der Streuung der häufigsten Zählerstände. *MEAN* erfordert durch das Berechnen des Mittelwerts im Vergleich vermehrt Zeit und Schaltungsaufwand. Der aufwendige Rückwärtsschritt von *MILO* wird bei *SILO* vermieden, doch handelt man sich damit in der Praxis auch wieder einige überzählige Einsen ein.

Die Anzahl an zu erwartenden überzähligen Einsen wurde für alle Verfahren experimentell ermittelt. Dazu wurden wiederholt Iteriertenbahnen der Länge 10.000 dahingehend ausgewertet, mit welcher relativen Häufigkeit h Einsen in den Iterierten zu beobachten sind:

	MINA	LSB	Sigma*	MEAN	MILO	SILO
h	0,571	0,500	0,558	0,429	0,528	0,543

Bezogen auf eine 128×128–Matrix bedeutet das an Anzahlen k überzähliger Einsen gerundet:

	MINA	LSB	Sigma*	MEAN	MILO	SILO
k	9,1	0	7,4	-9,1	3,6	5,5

Der elektronische Baustein in unseren Geräten, der `plumChip`, kann fast alle dieser Binarisierungsverfahren ausführen. Da sich die Verfahren auch kombinieren lassen, zum Beispiel einmal *MILO* mit einmal *LSB* oder zweimal *MINA* und so fort, eröffnet sich dadurch eine Vielfalt an möglichen Iteriertenbahnen, die nicht nur bei der Datenübertragung oder Datenablage zum Tragen kommt, sondern die auch ins Gold–Silber–Bronze–Schlüsselkonzept eingebunden wurde.[42]

Anmerkungen

[31] s. *LSB* und $\overline{LSB}$ auf S. 2 und *MAX* auf S. 4

[32] Zum Vergleich: Die Anzahl der Atome im Weltall wird auf 10^{84} bis 10^{89} geschätzt, seine sichtbare Masse auf 10^{53} kg.

[33] Die Abbildungen zeigen `GeoGebra`–Apps, die mit `GeoGebra` Classic 5.0.624.0-d erstellt wurden.

[34] Der Begriff Schwellwert ist der biologischen Vorstellung entnommen, dass eine Nervenzelle ab eines gewissen Schwellwerts zündet.

[35] *MINA* — Mindestaktivierung

[36] s. S. 2

[37] Bei der Namensgebung bezog man sich auf die übliche Wahl des griechischen Buchstabens σ (Sigma) für die Bezeichnung der Standardabweichung. σ ist ein Maß für die Streuung von Werten, hier für die Streuung der Zählerstände.

[38] *engl.* mean — das Mittel, der arithmetische Mittelwert

[39] Von Mari Miyamoto stammt der Hinweis, dass das nicht der arithmetische Mittelwert sein muss, sondern auch ein anderes Lagemaß sein kann. In einer Variante des *MEAN*–Verfahrens werden die Matrixspalten mehr als zur Hälfte starr befüllt. Dadurch wird die Wahrscheinlichkeit (ein wenig) größer, dass Einsen in den Zähler fallen.

[40] Die Bezeichnung *MILO* leitet sich von *MINA* und dem Vornamen Oli(ver) ab, der wegen des typischen Rückwärtsschritts von *MILO* hier folglich rückwärts geschrieben wurde.

[41] Wegen der Nutzung der bei dieser Häufigkeitsverteilung zu erwartenden Standardabweichung heißt das Verfahren SILO.

[42] s. Kapitel „Iteriertenbahnen“, S. 6

Statistische Untersuchungen

Das Betrachten eines Iteriertenteppichs wie in den Abb. 8, 9 oder 21 oder die Überlegungen zu den Vektographen im Kapitel „Iteriertenbahnen“ legen die Annahme nahe, dass sich die Iteriertenbahnen zum informationstechnisch sicheren, verschlüsselten Ablegen und Übertragen von Datenblöcken eignen, da sich die Reihenfolge der Einsen und Nullen in ihnen wie zufällig zu ergeben scheint. Das ist zu überprüfen. Dazu werden im Folgenden eine Reihe von Tests vorgestellt, die durchgeführt wurden und deren Ergebnisse die Auswahl eines geeigneten Verfahrens oder einer Kombination von Binarisierungsverfahren erleichtern.

Die Iterierten einer Iteriertenfolge (Iteriertenbahn) dienen zum Ver- und Entschlüsseln der jeweils vorliegenden Datenblöcke, weswegen es ausreichend schwierig sein soll, aus vorherigen Iterierten die kommenden Iterierten auch nur in Teilen vorausbestimmen zu können. Dazu werden die Iteriertenteppiche auf wiederkehrende Muster und andere Auffälligkeiten untersucht („Webfehler“, „Laufmaschen“).

Auffällig sind beispielsweise in Abb. 42 die Laufmaschen von einem MILO–Teppich. Sie können sicherheitstechnisch unerwünschte Effekte nach sich ziehen.

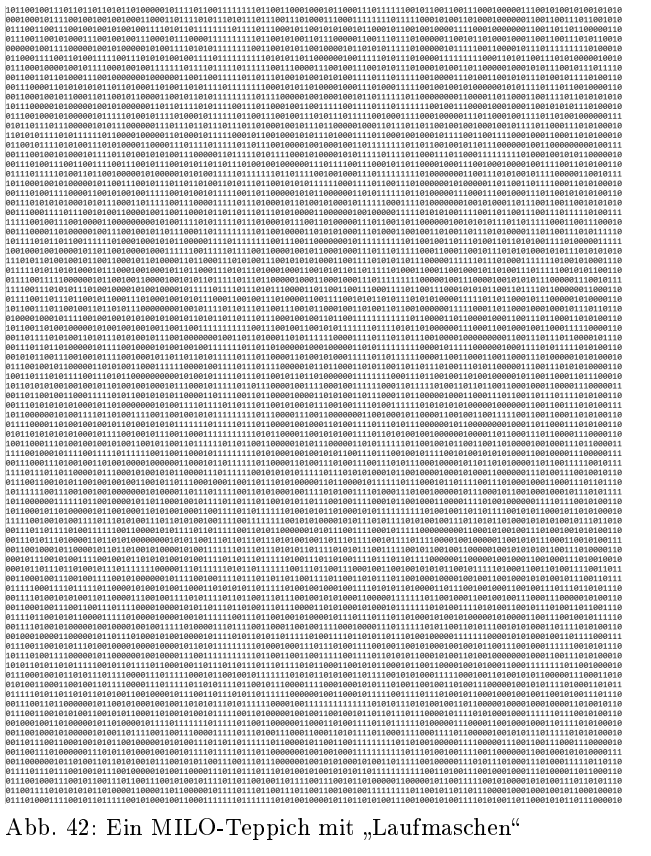


Abb. 42: Ein MILO-Teppich mit „Laufmaschen“

Aus der Bildunterschrift zur Abb. 22 lässt sich der Verdacht ableiten, dass diese senkrechten Streifen durch zeilenstarre Matrizen verursacht werden. Zur Erläuterung schauen wir uns das schon bekannte Beispiel aus Abb. 12 noch einmal genauer an und Verfolgen, was beim Abfragen solcher Spalten geschieht, die weniger stark mit Einsen belegt sind.

Die zeilenstarre Matrix

$$\begin{pmatrix} 1&1&0&0&0&1&1&0 \\ 0&0&0&1&0&1&1&1 \\ 0&1&1&0&0&1&0&1 \\ 0&1&0&1&1&0&0&1 \\ 1&0&0&1&1&0&0&1 \\ 0&0&0&1&1&1&0&1 \\ 0&1&0&1&0&0&1&1 \\ 1&1&0&1&0&1&0&0 \end{pmatrix}$$

erzeugt mit f_0 als Startfrage die Iteriertenbahn

$$\begin{aligned} f_0 &= \begin{pmatrix} 1&0&0&0&0&0&0&1 \end{pmatrix} \\ f_1 &= \begin{pmatrix} 1&1&0&0&0&1&0&0 \end{pmatrix} \\ f_2 &= \begin{pmatrix} 0&0&0&0&0&1&0&0 \end{pmatrix} \\ f_3 &= \begin{pmatrix} 0&0&0&1&1&1&0&1 \end{pmatrix} \\ f_4 &= \begin{pmatrix} 0&0&0&1&0&0&0&0 \end{pmatrix} \\ f_5 &= \begin{pmatrix} 0&1&0&1&1&0&0&1 \end{pmatrix} \\ f_6 &= f_4 \end{aligned}$$

und lässt in deren dritter und siebter Spalte nur Nullen erscheinen. Und man beobachtet in der Berechnung, dass die Spalten 3 und 7 der Matrix auch nie zur Antwort beitragen. Die Wahrscheinlichkeit, dass diese Spalten eine Eins in den Zähler liefern, ist geringer, da sie selbst aus weniger Einsen bestehen. Verschiebt man in jeder Zeile der Matrix die Einsen so, dass auch jede Spalte vier Einsen bekommt (doppeltstarr), verschwinden die Laufmaschen.[43]

