MINISTERIUM FÜR BILDUNG UND WISSENSCHAFT DER UKRAINE

STAATLICHE UNIVERSITÄT SUMY

ABTEILUNG FÜR ELEKTRONIK UND COMPUTERTECHNIK

# ERKLÄRENDES SCHREIBEN

Zur Bachelorarbeit zum Thema:

## ADRESS -VEKTORMETHODE VON BINÄRER DATENKOMPRESSION

Leiter der Abteilung: Opanasyuk A.S.

Erster Gutachter: Kulyk I.A.

Zweite Gutachterin: Zaitseva I.A.

Verfasserin: Moshchna I.B.

Gruppe TK-51

SUMY, JAHR 2019

# REFERAT

Das Thema und der Studienbereich dieser Bachelor Arbeit sind Datenkompression und ihre Arten und Methoden. Im Teil „Literaturrecherche" wurden wichtigste Difinitionen zum Thema „Datenkompression" erklärt, verschiedene Methoden zur verlustfreien und verlusthaften Datenkompressierung betrachtet und vergliechen und bedeutsamste Bewertungskriterien von Kompressionseffizienz gegeben.

Das Ziel dieser Studie ist Entwicklung einer effizienten Methode zur Datenkompressierung, der die folgenden Anforderungen erfüllt:

- Einfachheit ihrer technischen Umsetzung haben;
- Hohe Kompressionsgeschwindigkeit und Wiederherstellungsgeschwindigkeit von übertragenden Informationen enthalten;
- Kein Informationsverlust bei Datenkonvertierung verursachen;
- Allgemeingültigkeit für alle verarbeitenden Datenarten.

Die oben genannten Eigenschaften hat die Adress-Vektormethode von binärer Datenkompression. Dieses Projekt hat als Ziele das Folgende:

- Entwicklung einen verallgemeinerten Algorithmen zum Informationscodieren und -Decodieren nach der Adress-Vektorkomprimierungsmethode;
- Untersuchung wichtigsten Parameter von Effizienz der Algorithmen zur Adress-Vektorkomprimierung und Adress-Vektorwiederherstellung;
- Konstruktion eines Strukturmodells eines Telekommunikationssystems mit Anwendung der Adress-Vektorkomprimierung;
- Darstellung eines Funktionsschemas eines Telekommunikationssystems unter Verwendung der Adress-Vektorkomprimierung.

Diese Bachelor Arbeit besteht aus zwei Teilen und enthält 41 Seiten, 16 Abbildungen und 3 Tabellen. Im Anhang sind Strukturmodell eines Systems der Adressvektorkomprimirung, Blockschaltbild des allgemeinen Algorithmus von *A*VK (Adress-Vektorkomprimierung) und *A*VD (Adress-Vektorwiederherstellung oder Decodierung) und Graph von relativer Reduzant *A*VK und Funktionsgraphen für die Verktor- und Adresscodirung dargestellt.

# INHALT

## RELEVANZ DER STUDIE

Fast jeder Computerbenutzer benutzt freiwillig oder unfreiwillig, bewusst oder unbewusst Kompressionsverfahren: absichtlich z.B. zur Archivierung von Dateien, um deren Speicherkapazität zu senken, unfreiwillig bei einer Verwendung von Standarddatenformaten, in denen eine zu vordefininiernden Datenkompression schon enthält.[1]

Damit Information nicht mehr Speicherkapazität als nötig belegen und sich schnell unabhängig von Umfang sowie Größe übertragen lassen, werden sie comprimiert. Hierfür kommen, abhängig vom Anwendungszweck und der Art der Information, verschiedene Kompressierungsverfahren zum Einsatz.

Speichermedien bieten immer mehr Kapazität, ebenso lassen sich Informationen über moderne, feste und mobile Breitbandnetze schneller übertragen als je zuvor. Der auf diesen Gebieten permanent erzielte technische Fortschritt macht die Kompression von Daten jedoch keineswegs obsolet, da die Menge an erzeugten Informationen sowie die Größe der Dateien in einem weitaus höheren Taktschlag zunimmt.

Infolgedessen kommt heute jeder von uns tagtäglich mit Datenkompressionsverfahren in Berührung. Einerseits unbewusst – beispielsweise beim Besuch von Internetseiten, sobald wir an Online-Telefonkonferenzen teilnehmen, Videos abrufen oder das Fußball-Topspiel der Woche per Livestream verfolgen. Andererseits bewusst, indem wir Programme nutzen, um die Größe von Textdokumenten, Bildern, Präsentationen, Audio-Files und vielen weiteren Dateien so zu verringern, dass wir sie platzsparend archivieren und schnell übertragen können.[2]

# 1 LITERATURRECHERCHE UND PROBLEMSTELLUNG

## 1.1 Definitionen

### 1.1.1 Entropie

***E**ntrop**ie*** (***i**nform**ati**v*) - ein Maß für Zufälligkeit von Informationen, die Unsicherheit von Erscheinung eines Symbols eines primären Alphabets. Falls verlustfreier Informationsübertragung entspricht dies numerisch der Informationsmenge pro Zeichen der übertragenen Nachricht.

**Formale Definitionen**

Eine Informationsentropie für unabhängige zufällige Ereignisse x mit n möglichen Zuständen (von 1 bis n) berechnet sich nach der folgenden Formel:

$$H(x) = -\sum_{i=1}^{n} p(i) \log_2 p(i)$$

Dieser Wert wird auch als durchschnittliche Entropie der Nachricht bezeichnet. Der Wert $log_2 \frac{1}{p(i)}$ wird als Teilentropie bezeichnet, die nur den *i*-ten Zustand kennzeichnet.