Die doppeltstarre Matrix

$$\begin{pmatrix} 1&1&0&0&0&1&1&0 \\ 0&0&0&1&0&1&1&1 \\ 1&1&0&0&1&1&0&0 \\ 0&1&1&0&1&0&0&1 \\ 1&0&0&1&1&0&1&0 \\ 0&0&1&1&1&0&0&1 \\ 0&1&1&0&0&1&0&1 \\ 1&0&1&1&0&0&1&0 \end{pmatrix}$$

erzeugt die Iteriertenbahn:

$$\begin{aligned} f_0 &= \begin{pmatrix} 1&0&0&0&0&0&0&1 \end{pmatrix} \\ f_1 &= \begin{pmatrix} 1&0&0&0&0&0&1&0 \end{pmatrix} \\ f_2 &= \begin{pmatrix} 0&1&0&0&0&1&0&0 \end{pmatrix} \\ f_3 &= \begin{pmatrix} 0&0&0&1&0&0&0&1 \end{pmatrix} \\ f_4 &= \begin{pmatrix} 0&0&1&0&0&0&0&0 \end{pmatrix} \\ f_5 &= \begin{pmatrix} 1&1&0&0&1&1&0&0 \end{pmatrix} \\ f_6 &= \begin{pmatrix} 0&0&0&1&0&0&1&0 \end{pmatrix} \\ f_7 &= \begin{pmatrix} 0&1&1&0&0&0&0&1 \end{pmatrix} \\ f_8 &= \begin{pmatrix} 1&0&0&1&0&1&1&0 \end{pmatrix} \\ f_9 &= f_7 \end{aligned}$$

Das Beseitigen von Laufmaschen, die übrigens auch aus Einsen bestehen können, gelingt nicht immer so einfach, aber dieses Beispiel sei ein Beleg für den Nutzen spalten- und doppeltstarrer Matrizen.

Die Spaltensummen des ersten obigen Iteriertenteppichs lauten $\begin{pmatrix} 2&2&0&3&2&2&0&3 \end{pmatrix}$, die des zweiten $\begin{pmatrix} 4&3&2&3&1&3&3&3 \end{pmatrix}$. Auf die Beobachtung der Spaltensummen der Iteriertenteppiche greifen einige der unten beschriebenen Testverfahren zurück, indem mit den Mitteln der Statistik geprüft wird, ob die Spalten des Teppichs sich wie zufällig nebeneinander entwickeln und ihre Summen wie zu erwarten streuen.

Doch allein auf die Analyse der Spaltensummen kann man sich nicht verlassen, denn damit würde man zum

Beispiel zyklische Phänomene („Fixkreise“ im Vektographen), also sich ständig wiederholende Iterierte nicht entdecken.

Die Teppiche wären außer auf auffällige Spalten auch auf andere Vorhersagbarkeiten hin zu untersuchen, zum Beispiel auf Schrägstreifen, Parkette, wiederkehrende Muster. Daher wurden folgende Tests vorgeschlagen:

- Spaltensummentest: Wirkt jede Spalte wie von einem binären (Pseudo-) Zufallsgenerator erzeugt?
- Zeilentests: Streuen die Zeilensummen wie zu erwarten? Was liefert ein Run–Test zur Verteilung der Nullen und Einsen in den Iterierten?
- Periodizitätstest: Ab wann wiederholen sich die Iterierten?
- Schrägstreifentest: Tauchen im Teppich diagonale Muster auf?
- Dichtetest: Verhalten sich die Spaltensummenhäufigkeiten bei zufällig herausgegriffenen, rechteckigen „Flicken“ genauso wie beim ganzen Teppich?
- Flickendistanztest: Streut der Ähnlichkeitsabstand (Hamming–Distanz) zwischen je zwei zufällig herausgegriffenen Flicken im zu erwartenden Ausmaß?
- Zeilen-/Spaltendistanztest: Streut der Ähnlichkeitsabstand zwischen je zwei zufällig herausgegriffenen Zeilen oder Spalten ausreichend deutlich?
- Abhängigkeitstest: Lässt sich eine Iterierte immer in gleicher Weise als Linearkombination ihrer Vorgänger darstellen?

Spaltensummentest

Die zahlreichen Datensätze und Diagramme von vielen Versuchen auf Großrechnern (siehe Abb. 43 und 44)[44] und die hunderte von Experimenten mit dem `plumChip` lassen vermuten, dass die Spaltensummen bei von spaltenstarren Matrizen erzeugten Iterierten-teppichen binomialverteilt sind.

Abb. 43: Häufigkeitsverteilung der Spaltensummen bei einer spaltenstarren 1000×1000–Matrizen

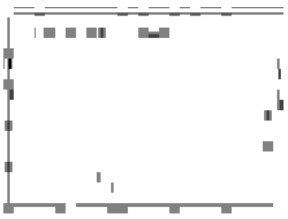


Abb. 44: Häufigkeitsverteilung der Spaltensummen bei einer zeilenstarren 1000×1000–Matrizen

Die Unterschiede in der Gestalt der Graphen der vorstehenden Abbildungen erklären sich wiederum durch den Einsatz einer spaltenstarren (Abb. 43) im Unterschied zu einer zeilenstarren Matrix (Abb. 44).

Im Folgenden werden nur Iteriertenfolgen von spaltenstarren 128×128–Matrizen untersucht.

Die Annahme, die Spaltensummen eines Teppichs seien binomialverteilt, wird durch χ^2–Tests für unterschiedlich erzeugte Teppiche überprüft. Sollte man diese Annahme mit einer Irrtumswahrscheinlichkeit von 5 Prozent nicht ablehnen können, dann dient das als Beleg dafür, den Eintrag an jeder Position in der Iterierten als Ergebnis eines BERNOULLI–Experiments auffassen zu können.[45]

Für eine 128×128–Matrix erbrachten die Verfahren *MINA*, *LSB*, *Sigma**, *MEAN*, *MILO* und *SILO* bei $i = 5.000$ Iterierten und nach Einteilung in $m = 10$ Klassen und mit den schon auf S. 12 genannten relativen Häufigkeiten h folgende Prüfwerte χ^2:

	MINA	*LSB*	*σ**	*MEAN*	*MILO*	*SILO*
h	0,571	0,500	0,558	0,429	0,528	0,543
χ^2	45,445	8,435	4,005	30,723	6,290	6,387

Der berechnete Prüfwert χ^2 muss kleiner als der kritische Prüfwert $\chi^2_{(0,95;\,9)} = 16,92$ sein, den man einer Tabelle entnimmt.[46] In diesem Experiment wird die Annahme, die Spaltensummen seien binomialverteilt (Nullhypothese), folglich mit 95–prozentiger Gewissheit für *LSB*, *Sigma**, *MILO* und *SILO* bestätigt.

Zeilentest

Die Anzahl an Einsen pro Iterierter sollte erwartungsgemäß streuen. Bei einem *MILO*–Teppich mit 5.000 Iterierten wurden die in Abb. 45 veranschaulichten Häufigkeiten gezählt.

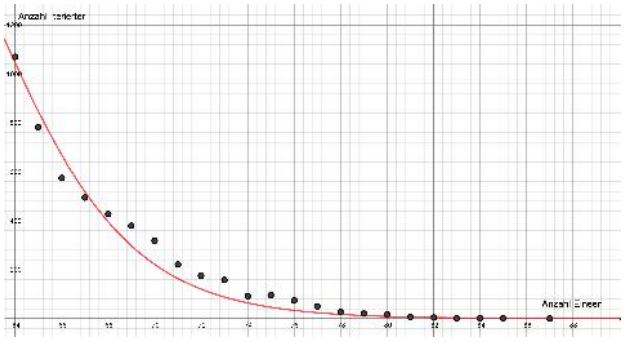


Abb. 45: Die Verteilung von 5.000 MILO–Zeilensummen bei einer 128×128–Matrix

Hier haben 1.070 von 5.000 *MILO*–Iterierten genau 64 Einsen, 782 besitzen 65 Einsen, 574 enthalten 66 Einsen und so fort. Die Verteilung der Zeilensummen (siehe Abb. 45) lässt entfernt an eine logistische Abnahme denken (roter Graph), was aber bisher nicht weiter untersucht wurde.

Stellt man die Zeilensummen–Verteilungen von *MINA*, *Sigma**, *MILO* und *SILO* gegenüber (s. Abb. 46), erkennt man die Vorteile von *MILO* (lila Graph) hinsichtlich des Bestrebens, in jeder Iterierten möglichst gleich viele Nullen und Einsen zu erhalten. Der Graph ist positiv gekrümmt und fällt von über 1.000 Zeilen mit der idealen Einsenanzahl 64 kommend anfangs mit größerer Steigung als die Graphen der anderen Binarisierungsverfahren.

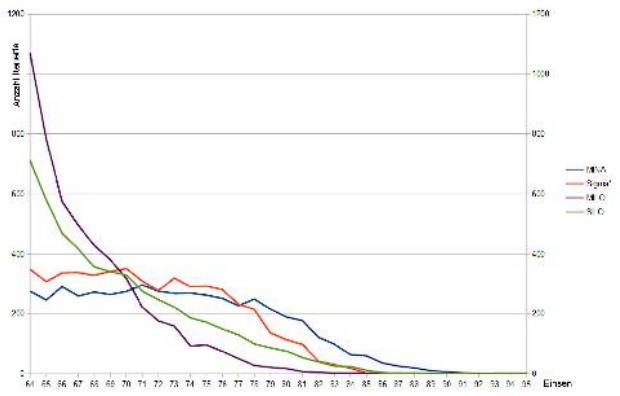


Abb. 46: Die Verteilung von 5.000 Zeilensummen bei einer 128×128–Matrix

Das *LSB*–Verfahren liefert Anzahlen an Einsen, die auch kleiner als die Hälfte der Matrixgröße sind, hier also kleiner als 64. Der Abb. 47 entnehme man die zugehörige typische Zeilensummenverteilung.

Abb. 47: Die Verteilung von 5.000 *LSB*–Zeilensummen bei einer 128×128–Matrix

Das *MEAN*–Verfahren liefert eine ähnliche Verteilung wie das *LSB*–Verfahren, jedoch zu kleineren Zeilensummen hin verschoben.

Run–Test

Als weiterer Test für die zufällige Verteilung der Einsen in den Iterierten steht der *Run–Test* (Wald–Wolfowitz–Test) zur Verfügung. Dazu werden alle Iterierten hintereinander gesetzt und die *Runs* gezählt, also die Anzahl an Eins– und Null–Läufen.

Zur Erläuterung des Vorgehens bilden wir beispielgebend bei einer 32×32–Matrix mit dem *MINA*–Verfahren die ersten beiden Iterierten und zählen anschließend die Runs.

```
Iterierte
10111101010111100100100101100111 [19]
11101101101101111000110111111011 [23]

Lauflängenverteilung
1:17 2:10 3:1 4:3 5:0 6:2 7:0 8:0 9:0 10:0 11:0
12:0 13:0 14:0 15:0 16:0 17:0 18:0 19:0 20:0
Anzahl Runs: 33
Anzahl Einsen: 42
Anzahl Nullen: 22
```

Dem oben gezeigten Protokoll entnimmt man, dass 17 Eins– und Null–Läufe der Länge 1, zehn Runs der Länge 2, nur ein Lauf der Länge 3, drei Läufe der Länge 4 und zwei Läufe der Länge 6 aufgetreten sind.

Von den 64 Einträgen in den Iterierten waren 42 Einsen und 22 Nullen. Dieses Ergebnis verträgt sich mit der Eigenschaft von *MINA*, dass es immer deutlich mehr Einsen als Nullen erzeugt.

Wenn der Run–Test davon ausgeht, dass die Wahrscheinlichkeit für eine Eins und für eine Null gleich groß ist, dann dürfte das *LSB*–Verfahren den Run–Test besonders gut bestehen. Eine Reihe von Experimenten mit $i = 10.000$ Iterierten bei einer 128×128–Matrix, die nachstehend tabellarisch erfasst sind, bestätigt dieses. Als Prüfwerte z für den Runtest ergaben sich mit h als relativer Häufigkeit für eine Eins:[47]

	MINA	*LSB*	*Sigma**	*MEAN*	*MILO*	*SILO*
h	0,571	0,500	0,558	0,429	0,528	0,543
z	5,388	0,188	0,319	6,371	5,648	2,239
99%	×	✓	✓	×	×	✓
95%	×	✓	✓	×	×	×
90%	×	✓	✓	×	×	×

Es fällt auf, dass der Prüfwert z bei denjenigen Verfahren kleiner ist, die relativ viele derjenigen Einsen einsammeln, die zu kleineren Zählerständen gehören (*SILO*, *Sigma**, *LSB*). Der Run–Test gilt als bestanden, wenn der Prüfwert z bei einem Prozent erlaubten Irrtums kleiner als 2,58 ist, bei 5 Prozent kleiner als 1,96 und bei 10 Prozent Irrtum kleiner als 1,64.

Es ist anzunehmen, dass der Run–Test für den Iteriertenteppich die gleichen Ergebnisse liefert, wenn man ihn wie vorstehend nicht zeilenweise, sondern stattdessen spaltenweise untersucht, da sich die Anzahlen überzähliger Einsen, die die Verfahren kennzeichnen, in den Spalten genauso auswirken. Das ist noch genauer zu überprüfen.

Die Verteilung der Lauflängen ist in Abb. 48 dargestellt. An der Hochachse wurden die Lauflängen, an der Rechtsachse deren Häufigkeit bei dem jeweiligen Verfahren abgetragen. Dem Augenschein nach unterscheiden sich die Verteilungen kaum, rechnerisch dann aber schon.

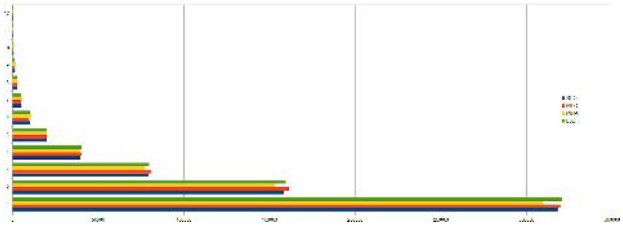


Abb. 48: Verteilung der Runs bei *SILO* (blau), *MILO* (rot), *MINA* (gelb) und *LSB* (grün)

Periodizitätstest

Eine Untersuchung von Iteriertenbahnen hinsichtlich ihrer Periodenlänge suchte mit großem Rechenaufwand nach dem Abfrageschritt, ab der sich Iterierte und die Iteriertenfolge wiederholen.[48] Schon bei einer vergleichweise kleinen Matrix, wie beim Experiment vom 02.09.2019, dessen Ergebnisse in Abb. 49 gezeigt werden, wiederholten sich Abfragen erst ab etwa 500 MINA–Schritten.

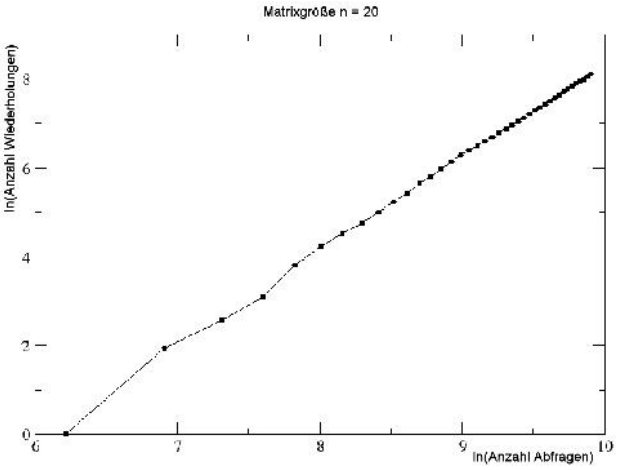


Abb. 49: Anzahl an Wiederholungen bei *MINA*–Iterierten (doppeltlogarithmisch dargestellt)

Diese Grafik gab Anlass genauer nach den Periodenlängen in den Iteriertenfolgen zu suchen. In einem Experiment mit einer schwächer belegten Matrix (20 % Einsen) vom 08.10.2019 ergaben sich folgende Werte:

Matrixgröße	Periodenlänge
10	5
20	8
30	13
40	62
50	96
60	277
70	207
80	125
90	2.912
100	2.018
110	26.457
120	52.259
130	138.760
140	410.381
150	1.022.355
200	größer 2.000.000
250	größer 5.000.000

Aus dem Umstand, dass auch nach zweiwöchigem Programmlauf auf einem Großrechner schon für eine 200×200–Matrix keine Wiederholungen, kein Zyklus in der Iteriertenbahn auftauchte, wurde gefolgert, dass in der Praxis bei ausreichend großer Matrix das *MINA*–Verfahren keine unerwünscht häufigen Wiederholungen liefert. Für die anderen Binarisierungsverfahren wird das von uns noch detaillierter untersucht.

Schrägstreifentest

Der Schrägstreifentest soll zum Aufspüren von schwach oder stark mit Einsen besetzten Diagonalen im Iteriertenteppich dienen. Dazu lässt man die Zeilen des Teppichs immer um eine Spalte weiter rotieren und wendet dann den oben beschriebenen Spaltensummentest an. Die auftretenden X^2–Werte sollten dabei wegen der in einem solchen Fall großen Abweichung zwischen Beobachtung und Erwartung auf etwaige Streifen wie in den Abb. 22 und 42 hinweisen.

Der Ablauf des Schrägstreifentests sei am Beispiel einer 16×16–Matrix erläutert. Nachstehend ist zu erkennen, wie die Zeilen des Teppichs von Mal zu Mal um eine Position weiter verschoben werden. Die Einträge, die dabei links „herausfallen“ werden von rechts wieder hinzugefügt.

```
1010111011001011    1010111011001011    1010111011001011
0000111100111101    0001111001111010    0011110011110100
1111111111101111    1111111110111111    1111111011111111
0101110101111101    1110101111101010    0101111101010111
0111111101000110    1111010001100111    0100011001111111
1111110110111111    1011011111111111    1111111111110110
1110111101000111    1101000111111011    0111111011110100
0101110110111111    1101111110101110    1101011101101111
1110111111000111    1100011111101111    1110111111000111
0001100110111001    0111001000110011    0110011011100100
1111011101001111    0011111111011101    0111010011111111
0101110101111111    1111101011101011    0101111111010111
1111111111100110    0110111111111110    1110011011111111
1111000110110101    1011111000110110    1101011111000110
1011010011000111    1110110100110001    0111101101001100
1010100111111101    1101010011111110    0110101001111111
Chi^2 = 6.30362     Chi^2 = 3.54635     Chi^2 = 18.84591
```

Die Zeilen des Iteriertenteppichs werden um 1, 2, 3, 4,... Positionen nach links rotiert. Danach wird mit dem X^2–Test auf Binomialverteilung der Spaltensummen bei weniger als 5 % Irrtum nach auffälligen Spalten gesucht.