Somit ist die Entropie des Ereignisses x die Summe mit dem entgegengesetzten Vorzeichen aller Erzeugnise von relativer Häufigkeit des Auftretens des Ereignisses und mit ihren binären Logarithmen multipliziert. Diese Definition für diskrete Zufallsereignisse kann für die Wahrscheinlichkeitsverteilungsfunktion erweitert werden.

Sh***a***nnon hat diese Entropiedefinition aus folgenden Annahmen abgeleitet:

- Maßnahme sollte kontinuierlich sein; das heißt, eine Änderung des Wertes des Wahrscheinlichkeitswerts um eine kleine resultierende Änderung der Entropie verursachen sollte;
- Falls, alle Varianten (die Buchstaben im obigen Beispiel) gleich wahrscheinlich sind, sollte eine Erhöhung der Anzahl der Varianten (Buchstaben) immer die Gesamtentropie erhöhen;
- Es sollte möglich sein, eine Auswahl (in unserem Beispiel für Buchstaben) in zwei Schritten zu treffen, wobei die Entropie des Endergebnisses die Summe aller Entropien von Zwischenergebnisse sein sollte.

Shannon zeigte, dass jede Entropiedefinition, die diese Annahmen erfüllt, in der folgenden Form vorliegen sollte:

$$-K \sum_{i=1}^{n} p(i) \log_2 p(i)$$

Wobei K ist eine Konstante (und eigentlich nur zur Auswahl von Maßeinsen benötigt wird).

Also, Shannon stellte fest, dass Entropie eine Differenz zwischen in der Nachricht enthaltenen Informationen und einem Teil der Informationen ist, der in der Nachricht bekannt ist (oder gut vorhergesagt wird).

### 1.1.2 Datenkompression

Datenkompression ist eine algorithmische Datenkonvertierung, die zur Reduzierung des von ihnen belegten Volumens durchgeführt wird. Es wird für eine rationellere Verwendung von Datenspeicher- und -übertragungsgeräten verwendet. Der umgekehrte Vorgang wird als Datenwiederherstellung (Entpacken, Dekomprimieren) bezeichnet.

Die Komprimierung basiert auf der Beseitigung der in den Quelldaten enthaltenen Redundanz. Das einfachste Beispiel für Redundanz ist die Wiederholung von Fragmenten im Text (z. B. Wörter aus der natürlichen oder Maschinensprache). Eine solche Redundanz wird normalerweise beseitigt, indem eine sich wiederholende Sequenz durch einen Verweis auf ein bereits codiertes Fragment ersetzt wird, das seine Länge angibt. Eine andere Art der Redundanz ist auf die Tatsache zurückzuführen, dass einige Werte in komprimierbaren Daten häufiger vorkommen als andere. Die Reduzierung des Datenvolumens wird erreicht, indem häufig angetroffene Daten durch kurze Codewörter und seltene Daten durch lange Codewörter ersetzt werden (Entropiecodierung). Eine Komprimierung von Daten, die nicht redundant sind (z. B. ein zufälliges Signal oder weißes Rauschen, verschlüsselte Nachrichten), ist grundsätzlich ohne Verlust nicht möglich.

Falls einer verlustfreien Komprimierung können die ursprüngliche Nachrichten vollständig wiederherstellt werden, da in der darin enthaltene Informationsmenge trotz der Verringerung ihre Länge ihr Inhat nicht ändert wird. Eine solche Möglichkeit ergibt sich nur, wenn die Wahrscheinlichkeitsverteilung auf dem Nachrichtensatz nicht einheitlich ist, beispielsweise ein Teil der Nachrichten, die in der vorherigen Codierung theoretisch möglich sind, in der Praxis nicht angetroffen wird.

### 1.1.3 Verlustbehaftete Kompression

In meisten Fällen implementiert eine Datenübertragung in Telekommunikationssystemen verlustbehaftete Komprimierungsmethoden. Unter Berücksichtigung der Besonderheiten von übertragenen Daten können Verfahren unterschieden werden, die sich auf die Komprimierung von physiologischen Signalen, Bildern, Audio-, hauptsächlich Sprach- und Videodaten konzentrieren.

Datenverlustbehaftete Komprimierung ist eine Methode zur Datenkomprimierung (Komprimierung), bei der sich die entpackten Daten von den ursprünglichen Daten unterscheiden. Allerdings ist es trotz der Differenz problemlos, diese Information auf der empfangenden Seite zu erkennen und weiter anzuwenden. Diese Art der Komprimierung wird häufig verwendet, um Audio- und Videodaten, Bilder in Internet (insbesondere im Streaming) und die digitale

Telefonie zu komprimieren. Eine Alternative ist verlustfreie Komprimierung. Betrachten wir weiter bekannteste verlustbehaftete Kompressionsmethoden.

## JPEG-Verfahren

Das JPEG-Verfahren (Joint Photographic Expert Group) wurde Anfang der 90er von einer Expertengruppe entwickelt um ein leistungsfähiges Kompressionsverfahren für Bilder/Fotos zu entwickeln. JPEG stellt bis heute das am weitesten verbreitete Bildformat dar und ist im Internet allgegenwärtig.

Das JPEG-Verfahren ist eine Hintereinanderschaltung mehrerer Kompressions- und Transformationsverfahren. Die folgende Skizze zeigt das grobe Ablaufschema.

Farbbilder werden normalerweise mit Hilfe von drei Farbkanälen gespeichert (Rot, Grün, Blau), die als RGB-Format bezeichnet werden. Die optische Wahrnehmung des Menschen nimmt geringe Helligkeitsunterschiede sehr viel stärker war als minimale Farbunterschiede. Diese Gegebenheit führte zur „Entwicklung“ eines neuen Farbraums, dessen drei Kanäle als Y, U und V bezeichnet werden. Y stellt dabei den Helligkeitswert dar, der sich durch folgende empirische Formel aus dem RGB-Farbraum ableitet:

$$Y = 0{,}299\,R + 0{,}587\,G + 0{,}114\,B$$

Die beiden anderen Kanäle enthalten die Farbinformationen als Differenz der Kanäle R und B von der Helligkeit. Der Grün-Faktor stellt in der oben berechneten Helligkeit ja bereits fast 60% dar. Die beiden anderen Teile werden folgendermaßen berechnet:

$$\mathbf{U = R - Y}$$
$$\mathbf{V = B - Y}$$

Diese Umwandlung reduziert die anfallenden Daten jedoch nicht, sondern transformiert die Daten nur in eine bessere Form. Wie schon erwähnt reagiert das Auge wesentlich sensitiver auf Helligkeitsänderungen als auf Farbänderungen.

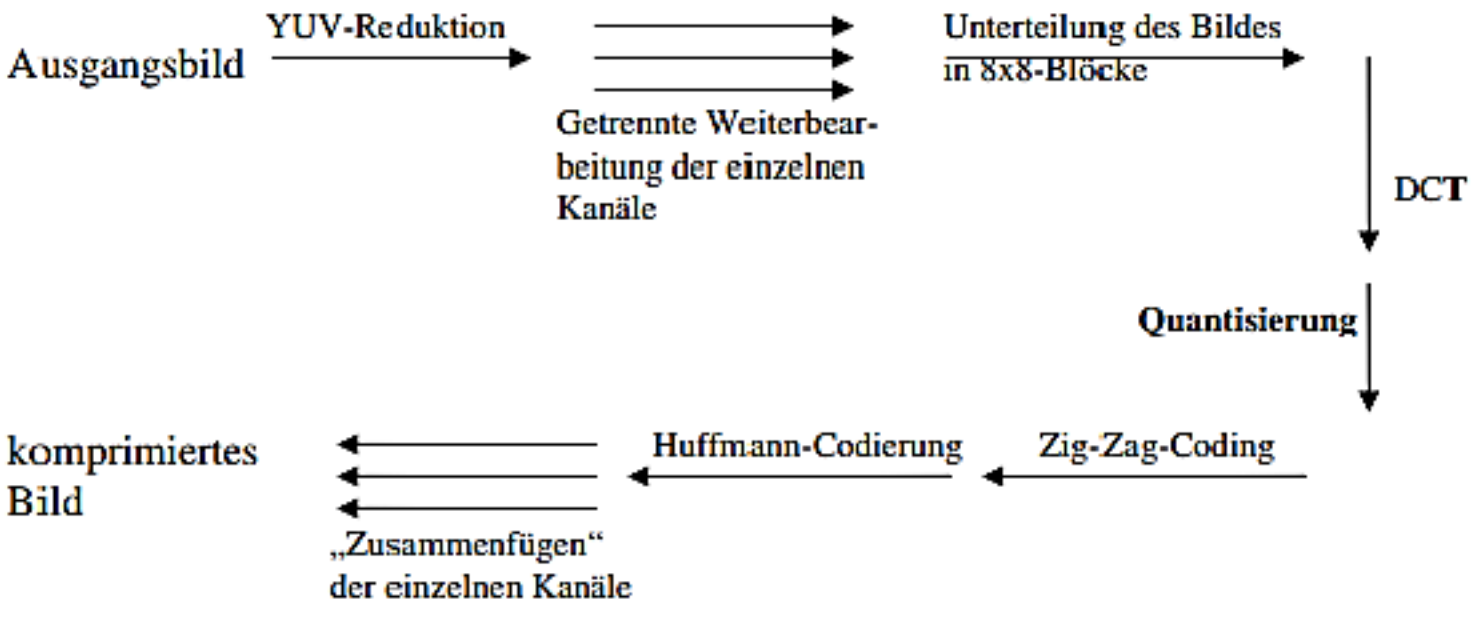



*Abb. 1.1: das JPEG-Verfahren*

Durch die „Entkopplung“ der Helligkeitsinformationen aus den drei Farbkanälen RGB in einen separaten Helligkeitskanal können die Farbkanäle wesentlich stärker (verlustbehaftet) komprimiert werden, ohne dass die Helligkeit des Bildes beeinflusst wird (Die Kompression erfolgt durch eine Reduktion der Auflösung der beiden Farbkanäle.).