Ein solcher Test wurde für 128×128–Matrizen durchgeführt und lieferte, aufgefächert nach den verschiedenen Verfahren, folgende Beobachtungen ($\chi^2_{(0{,}95;\,9)} = 16{,}92$):

Verfahren	X^2–Test bestanden
MINA	119 von 127
LSB	123 von 127
*Sigma**	120 von 127
MEAN	120 von 127
MILO	124 von 127
SILO	121 von 127

Es wurden bei keinem der Verfahren Werte sichtbar, die auf Diagonalen im Iteriertenteppich hinwiesen.

Dichtetest

Der Dichtetest durch Untersuchung von zufällig gewählten, nicht zu kleinen rechteckigen „Teppichflicken“ auf Spaltensummenhäufigkeit — wie beim eingangs vorgestellten Spaltensummentest — sollte stets binomialverteilte Spaltensummen aufweisen. Dazu wurden beispielsweise für 128×128–Matrizen mit jedem der sechs vorliegenden Binarisierungsverfahren Teppiche aus je 1.000 Iterierten gebildet. Danach bestimmte ein Zufallsgenerator[49] Ort und Größe rechteckiger Flicken, für die der Spaltensummentest (X^2–Test) ausgeführt wurde.

Folgende Übersicht zeigt die sich dabei typischerweise ergebenden Testergebnisse bei Auswahl von je 10.000 zu testenden Flicken.

Verfahren	Anzahl bestandener Tests
MINA	96,66 %
LSB	96,80 %
*Sigma**	98,79 %
MEAN	87,66 %
MILO	98,22 %
SILO	98,09 %

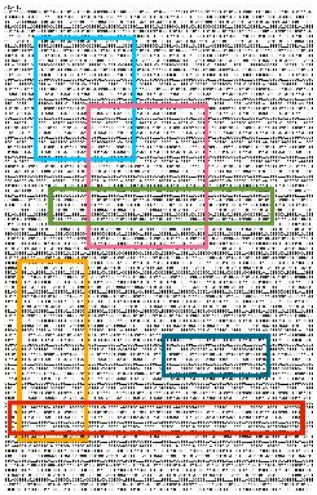


Abb. 50: Es werden zufällig rechteckige Flicken aus dem Teppich ausgewählt und untersucht.

Dieses Ergebnis ist ein Beleg dafür, dass der Iteriertenteppich bei nahezu allen Verfahren überall in etwa die gleiche Dichte besitzt, also sich die Einsen in allen Bereichen mit der gleichen Wahrscheinlichkeit auffinden lassen.

Flickendistanztest

Das Herausgreifen eines beliebigen Teppichflickens wie in Abb. 50 und die Messung des Ähnlichkeitsabstands (Hammingdistanz) zu anderen zufällig herausgegriffenen, gleich großen Flicken des Teppichs sollte im Mittel um eine Distanz von 50 % der Flickengröße streuen. Bei 128×128–Matrizen wurden aus Iteriertenteppichen bestehend aus 1.000 Iterierten jeweils 10.000 gleich große Flickenpaare zufällig bestimmt und ihre Hammingdistanz berechnet.

Verfahren	Spannweite	mittlere Hammingdistanz
MINA	<49,189 %; 52,752 %>	51,08 %
LSB	<48,431 %; 51,623 %>	50,00 %
*Sigma**	<49,286 %; 52,871 %>	50,76 %
MEAN	<48,979 %; 53,158 %>	50,99 %
MILO	<48,633 %; 52,133 %>	50,17 %
SILO	<48,824 %; 52,129 %>	50,37 %

Wie es erwartet wurde, streuen die Abstände der Flicken aller Verfahren um 50 % ihrer Größe. Das bedeutet, dass sich im Mittel je zwei beliebige, gleich große Flicken des Teppichs in der Hälfte ihrer Einträge unterscheiden.

Zeilen- und Spaltendistanztest

Analog zum Flickendistanztest müsste auch ein Zeilen- und Spaltendistanztest als Sonderfall des Flickendistanztests ein Ergebnis ähnlich zu diesem ergeben.

Verfahren	Spannweite	mittlere Hammingdistanz
MINA	<39,100 %; 62,900 %>	51,20 %
LSB	<44,900 %; 55,200 %>	49,98 %
*Sigma**	<45,300 %; 56,600 %>	50,67 %
MEAN	<37,200 %; 62,800 %>	51,14 %
MILO	<43,600 %; 57,400 %>	49,94 %
SILO	<42,600 %; 59,000 %>	50,26 %

Ein Spaltendistanztest liefert dann auch tatsächlich stets in etwa vorstehende Werte bei 10.000 überprüften Spaltenpaaren, einem Teppich aus 1.000 Iterierten und einer 128×128–Matrix.

Der dementsprechende Zeilendistanztest ergibt immer ungefähr folgende Ähnlichkeitsabstände:

Verfahren	Spannweite	mittlere Hammingdistanz
MINA	<31,250 %; 71,094 %>	51,12 %
LSB	<34,375 %; 66,406 %>	50,00 %
*Sigma**	<34,375 %; 70,313 %>	50,74 %
MEAN	<29,688 %; 71,875 %>	51,00 %
MILO	<34,375 %; 67,969 %>	50,13 %
SILO	<33,594 %; 67,969 %>	50,42 %

Es lässt sich also beobachten, dass je zwei Zeilen oder Spalten des Iteriertenteppichs sich im Mittel zur Hälfte in ihren Einträgen unterscheiden. Sie stimmen wenigstens in ungefähr einem Drittel überein und höchstens in circa zwei Dritteln.

Sonstige Tests

Eine weitere Reihe von Tests und Testergebnissen für *MEAN* mit einer spaltenstarren 128×128–Matrix sind in Miyamoto [15. Dezember 2020] beschrieben. Dort werden bei jeweils gleichem Testverfahren (u.a. X^2–Test, Run–Test) die Ergebnisse von *MEAN* mit denjenigen des Pseudozufallsgenerators „Mersenne–Twister“ verglichen, gegen den sich *MEAN* nicht durchsetzen kann. Ein Test auf lineare Abhängigkeit der Teppichzeilen wurde in Dierks [20. Januar 2021] bereits angedacht.

Test zur Rekonstruktion der Matrix

Ein Verfahren, welches prüft, ob man mit Frage–Antwort–Paaren auf die jeweils eingesetzte, spaltenstarre Matrix zurückschließen kann, liegt vor und wurde auf einige der Binarisierungsverfahren angewandt.[50]

Es werden dazu alle Spalten einer $n \times n$–Matrix M und k Fragen f_i zufällig genau zur Hälfte mit Einsen gefüllt, die Versuchsobjekte also spaltenstarr vorbereitet. Zu den k Fragen ergeben sich vermöge M und einem der Binarisierungsverfahren die Antworten a_i mit $i = 1, ..., k$, die sich je nach gewähltem Verfahren *MINA*, *LSB*, *Sigma** unterscheiden.[51] Es wurde experimentell abgeklärt, ob mit Hilfe der Paare (f_i, a_i) die Matrix M rekonstruiert werden kann.

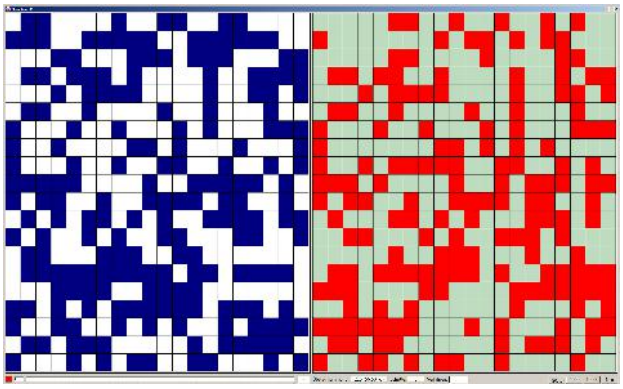


Abb. 51: *Rekonstruktionsversuch*: Links wird die Matrix M dargestellt, rechts in rot ihre (noch) nicht richtig ermittelten Einträge.

Dazu wird je ein Frage–Antwort–Paar (f_i, a_i) in eine leere $n \times n$–Matrix S_i gemäß der L–Lernregel (s. Bentz and Dierks [2013], S. 40) eingetragen. Die k Matrizen S_i werden anschließend zur *Summenmatrix* $S = \sum_{i=1}^{k} S_i$ aufsummiert. S wird weiter zur binären

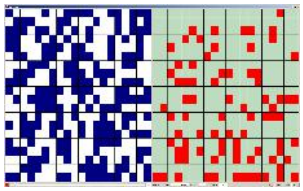
$n \times n$–*Medianmatrix* G umgeformt, indem in jeder ihrer n Spalten der Median m_j berechnet wird und alle Einträge g_{ij} auf 1 gesetzt werden, für die s_{ij} größer oder gleich m_j ist, und alle anderen auf 0:

$$g_{ij} = \begin{cases} 1, & \text{wenn } s_{ij} \geq m_j, \\ 0, & \text{sonst,} \end{cases}$$

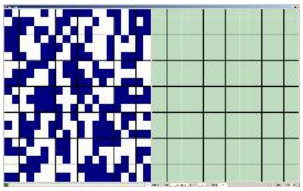
mit $i, j = 1, ..., n$.