In den folgenden Schritten werden die einzelnen Kanäle getrennt voneinander bearbeitet. Im nächsten Schritt wird das Ausgangsbild in 8x8-große Blöcke (*A*bb. 1.1) zerteilt. Die 64 Farbwerte dieser Blöcke werden nun mittels einer diskreten Cosinustransformation (DCT) untersucht und transformiert. Die transformierten Werte (*A*bb. 1.2) besitzen im Gegensatz zu den Ausgangswerten eine günstigere Beschaffenheit für die anschließenden Schritte. Im nächsten Schritt, der Quantisierung, werden die Werte des 8x8-großen Blocks durch die Werte einer ebenfalls 8x8-großen Quantisierungstabelle dividiert und die entstehenden Werte auf Ganzzahlen gerundet.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 139 | 144 | 149 | 153 | 155 | 155 | 155 | 155 |
| 144 | 151 | 153 | 156 | 159 | 156 | 156 | 155 |
| 150 | 155 | 160 | 163 | 158 | 156 | 156 | 156 |
| 159 | 161 | 162 | 160 | 160 | 159 | 159 | 159 |
| 159 | 160 | 161 | 161 | 160 | 155 | 155 | 155 |
| 161 | 161 | 161 | 161 | 160 | 157 | 157 | 157 |
| 162 | 162 | 161 | 163 | 162 | 157 | 157 | 157 |
| 162 | 162 | 161 | 161 | 163 | 158 | 158 | 158 |

*A*bb. 1.2: F*a*rbw*e*rt*e* d*e*r *A*usg*a*ngsd*a*t*ei*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 235.6 | -1.0 | -12.1 | -5.2 | 2.1 | -1.7 | -2.7 | 1.3 |
| -22.6 | -17.5 | -6.2 | -3.2 | -2.9 | -0.1 | 0.4 | -1.2 |
| -10.9 | -9.3 | -1.6 | 1.5 | 0.2 | -0.9 | -0.6 | -0.1 |
| -7.1 | -1.9 | 0.2 | 1.5 | 0.9 | -0.1 | 0.0 | 0.3 |
| -0.6 | -0.8 | 1.5 | 1.6 | -0.1 | -0.7 | 0.6 | 1.3 |
| 1.8 | -0.2 | 1.6 | -0.3 | -0.8 | 1.5 | 1.0 | -1.0 |
| -1.3 | -0.4 | -0.3 | -1.5 | -0.5 | 1.7 | 1.1 | -0.8 |
| -2.6 | 1.6 | -3.8 | -1.8 | 1.9 | 1.2 | -0.6 | -0.4 |

*A*bb. 1.3: W*e*rt*e* n*a*ch d*e*r DCT

Durch diese Prozedur werden die kleinen durch die DCT entstehenden Werte auf Null gerundet. Je kleiner die Werte der Quantisierungstabelle sind, desto besser die spätere Bildqualität (Übergabe der Qualitätseinstellungen des Benutzers an den Encoder). Es entsteht der in *A*bb. 1.3 dargestellte 8x8-große Block, der fast nur aus Nullen besteht. Der anschließende Schritt des Zig-Zag-Codings ist eine spezielle

Form der RLE (verlustfrei!). Hierbei werden die Werte nicht zeilenweise eingelesen, sondern wie der Name besagt in Zick-Zack-Form um die Anzahl sich wiederholender Nullen ohne Unterbrechung zu erhalten. Zur weiteren Verkleinerung der Datenmenge wird noch das Huffmann-Verfahren angewendet.

Die Vorteile des JPEG-Verfahrens sind die effektive Kompression bei guter Bildqualität und die optionale Progressivität (das Bild wird nicht zeilenweise, sondern mit immer mehr Details aufgebaut), die zur Verbreitung im Internet sehr beigetragen hat. Dennoch hat das JPEG-Verfahren einige Mängel.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 240 | 0 | -10 | 0 | 0 | 0 | 0 | 0 |
| -24 | -12 | 0 | 0 | 0 | 0 | 0 | 0 |
| -14 | -13 | 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 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Abb. 1.4: Werte nach der Quantisierung der DCT-Werte*

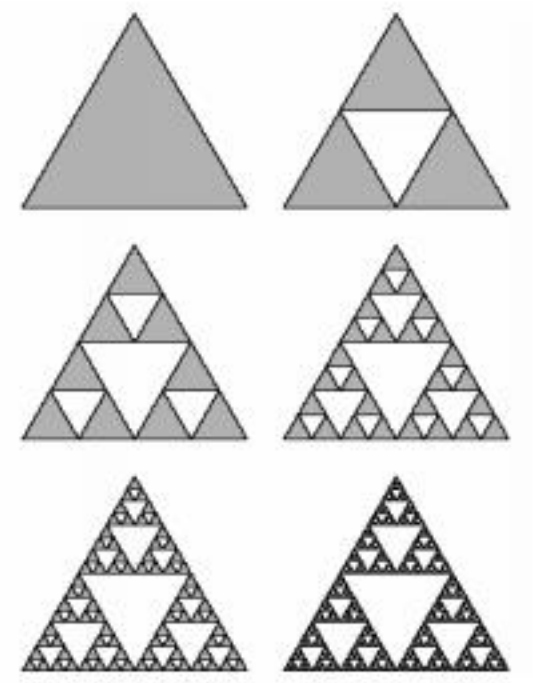
So entstehen durch die Verarbeitung der 8x8-großen Blöcke ohne Abgleich mit den Nachbarblöcken an den Blockgrenzen z.T. harte Übergange. Die Abschätzung der Größe des JPEG-Bildes ist aufgrund der abstrakten Parameter (es kann nur eine abstrakte Qualitätseinstellung an den Encoder übergeben werden) nicht möglich.[8]

## Fraktale Bildkompression

Die Grundidee der fraktalen Bildkompression ist, dass in einem Bild oft mehrere sehr ähnliche Objekte auftreten bzw. das ganze Bilder aus nur einem „Grundbaustein“ bestehen. Diese Grundbausteine könnten durch die Verwendung von „programmierten Linsen“ vergrößert, gedreht, gespiegelt und verschoben werden. Durch diese mit wenigen Daten (Linsenparameter) durchführbaren Aktionen könnte ein Bild sehr effektiv gespeichert werden. An dem folgenden Beispiel (Sierpinski-Dreieck) soll die Effektivität des Verfahrens demonstriert werden. Das rechte untere Dreieck kann durch die Verkleinerung und der Verschiebung der oberen Dreiecke erzeugt werden. Es ist verständlich, dass die Speicherung des linken oberen Dreiecks und ein paar „Linsenparametern“ sehr Platz sparend ist.

Dieses Verfahren bietet zwar gigantische Kompressionsraten, ist aber aufgrund der enormen Rechenleistung zur Auffindung von Mustern nicht für den Hausgebrauch geeignet. Obwohl der Verbreitung im Internet nichts im Wege stehen

würde, da dieses Verfahren sowohl optionale Progressivität als auch im Vergleich zur Kompression eine blitzschnelle Dekompression besitzt.[8]



*A*bb. 1.5: Fr*a*kt*a*l*e* B*i*ldkompr*e*ssion

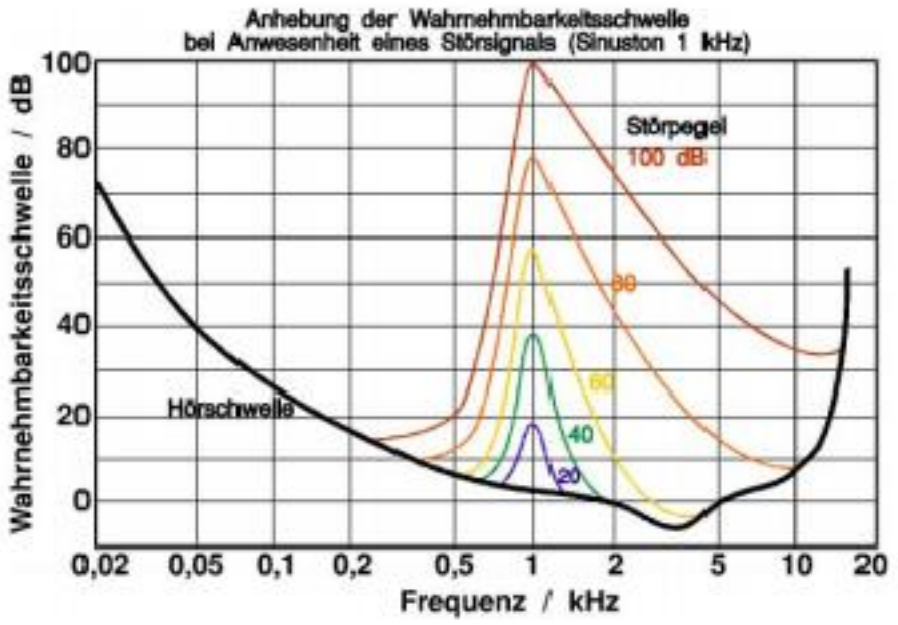
**MP3 (MPEG-Audio-Layer 3)**

Das wohl bekannteste Audiokompressionsverfahren wurde Anfang der 90er entwickelt und erreichte durch die massenhafte Verbreitung von MP3-Files in den Internettauschbörsen Ende der 90er einen hohen Bekanntheitsgrad. Das Verfahren beruht auf der Ausnutzung der Schwächen des menschlichen Ohres, die von der Psychoakustik untersucht werden. Im Folgenden soll kurz auf die Methoden zur Ausnutzung der Schwächen, die als Maskierung bezeichnet wird, eingegangen werden. Eine wichtige Maskierung bei der MP3-Kompression stellt die simultane oder zeitgleiche Maskierung dar. Aus *A*bb. 1.6 ist ersichtlich, dass ein Störsignal die Hörschwelle der „benachbarten" Frequenzen deutlich beeinflusst. Durch diese Tatsache ist es dem Encoder möglich diejenigen Frequenzen, deren Schalldruck sich unterhalb der Hörschelle befindet, zu entfernen.

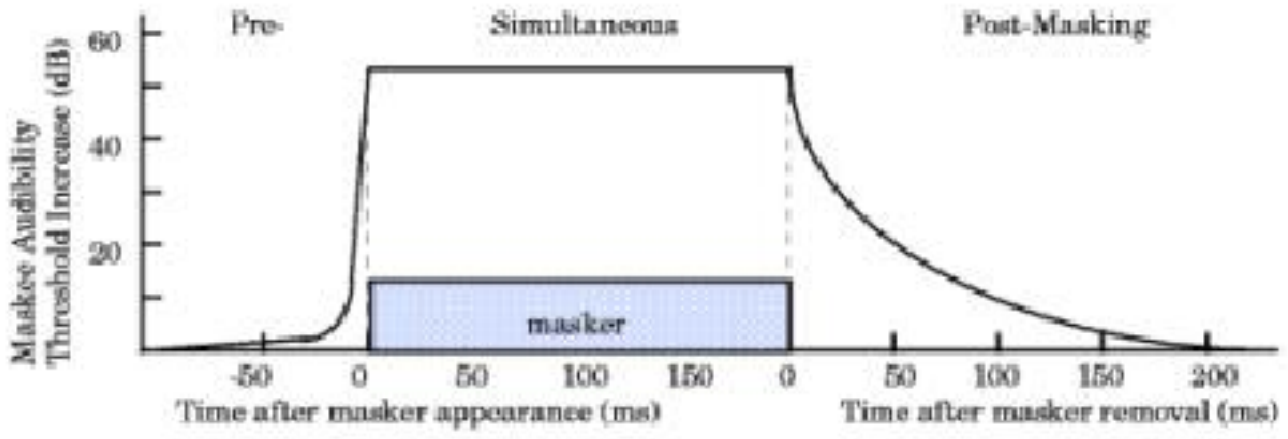
Dadurch kann die eingehende Informationsmenge beträchtlich reduziert werden. Eine weitere Maskierungsart ergibt sich aus der physiologischen Begebenheit, dass ein lautes Geräusch die Wahrnehmung anderer Töne für eine gewisse Zeit vor und nach seinem Auftreten verhindert (*A*bb. 1.7).