Die aus der Summenmatrix S hervorgegangene Medianmatrix G wird mit der eingangs erzeugten Matrix M verglichen, um etwaige Übereinstimmungen aufzuzeigen.

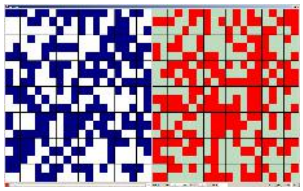
Je nach gewähltem Binarisierungsverfahren führt dieser Versuch zum Erfolg, also zur Rekonstruktion der Matrix. Die nachstehenden Abbildungen zeigen typische Ergebnisse am Beispiel einer 20×20–Matrix M. Die Matrix ist jeweils links angezeigt, rechts daneben sind in rot diejenigen ihrer Positionen markiert, an denen die Matrix mit der Medianmatrix G noch nicht übereinstimmt, M also noch nicht korrekt rekonstruiert wurde.



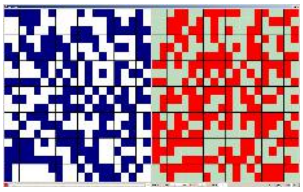
MINA: Nach 50 Schritten stimmen fast 80 % überein.



MINA: Nach 536 Schritten ist M rekonstruiert.



LSB: Nach 1001 Schritten stimmt etwa 51 % überein.



*Sigma**: Nach 1001 Schritten stimmt ca. 49 % überein.

Abb. 52: Die Binarisierungsverfahren *MINA*, *LSB* und *Sigma** reagieren auf den Versuch unterschiedlich.

Die Experimente zeigen, dass die Binarisierungsverfahren *LSB* und *Sigma* bezüglich eines solchen Rekonstruktionsversuches *robust* sind, das heißt, sie lassen keine Rückschlüsse auf die im `plumChip` arbeitende Matrix M zu. Bei *MINA* gilt das jedoch nicht (s. Abb. 52, obere Reihe).

Führt man zwei Binarisierungsverfahren hintereinander aus, es wurden *LSB* mit *MINA*, *MINA* mit *LSB* und auch *MINA* mit *MINA* kombiniert, bleiben die hier beschriebenen Rekonstruktionsversuche erfolglos.

Das Experiment setzt bei einem Einsatz in der Praxis voraus, dass man einen Zugriff auf geeignete Matrixoperationen des `plumChip`s besitzt, um die benötigten Frage–Antwortpaare (f_i, a_i) zu erhalten. Dazu gehört es, der Matrix eine beliebige Frage stellen und die damit von der Matrix assoziierte Antwort abrufen zu können. Doch der `plumChip` liefert bei jedem Aufruf immer nur den mit der nächsten Iterierten auf der gewählten Iteriertenbahn ver- oder entschlüsselten Datenblock zurück, aus dem man die Iterierte erst zu gewinnen hätte, was die eingangs auf S. 10 erwähnten Probleme bereitet. Zum `plumChip`, diesem zentralen Baustein unserer IT–Sicherheits–Geräte gibt das nächste Kapitel Auskunft.

Anmerkungen

[43] Ein Verfahren zur Erzeugung doppeltstarrer Matrizen wurde von Detlef Romberger implementiert (s. Romberger [18. November 2019]).

[44] Die Abbildungen wurden Braun [04. Januar 2019] entnommen, die Experimente von Jürgen Braun durchgeführt.

[45] Ein BERNOULLI–Experiment, benannt nach Jakob I Bernoulli (1655–1705), ist ein Zufallsexperiment, dass nur zwei Ausgänge hat. Von Bernoulli stammt der Satz: „Jede Wissenschaft bedarf der Mathematik, die Mathematik bedarf keiner.“

[46] vgl. „dtv–Atlas zur Mathematik, Band 2: Analysis und angewandte Mathematik“, S. 475, dtv 1978

[47] Ausführliche Beschreibungen der Berechnungen sind in Dierks [20. Januar 2021] zu finden.

[48] Diese Periodizitätstests wurden von Jürgen Braun für *MINA* implementiert und durchgeführt. Die Abb. 49 wurde seinen Ergebnissen entnommen.

[49] Es wurde zur zufälligen Auswahl von Position und Ausmaß der Flicken der Zufallsgenerator von *Free Pascal 3.0.0* benutzt.

[50] Dieser Test wurde von Elisabeth Ente entwickelt und beschrieben.

[51] Die Überprüfung der Binarisierungsverfahren *MEAN*, *MILO* und *SILO* mit diesem Test steht noch aus.

`plumChip` und `plumGeräte`

Der der Verarbeitung der Datenblöcke dienende, zentrale Baustein der `aPlum`–Sicherheitshardware[52] befindet sich derzeit auf einer Entwicklungsplatine[53] in Form eines FPGA[54] (s. Abb. 53) und wird von drei Prozessoren bedient.[55] Die Prozessoren sind über serielle Schnittstellen mit dem Baustein verbunden. Im Folgenden wird er `plumChip` genannt.

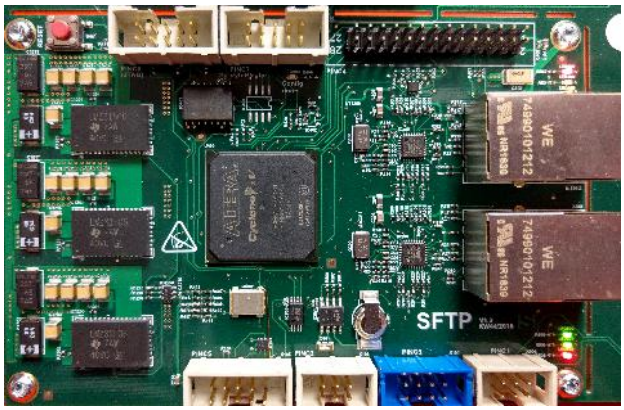


Abb. 53: Teil des SFTP–Boards mit dem FPGA in der Mitte

Vollen Zugriff auf den `plumChip` hat nur der Prozessor 3, der als `Beagle 3` bezeichnet wird (s. Abb. 54).[56] Die anderen beiden Prozessoren `Beagle 1` und `Beagle 2` dienen der Abschirmung des `Beagle 3` von der Außenwelt, indem sie die notwendigen Ein- und Ausgaben der Daten abwickeln. Die drei Prozessoren sind **nicht** direkt miteinander verbunden, sondern tauschen die nötigen Informationen über die Kommunikationsbefehle des `plumChips` untereinander aus.

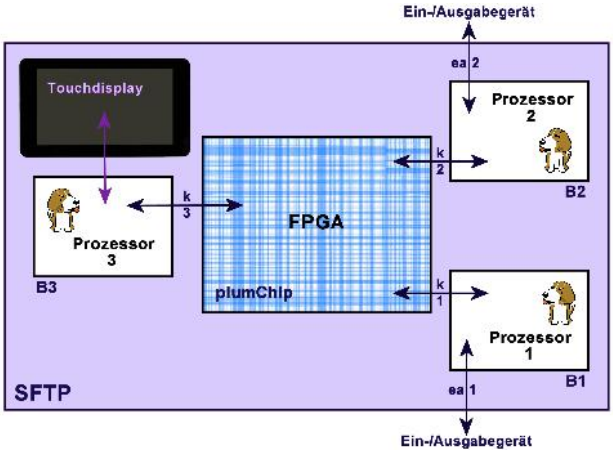


Abb. 54: Drei Prozessoren sind mit dem `plumChip` verbunden.

Für die drei Prozessoren wurden Programme entwickelt, die den Betrieb der `aPlum` für die gewünschten Anwendungsfälle ermöglichen. Der dafür zur Verfügung stehende Befehlssatz des `plumChips` beinhaltet die Befehle für die Matrixoperationen (löschen, starr befüllen), für das Ein- und Ausgeben von Datenblöcken, das Eintragen einer Saat in den Zufallsgenerator (s. Kapitel „Zufallsgeneratoren“), das Erzeugen von Iteriertenbahnen und das Liefern von *Salz*.[57]

Während die Programme für die Prozessoren 1 und 2 als Serviceprogramme zur Abwicklung der angeforderten Ein- und Ausgaben verstanden werden, hat der Prozessor 3 nicht nur den Verarbeitungsprozess durch den `plumChip` anzuleiten, sondern auch die Benutzerführung abzuarbeiten.

Im `plumChip` werden die Datenverarbeitungsvorgänge derzeit über eine quadratische Matrix M abgewickelt, die in Abb. 55 als `A-Matrix` beschriftet und die in den RAM-Blöcken des FPGA untergebracht ist. Da ein einziger RAM–Block dafür zumeist nicht ausreicht,[58] werden mehrere RAM–Blöcke zusammengelegt. Als Programmierender der Anwendungssoftware braucht man darauf keine Rücksicht zu nehmen, denn der `plumChip` erledigt den Zugriff so, als hätte man die Matrix im Ganzen vor sich. Es ist im Gespräch, eine zweite Matrix in den RAM–Blöcken unterzubringen, damit man einen Datenblock umschlüsseln kann, ohne dass außerhalb Klartext erscheint.[59]

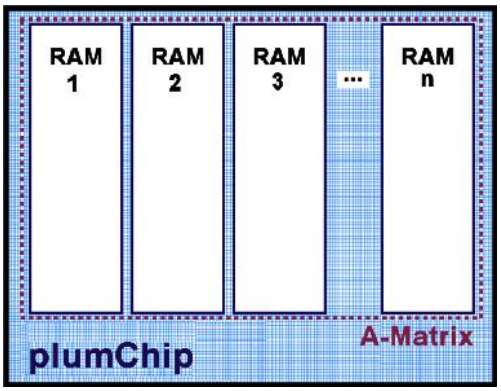


Abb. 55: Die Unterbringung der Matrix geschieht im `plumChip` in RAM-Blöcken.

Um alle Spalten der Matrix zur Hälfte zufällig mit Einsen zu belegen, beginnt der `plumChip` im RAM–Block 1 in Spalte 1 (ganz links, siehe Abb. 55) mit Hilfe seines Zufallsgenerators, dem alternierender Stop–and–Go–Generator aus Abb. 24, solange Positionen auszuwürfeln und dort Einsen hineinzuschreiben, bis die Spalte genau gleich viele gesetzte wie gelöschte Bits enthält, gleich viele Nullen und Einsen. Sind die Spalten eines RAM–Blocks von links nach rechts auf diese Weise versorgt worden, setzt der `plumChip` mit dem nächsten RAM–Block fort, immer streng in der Reihenfolge von links nach rechts. Es ist zu beachten, dass die RAM–Blöcke vor dieser Operation gelöscht werden, das geschieht nicht automatisch.

Neben der Matrix enthält der `plumChip` ein Lernantwortregister L und ein Frageregister Q, wie in Abb. 56 dargestellt. Im allgemeinen Fall dienen diese beiden Register der Aufnahme eines Frage–Antwort–Paares, welches in die Matrix durch die L- oder K–Lernregel eingetragen werden soll.[60] Für unsere Zwecke werden die beiden Register jedoch anders eingesetzt.

Das Frageregister dient bei der Verarbeitung von Daten dem Aufbewahren der für den jeweiligen Datenblock genutzten Iterierten.[61] Das Lernantwortregister enthält den zu ver- oder entschlüsselnden Datenblock.

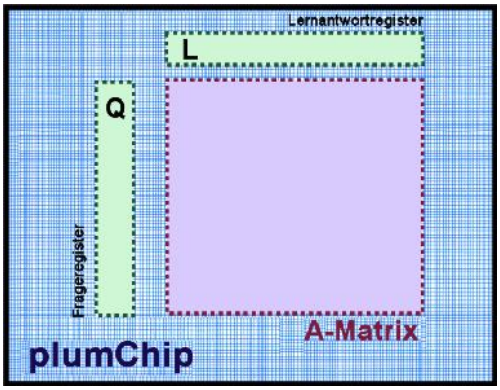


Abb. 56: Das Frage– und ein Antwortregister dienen dem `plumChip` auch zur Generierung einer Iteriertenbahn.

Vor der Verarbeitung eines Datenblocks fordert der Programmierende die nächste Iterierte an. Dabei wählt er auch das Binarisierungsverfahren.[62]

Im `plumChip` befinden sich derzeit Schaltungen für drei verschiedene Verfahren zur Binarisierung, das *MINA*–, *LSB*– und *Sigma*–Verfahren. Durch diese Schaltungen wird aus dem bisherigen Inhalt des Frageregisters mit Hilfe der Matrix der nächste Inhalt des Frageregisters gebildet, der anschließend mit dem nächsten Datenblock geXORt werden kann.

Zur Auslösung der Befehle werden dem `plumChip` über seine seriellen Schnittstellen Befehlsbuchstaben übermittelt, zum Beispiel hat der Befehlsbuchstabe `z` zur Folge, dass die Matrix des `plumChip` spaltenstarr gefüllt wird. Wenn das erledigt ist, gibt der `plumChip` als Bereitschaftssignal die Zeichenfolge `>ok<` an den Prozessor zurück und wartet auf den nächsten Befehlsbuchstaben.

Die oben beschriebene Hardwarekonfiguration wurde (jeweils) als ein eigenständiges Gerät gefertigt und mit den entwickelten Softwaremodulen ausgestattet. Eine Variante ist das Labor–Modell „Flieder–Plum“. Sie hat diverse Anschlüsse, erlaubt Eingaben per Touch–Display (7 Zoll), zeigt einzelne Schritte des Ablaufs an und erlaubt Messungen sowie Prüfungen.[63]

Abb. 57: Die Flieder–Plum

Es gibt zum Betrieb der „Flieder-Plum“ zwei Modi:

- den *Standard Plum Modus* und
- den *AdHoc Modus*

Im *Standard Plum Modus* wird ein Geräteschlüssel (PIN) benötigt, der die Sperre des Gerätes löst und die Gerätefunktionen aktiviert. Auf diese Weise gelangt man zur Schlüsselverwaltung und zu weiteren Funktionen. Die Verarbeitung einer Datei geschieht nach deren Bereitstellung (z.B. vom Host) und der Auswahl des gewünschten Startschlüssels. Die Salzvergabe geschieht hier immer automatisch, genauso wie die Erzeugung der jeweiligen Matrix mittels des ausgewählten Schlüssels.

Im *AdHoc Modus* hingegen liegt der Startschlüssel nicht auf dem Gerät, vielmehr fordert die Flieder–Plum ihn ad hoc an; dann wird aus der Eingabe ein kompletter Durchlauf der Verarbeitungsschritte unmittelbar vorgenommen. Das Ergebnis ist eine mit diesem Schlüssel verarbeitete Datei (auf dem USB–Ausgang). Der Benutzer wird hier durch ein Menü geführt, welches die Abb. 58 zu den Dialogabläufen zeigt.

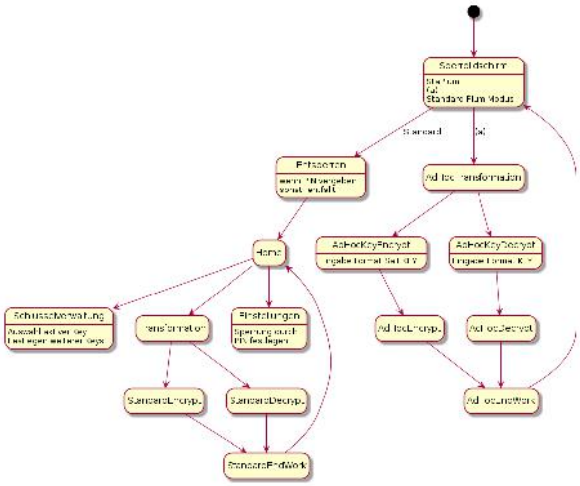


Abb. 58: Die Flieder–Plum–Dialogführung

Das Format der Eingabe des Startschlüssels (KEY) sieht auch eine Hinzufügung von Salz vor, und zwar entweder durch den Benutzer per Hand oder durch die Plum automatisch: `<SALZ:KEY>`. Als Trennzeichen ist der Doppelpunkt „`:`“ festgelegt. Mit diesen Möglichkeiten können – neben dem Einsatz in der täglichen Praxis – auch Demonstrationen und Prüfungen der Einzelschritte vorgenommen werden. Hierbei liegt alles in der Hand des Benutzers und wird zu 100 % von ihm kontrolliert. Der AdHoc Modus ist auch bei ansonsten gesperrter Flieder–Plum verfügbar. Da der KEY nicht auf dem Gerät verwaltet wird, muss man ihn sich allerdings sorgfältig merken.

Ein `plumGerät` für einen anderen Einsatzzweck ist die „RackPlum“ (s. Abb. 59), die, wie der Name schon verrät, für ein Rack[64] im IT–Bereich gedacht ist, an den Unbefugte schon durch bauliche Maßnahmen nicht herankommen. Daher spart man sich zwei der Prozessoren ein, die auf einem SFTP–Boards bereit stehen. Im `plumChip` kann man dann die Kommunikationsbefehle weglassen, die die Zusammenarbeit der drei `Beagles` regeln, was den Schaltungsaufwand vermindert.

Abb. 59: Die RackPlum

In der Abb. 60, die Festerling [13. Januar 2021] entnommen wurde, erhält man eine Übersicht zu den Anschlüssen und einen Einblick in den Zweck dieses `plumGeräts`. Über einen USB–Anschluss ((2)) wird dem Gerät der Startschlüssel mitgeteilt, also die gewünschte Iteriertenbahn angewählt, und über den Ethernet–Anschluss ((6)) werden anschließend die sicher abzulegenden oder abgelegten Datenblöcke geholt und weitergeleitet. Einige Leuchtdioden ((1)), ((4)), ((5)) und ((7)) geben über den Betriebszustand der RackPlum Auskunft.

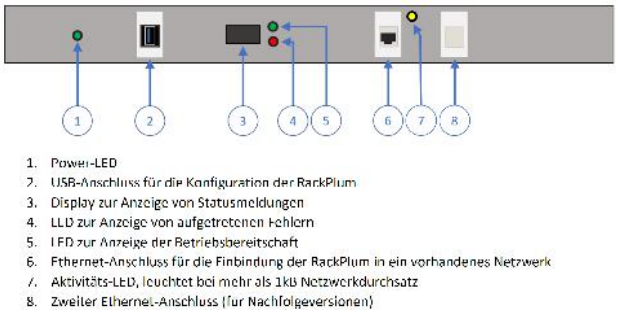


Abb. 60: Die Anschlussseite der RackPlum

Anmerkungen

[52]aplum — associative processes linking universal memory, d. i. in assoziative Prozesse einzubindender, vielseitig einsetzbarer Baustein; aPlum — Gerät, welches einen aplum für kryptografische Zwecke einsetzt; staPlum — stationär zu betreibende Version einer aPlum

[53]SFTP–Board — staPlum-FPGA-Trägerplatine

[54]FPGA — Field Programmable Gate Array, d. i. ein integrierter Schaltkreis, in den digitale, logische Schaltungen geladen werden können

[55]Als Prozessoren sind auf dem SFTP–Board drei `Beagle Bone Black` eingesetzt.