Diese Maskierungsart wird als zeitliche Maskierung bezeichnet. Sie ermöglicht die Entfernung „unhörbarer Zeitbereiche" aus dem Eingangsmaterial. Die letzte und ebenfalls sehr effektive Maskierungsart wird als Stereo-Maskierung bezeichnet und berücksichtigt die Schwäche des menschlichen Gehörs bei der Ortung von Geräuschen. So können zum Beispiel tiefe Töne nicht geortet werden (Subwoofereffekt) und müssen folglich nicht in Stereo gespeichert werden. Des Weiteren unterscheiden sich die beiden Kanäle (links und rechts) in den meisten Fällen nur minimal, so dass eine Reduktion der Stereoinformation auf einen Kanal

und die Speicherung eines zusätzlichen Seitensignals, das die Unterschiede zwischen den beiden Ursprungskanälen speichert, eine enorme Platzersparnis bietet (Joint-Stereo-Coding).




*A*bb. 1.6: Hörschwelle des Menschen




*A*bb. 1.7: Phänomen der zeitlichen Maskierung

Das MP3-Kompressionsverfahren bietet eine Reihe von Vorteilen, die z.T. auch zu seiner rasanten Verbreitung führten. So kann man Audio-Daten um den Faktor 10 komprimieren und kaum Unterschiede zur CD-Qualität feststellen. Zudem ist es im Vergleich zum JPEG-Verfahren möglich, eine konstante Kompressionsrate und somit die Endgröße genau festzulegen. Auch die Erweiterung der MP3-Kompression mit der Verwendung variabler Bitraten trug zur Steigerung der Sound-Qualität der MP3- Files bei, da für komplexere Musikbereiche mehr Speicher zur Verfügung gestellt wurde, der in einfacheren Bereichen „eingespart“ werden konnte.[7]

### 1.1.4 Verlustfreie Kompression

Verlustfreie Komprimierung ist ein Art von Datenkompression, in dem die Daten Bit-Zu-Bit wiederhergestellt werden können, was zu keinem

Informationsverlust führt. Die verlustfreie Komprimierung zeigt jedoch normalerweise die schlechtesten Komprimierungsverhältnisse.

Alle Datenkomprimierungsmethoden basieren auf einem einfachen logischen Prinzip. Wenn wir uns vorstellen, dass die gängigsten Elemente mit kürzeren Codes und seltener mit längeren Codes codiert werden, benötigen alle Daten weniger Speicherplatz als wenn alle Elemente durch Codes gleicher Länge dargestellt würden.

Die genaue Beziehung zwischen der Häufigkeit des Auftretens von Elementen und der optimalen Länge von Codes ist im sogenannten Shannon's Source Coding Theorem beschrieben, das die Grenze der maximalen verlustfreien Komprimierung und Shannon's Entropie definiert.

Am meisten wird verlustfreie Komprimierung bei Textdateien anwendet, weil in diesem Fall es sehr wichtig ist, die Datei auf der Empfangseite Wort-für-Wort wiederherstellen zu können. Weiter wird ein der bekanntesten Algorithmus zur Komprimierung beschrieben, das Huffman-Algorithmus.

## Huffman-Algorithmus

Der Huffman-Algorithmus verwendet die Häufigkeit des Auftretens identischer Bytes im Eingabedatenblock und setzt entsprechend häufig vorkommende Blöcke einer Folge von Bits kürzerer Länge und umgekehrt. Dieser Code ist der mindestens redundante Code. Betrachten einen Fall, in dem das Alphabet des Ausgabestreams unabhängig vom Eingabestream nur aus zwei Zeichen besteht - Null und Eins.

Wenn wir mit einem Huffman-Algorithmus codieren, müssen wir zunächst ein Schema von Summen konstruieren. Dies geschieht wie folgt:

- Alle Buchstaben des eingegebenen Alphabets werden in absteigender Reihenfolge der Wahrscheinlichkeit sortiert. Alle Wörter aus dem Ausgabestream-Alphabet (das heißt, was wir codieren) werden anfangs als leer betrachtet (das Ausgabestream-Alphabet nur aus {0,1} Zeichen besteht).
- Zwei Zeichen $a_{j-1}$ und $a_j$ des Eingabestreams mit den geringsten Auftrittswahrscheinlichkeiten werden zu einem "Pseudo-Symbol" kombiniert, wobei die Wahrscheinlichkeit p gleich der Summe der Wahrscheinlichkeiten der darin enthaltenen Zeichen ist. Dann hängen wir 0 an den Anfang des Wortes $B_{j-1}$ und 1 an den Anfang des Wortes $B_j$ an, was anschließend die Zeichencodes $a_{j-1}$ bzw. $a_j$ sind.
- Diese Zeichen wirt aus dem Alphabet der ursprünglichen Nachricht gelöscht, jedoch das generierte Pseudozeichen zu diesem Alphabet hinzugefügt (es muss natürlich an der richtigen Stelle unter Berücksichtigung der Wahrscheinlichkeit in das Alphabet eingefügt werden).