[56]Auf der SFTP-Platine ist er durch die Aufschrift „B3“ markiert.

[57]Unter einem *Salz* wird hier eine zufällige Bitfolge verstanden, mit der der Startschlüssel einer Iteriertenbahn vor dem Verarbeiten der Daten geXORt wird. Dadurch wird verhindert, dass ein und dieselben Daten auf dem Speichermedium als solche erkennbar sind. Für die Erzeugung von Salz wurde im `plumChip` ein eigener alternierender Stop–and–go–Generator derselben Bauform wie in Abb. 24 eingebaut, der sich beim Einschalten des Geräts selbsttätig in Gang setzt und unentwegt läuft.

[58]Im zurzeit eingesetzten FPGA befinden sich RAM–Blöcke der Größe 10 KBit.

[59]vgl. H.-J. Bentz: „Vektographie und Assoziative Transformationsmatrizen“, imbit.net, 07. Dezember 2020

[60]s. Bentz and Dierks [2013], Kapitel 2.5.2

[61]Iterierte — Zwischenergebnisse ein und desselben Verfahrens, das wiederholt angewandt wird; hier: Es entsteht dadurch eine Iteriertenbahn (Iteriertenfolge).

[62]s. Kapitel „Binarisierungsverfahren“ und „Statistische Untersuchungen“ bezüglich der Unterschiede der Verfahren

[63]Die „Flieder–Plum“ enthält ein SFTP–Board, das in ein metallenes Pultgehäuse eingebaut wurde. Es ist mit zwei USB–Ports für `Beagle` 1 und 2 versehen.

[64]*engl.* rack — Rahmen, Schrank, Gestell; hier: Haltevorrichtung für IT–Komponenten

Unterschiede zu AES und ECC

Ein informationstechnisches Verfahren zum Sichern und Übertragen von Daten, das sich auf zu zeitaufwendigen Berechnungen von Primzahlzerlegungen verlässt, wird darauf achten müssen, dass es von der Entwicklung leistungsfähigerer Rechner oder Rechenverfahren nicht überholt wird, die das Faktorisieren erheblich schneller erledigen. Das asymmetrische Verschlüsselungsverfahren *RSA* setzt auf eine solche Faktorisierung und wird daher vom Bundesamt für die Sicherheit in der Informationstechnik (BSI) nur bis zum Jahr 2022 mit Schlüssellängen von mindestens 2.000 Bit empfohlen, danach sollte die Schlüssellänge mindestens 3.000 Bit betragen. Mit Primzahlzerlegung oder anderen zahlentheoretischen Algorithmen haben wir es bei dem vorstehend beschriebenen Verfahren zur IT-Sicherheit mit Assoziativmatrizen nicht zu tun.[65]

Beim Stichwort Matrix könnte man an die Matrizen denken, die im weit verbreiteten *AES*–Verfahren[66] benutzt werden. Dabei handelt es sich jedoch einerseits um zwei Ersetzungstabellen, zwei Substitutionstafeln, die bei der Berechnung der Rundenschlüssel des *AES* zum Einsatz kommen, die eine beim Verschlüsseln, die andere beim Entschlüsseln.[67] Sie haben zwar wie unsere $n \times n$–Matrix auch eine Größe von 128×128 Bit, aber sie sind konstant, nicht zufällig bestimmt, sondern in immer gleicher Weise festgelegt. Und andererseits wird bei der Variante *AES–128* in eine 128×128 Bit große Tafel der zu ver- oder entschlüsselnde Datenblock eingetragen. Auch das ist also mit der zufällig befüllten Assoziativmatrix nicht zu verwechseln, sondern ihr Inhalt ist von den zu verarbeitenden Daten bestimmt.[68]

Ein Datenblock aus 16 Bytes mit dem Text `imbit Hildesheim` füllt einen 128×128–Block zum Beispiel in folgender Weise:[69]

$$\begin{bmatrix} \mathrm{i} & \mathrm{t} & \mathrm{l} & \mathrm{h} \\ \mathrm{m} & & \mathrm{d} & \mathrm{e} \\ \mathrm{b} & \mathrm{H} & \mathrm{e} & \mathrm{i} \\ \mathrm{i} & \mathrm{i} & \mathrm{s} & \mathrm{m} \end{bmatrix}$$

Dieser Datenblock wird bei *AES–128* in zehn Runden durch umkehrbare Operationen wie Rotieren, Ersetzen, Mischen und Addieren des Rundenschlüssels umgeformt und zum Schluss ins Ziel geschrieben.

Das *AES*-Verfahren zeichnet sich vermeintlich dadurch aus, dass noch keine Angriffe auf das Verfahren bekannt wurden, die in der Praxis relevant sind. Es lässt sich auch auf leistungsschwächeren Prozessoren so implementieren, dass es ausreichend schnell läuft. Das *AES*–Passwort wird zu im Fortgang des Verfahrens genutzten Rundenschlüsseln expandiert (gleiches Passwort, gleiche Rundenschlüssel) und steht bezüglich der Sicherheit im Zentrum.

Man erkennt, dass *AES* im Vergleich zur unserer Verarbeitung von Daten mit Assoziativmatrizen nichts dem Zufall überlässt.

Noch spannender ist die sichere Verarbeitung von Daten mit Hilfe elliptischer Kurven. Ihr Vorteil bestehe darin, mit erheblich kürzeren Schlüssellängen als das *RSA*–Verfahren auszukommen. Beispielsweise erreiche man bei einem *ECC*–Verfahren[70] mit einem Schlüssel der Länge 160 Bit eine ähnliche Sicherheit wie bei *RSA* mit 1.024 Bit (vgl. Paar and Pelzl [2016], S. 291).[71]

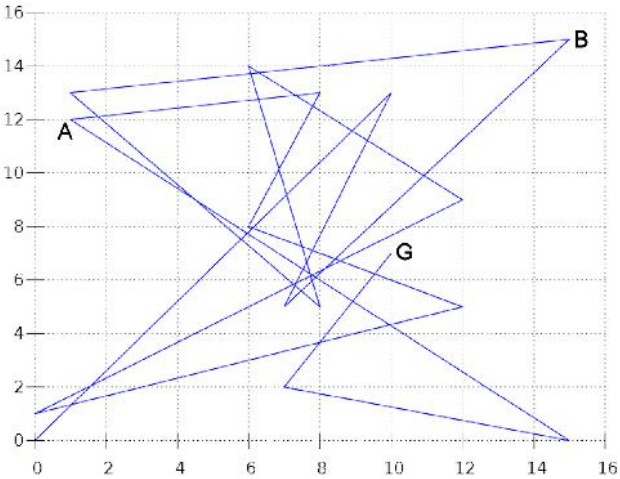


Abb. 61: Auf einer elliptischen Kurve, die von einem Generatorpunkt G ausgeht, befinden sich die Punkte A und B, deren Besitzer den Punkt des jeweils anderen nicht kennen.

Durch einen anfänglichen Schlüsselaustausch, der dank eines privaten und eines öffentlichen Schlüssels als sicher angenommen wird,[72] legen sich Sender und Empfänger auf einen gemeinsamen Punkt K fest, der auf der durch im Vorfeld getroffene Absprachen bestimmten elliptischen Kurve liegt (wie in Abb. 61). Von diesem Punkt K aus folgen beide der Kurve Schritt für Schritt und nutzen die Koordinaten der besuchten Punkte — Koordinaten, die übrigens Polynome sind — zur Ver- beziehungsweise Entschlüsselung ihrer Daten.

Dieses Vorgehen erinnert dem Folgen einer Iteriertenbahn von einer Startfrage aus, bei dem jede Iterierte der Verarbeitung der Datenblöcke dient. In dem einen Fall läuft man auf einer elliptischen Kurve, im anderen Fall auf einem Vektographen. Hier gleichen sich unser Verfahren. Doch stützt sich *ECC* auf das wohldefinierte Rechnen in besonderen endlichen Körpern[73] und besitzt eine eigene Addition von Punkten auf der elliptischen Kurve, deren Summe wieder ein Punkt ist, der auf der Kurve liegt, sowie eine Punktverdoppelung mit den gleichen guten Eigenschaften. Zufällige Prozesse spielen bei *ECC* jedenfalls keine Rolle.

Durch diese Beschreibungen werden die grundlegenden Unterschiede zwischen herkömmlichen Verfahren und unserem Verfahren zur IT–Sicherheit mit Assoziativmatrizen deutlich: Am Anfang steht der Zufall, und danach verschwindet er auch nicht. In einer Iteriertenbahn einer spaltenstarren Assoziativmatrix kann man sich den vorliegenden Erkenntnis-

sen zufolge (s. Kapitel „Statistische Untersuchungen“ zu 128×128–Matrizen) recht sicher sein, dass keine wohlbekannte mathematische Berechnung von der einen Iterierten zur nächsten führt, sondern dass ein Pseudozufall generiert wie von einem guten Zufallsgenerator hinter der Abfolge steckt.

Die Iterierten einer Bahn werden zur informationstechnischen sicheren Datenablage und -übertragung eingesetzt. Es ist, als regierte der Münzwurf, ob eine Null oder eine Eins an einer beliebigen Position einer Iterierten auftaucht. Und doch behält man alles unter Kontrolle.

Anmerkungen

[65] s. BSI [29. Mai 2018], S. 38

[66] AES — Advanced Encryption Standard

[67] Zu den Verfahren *RSA* und *AES* stehen ausführliche Erläuterungen mit Beispiel in Dierks [02. Mai 2019].

[68] vgl. Daemen and Rijmen [2002]

[69] Jedes Zeichen wird mit 8 Bits dargestellt, also kommt man mit $4 \times 4 \times 8 = 128$ auf die erwähnte 128×128–Matrix.

[70] ECC — elliptic curve cryptography

[71] Zu *ECC* gibt Dierks [08. Mai 2019] ausführliche Erklärungen und ein Beispiel.

[72] Zur Sicherheit des Schlüsselaustausches s. Beutelspacher et al. [2010], S. 137f.

[73] Die Rechnungen werden zum Beispiel in GALOIS–Körpern oder in Primkörpern ausgeführt.

Literatur

Hans-Joachim Bentz and Andreas Dierks. *Neuromathematik und Assoziativmaschinen.* Springer Verlag, Berlin Heidelberg, 2013. ISBN 978-3-642-37937-6.

Albrecht Beutelspacher, Heike B. Neumann, and Thomas Schwarzpaul. *Kryptografie in Theorie und Praxis — Mathematische Grundlagen für Internetsicherheit, Mobilfunk und elektronisches Geld, 2. Auflage.* Vieweg + TVieweg, Wiesbaden, 2010. ISBN 978-3-8348-0977-3.

Jürgen Braun. *1000x1000_ZFM_100_Abfragen, 1000x1000_ZFM_fixed_100_Abfragen.* imbit.net, 04. Januar 2019.

Bundesamt für Sicherheit in der Informationstechnik BSI. *Technische Richtlinie — Kryptographische Verfahren: Empfehlungen und Schlüssellängen, Version 2018-02.* https://www.bsi.bund.de, Bonn, 29. Mai 2018.

E. Rodney Canfield and Brendan D. McKay †. *Asymptotic Enumeration of Dense 0–1 Matrices with Equal Row Sums and Equal Column Sums.* Department of Computer Science, University of Georgia, Athens, USA, 2004.

John Daemen and Vincent Rijmen. *The Design of Rijndael: AES — The Advanced Encryption Standard.* Springer, Berlin Heidelberg, 2002. ISBN 978-3-540-42580-9.

Andreas Dierks. *Gold-, Silber-, Bronzeschlüssel.* imbit.net, 02. Dezember 2019.

Andreas Dierks. *Verschlüsselungsverfahren RSA und AES.* imbit.net, 02. Mai 2019.

Andreas Dierks. *Zufallsgenerator des plumChips in der Version v2.4.* imbit.net, 05. August 2020.

Andreas Dierks. *ECC — Elliptische Kurven.* imbit.net, 08. Mai 2019.

Andreas Dierks. *Testverfahren für die Iteriertenfolge.* imbit.net, 20. Januar 2021.

Andreas Dierks. *LFSR für plumChip v2.0.* imbit.net, 20. Oktober 2019.

Andreas Dierks. *VidAs — Aufbau einer robusten, frei programmierbaren Maschine aus Assoziativmatrizen. Simulation und Hardware-Lösung.* Universität Hildesheim, 2005. in: „Hildesheimer Informatik-Berichte“, Band 2/2005 (September 2005), ISSN 0941-3014.

Oliver Festerling. *RackPlum Prototyp Benutzerdokumentation.* imbit.net, 13. Januar 2021.

Mari Miyamoto. *Randomness tests on the sequences generated by the column rigid matrix and the mean reduction.* imbit.net, 15. Dezember 2020.

Christof Paar and Jan Pelzl. *Kryptografie verständlich — Ein Lehrbuch für Studierende und Anwender.* Springer Vieweg, Berlin Heidelberg, 2016. ISBN 978-3-662-49296-3.

Günther Palm. *Neural Assemblies — An Alternative Approach to Artificial Intelligence.* Springer-Verlag, Berlin Heidelberg New York, 1982. ISBN 3-540-11366-5.

Günther Palm and Hans-Joachim Bentz. *Theoretische Untersuchungen zu einfachen neuronalen Netzen.* imbit.net, 2021.

Detlef Romberger. *Erzeugung doppeltstarrer Matrizen $n \times n$.* imbit.net, 18. November 2019.

Bruce Schneier. *Applied Cryptography, second edition — Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Indianapolis, 20th anniversary edition, 1996. ISBN 978-1-119-09672-6.

Karl Steinbuch. *Automat und Mensch — Kybernetische Tatsachen und Hypothesen.* Springer Verlag, Berlin Heidelberg New York, 3. Auflage, 1965.

John Walker. *HotBits: Genuine random nnumber, generated by radioactive decay.* fourmilab.ch, 2017.

Redaktion: a.dierks@imbit.net, Stand: 9. Februar 2021

Dieses Projekt wurde durch Mittel des Europäischen Fonds für regionale Entwicklung unterstützt.

EUROPÄISCHE UNION
Europäischer Fonds für regionale Entwicklung

EUROPA FÜR NIEDERSACHSEN

Anhang: Mathematischer Hintergund

Gegeben seien binäre Tupel $f \in \{0,1\}^n$.

Wir betrachten eine Abbildung

$$[\ \ M]\_R:\ \{0,1\}^n \xrightarrow{M} \{0,1,2,\ldots,n\}^n \xrightarrow{R} \{0,1\}^n, \quad f \mapsto [f \cdot M]\_R,$$

wobei der erste Teil mittels der Multiplikation mit einer binären $n \times n$–Matrix M durchgeführt wird[74] und der zweite Teil mit einer Binarisierungsfunktion R geschieht.[75] Die Abbildung ist weder injektiv noch surjektiv und im Allgemeinen auch nicht linear.

Zusätzlich definieren wir eine Folge (f_j) iterativ:

$$f_j = [f_{j-1} \cdot M]\_R \ \text{ für } \ j \in \{1,...,k\},\ k \in \mathbb{N}.$$

Der Beginn der Iteration wird durch eine 0. Iterierte f_0 festgelegt.[76]

Eine Kette aufeinanderfolgender Iterierter bildet eine *Iteriertenbahn*[77] oder ein Teilstück einer Bahn. Die Menge aller Bahnen bildet einen Graphen, den wir *Vektograph* genannt haben.[78] Die Konkatenation von k Iterierten in einer Bahn oder einem Bahnabschnitt liefert eine binäre Sequenz der Länge $k \cdot n$ (Bit), deren kryptographische Eigenschaften von Interesse sind und von uns genutzt wird; jeder Knoten f im Vektographen erscheint als *Block* von n Bits. Diese Blöcke dienen der Verschlüsselung von Datenblöcken gleicher Größe.

Die Abbildungen $[\ \ M]\_R$ zeigen Eigenschaften von Einwegfunktionen, da sie einfach zu berechnen, aber schwierig umzukehren sind. Die Berechnung von $[\ \ M]\_R$ erfordert einen Aufwand, der von der Multiplikation von f mit M und der Durchführung der Binarisierungsfunktion R abhängt. Bei $[\ \ M]\_MAX$ und den anderen beschriebenen Binarisierungsverfahren werden diese Berechnungen in Polynomialzeit ausgeführt,[79] womit das erste Kriterium für Einwegfunktionen erfüllt ist. Das zweite Kriterium würde sich erfüllen, wenn kein Algorithmus zu finden wäre, der durch die Wahl zufälliger Zwischenergebnisse und ohne zusätzlicher Informationen in polynomialer Zeit von einem f_j auf f_{j-1} zurückschließen ließe.

Des Weiteren lässt sich untersuchen, ob die Umkehrung von $[\ \ M]\_R$ mit Hilfe der Kenntnis von M oder R ohne großen Aufwand gelingen kann. Damit hätte $[\ \ M]\_R$ die Eigenschaft einer Falltürfunktion.

Anmerkungen

[74] Zu den Elementen von M s. S. 1f.

[75] Einige dieser Funktionen sind in Kapitel „Binarisierungsverfahren“, S. 10ff, beschrieben.

[76] Die Bedeutung der 0. Iterierten als Startschlüssel/Startfrage wird auf S. 4 verdeutlicht.

[77] s. S. 3

[78] Die Vielfalt an Gestalten der Vektographen wird im Kapitel „Iteriertenbahnen“ sichtbar.

[79] Die Binarisierungsfunktion *MAX* ist auf S. 4 oder in Bentz and Dierks [2013], S. 43f, erklärt. Die Multiplikation von f mit M ist durch $O(n^2)$ beschränkt, die Suche nach dem größten Wert im Zwischenergebnis und die Abbildung des Zwischenergebnisses auf $\{0,1\}^n$ durch $O(n)$.