Die Schritte 2 und 3 werden wiederholt, bis nur noch eins Pseudozeichen im Alphabet vorhanden ist, das alle Anfangssymbole des Alphabets enthält. Da bei jedem Schritt und für jedes Zeichen das entsprechende Wort $B_i$ geändert wird (durch

Hinzufügen von Eins oder Null), wird nach Abschluss dieser Prozedur ein Anfangscode $B_i$ jedem Anfangszeichen des Alphabets $a_i$ entsprechen.

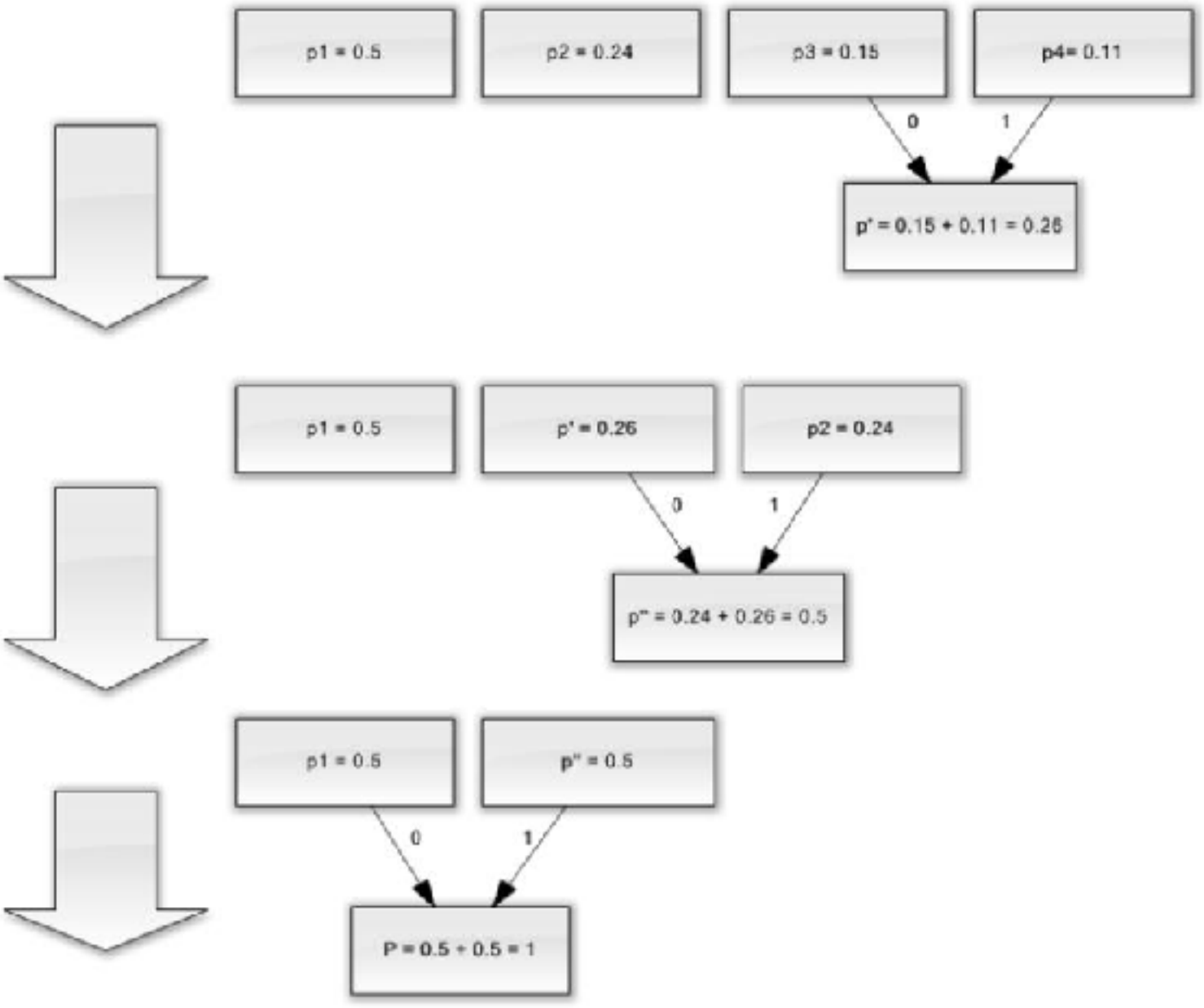
Betrachten wir zur besten Veranschaulichung ein kleines Beispiel.

Es gibt ein Alphabet, das nur aus vier Zeichen besteht - $\{a_1, a_2, a_3, a_4\}$. Es ist auch angenommen, dass die Wahrscheinlichkeiten des Auftretens dieser Symbole $p_1 = 0{,}5$; $p_2 = 0{,}24$; $p_3 = 0{,}15$; $p_4 = 0{,}11$ jeweils gleich sind (die Summe aller Wahrscheinlichkeiten ist offensichtlich gleich eins).

Also konstruieren wir ein Schema für das gegebene Alphabet.

1 Zwei Zeichen mit geringsten den Wahrscheinlichkeiten (0,11 und 0,15) werden zu einem Pseudo-Symbol p' kombiniert.
2 Diese kombinierten Zeichen werden gelöscht und das resultierende Pseudozeichen in das Alphabet eingefügt.
3 Zwei Zeichen mit den geringsten Wahrscheinlichkeiten (0,24 und 0,26) werden zum Pseudozeichen p'' vereinigt.
4 Die kombinierten Zeichen werden gelöscht und das resultierende Pseudozeichen in das Alphabet eingefügt.
5 Zum Schluss werden die verbleibenden zwei Zeichen zusammengeschlossen und wir erhalten die Baumspitze.

Wenn diesen Vorgang veranschaulicht wird, erhalten wir etwa Folgendes:




*A*bb.1.8: Huffm*a*n-*A*lgor*i*thmus zur D*a**t*encod*ie*rung

Wie es dargestellt ist, bei jeder Vereinigung den zu verbindenden Zeichen die Codes 0 und 1 zugewiesen wird. Auf diese Weise können wir beim Bauen eines Baums leicht den Code für jedes Zeichen erhalten. In unserem Fall sehen die Codes folgendermaßen aus:

$$a_1 = 0$$
$$a_2 = 11$$
$$a_3 = 100$$
$$a_4 = 101$$

Da keiner dieser Codes ein Präfix eines anderen ist (d.h., der Präfixsatz wurde erhalten), könnte jeder Code im Ausgabestream eindeutig identifiziert werden. Wir haben also erreicht, dass das häufigste Symbol durch den kürzesten Code codiert wird und umgekehrt.

Wenn wir annehmen, dass anfangs ein Byte zum Speichern jedes Zeichens verwendet wurde, können wir berechnen, um wie viel es uns gelungen ist, die Daten zu reduzieren.

### 1.1.5 Informationsredundanz

Redundanz ist ein Begriff aus der Informationstheorie, der einen Überschuss von Informationsmenge bezeichnet, die zum Übertragen oder Speichern einer Nachricht über ihre Informationsentropie verwendet wird. Um die Redundanz zu verringern, wird eine datenverlustfreie Komprimierung verwendet, während die Prüfsumme verwendet wird, um dem Datenstrom zusätzliche Redundanz hinzuzufügen, wodurch Fehler bei der Übertragung von Informationen über Verzerrungskanäle (Satellitenrundfunk, drahtlose Übertragung usw.) korrigiert werden können.

### Formale Definition

Der Informationsgehalt einer einzelnen Nachricht in einem Stream ist im Allgemeinen wie folgt definiert:

$$r = \mathbb{E}H(M_t|M_{t-1}, M_{t-2}, M_{t-3}, \ldots)$$

Bezeichnen Sie mit R den Logarithmus der Anzahl der Zeichen im Nachrichtenalphabet:

$$R = \log|M|$$

Absolute Redundanz kann als Differenz dieser beiden Größen definiert werden:

$$D = R - r$$

Das Verhältnis $\frac{D}{R}$ wird relative Redundanz genannt und gibt eine mathematische Schätzung des maximalen Komprimierungsverhältnisses an, um das die Dateigröße reduziert werden kann.

## 1.2 Bewertungskriterien für die Datenkomprimierungsqualität

Der Zweck des Komprimierungsprozesses besteht in der Regel darin, einen kompakteren Ausgabestream von Informationseinsen aus einem anfänglich nicht kompakten Eingabestream unter Verwendung einiger ihrer Transformationen zu erhalten. Die wichtigsten technischen Merkmale der Kompressionsprozesse und die Ergebnisse ihrer Arbeit sind:

- Verdichtungsverhältnis (compress rating) oder das Verhältnis (ratio) ein Volumenverhältnis von anfänglichen und resultierenden Streams;
- Komprimierungsgeschwidigkeit - eine Zeit, die für ein Komprimieren einer bestimmten Menge an Informationseingabestream aufgewendet wurde, um daraus den äquivalenten Ausgabestream zu erhalten;
- Komprimierungsqualität - Ein Wert, der angibt, wie stark der Ausgabestream gepackt ist, indem auf ihn eine erneute Komprimierung mit demselben oder einem anderen Algorithmus angewendet wird.

### Verdichtungsverhältnis

Ein Merkmal des Komprimierungsverhältnisses ist das Komprimierungsverhältnis: Es gibt ein Verhältnis von Größe einer komprimierten Datei zu ihrer ursprünglichen Größe:

$$R_1 = \frac{V_{vor\ Komprimierung}}{V_{nach\ Komprimierung}},$$

$$R_2 = \frac{V_{vor\ Komprimierung} - V_{nach\ Komprimierung}}{V_{vor\ Komprimierung}} = 1 - \frac{V_{nach\ Komprimierung}}{V_{vor\ Komprimierung}} =$$

$$= 1 - R_1$$

Wobei $V_{nach\ Komprimierung}$ - eine Informationsmenge nach einer Komprimierung ist; und $V_{vor\ Komprimierung}$ - eine Informationsmenge vor einer Komprimierung ist. In der Praxis wird der Koeffizient $R_2$ verwendet.

Das Kompressionsverhältnis kann sowohl konstant (einige Kompressionsalgorithmen für Ton, Bild usw.) als auch variabel sein. Im zweiten Fall kann es entweder für jede bestimmte Nachricht bestimmt oder anhand einiger Kriterien bewertet werden:

- Üblich (normalerweise für einige Testdaten);
- Maximum (bei bester Kompression);
- Minimum (bei schlechtester Kompression);

Das verlustbehaftete Komprimierungsverhältnis hängt stark vom zulässigen Komprissionsfehler, der normalerweise als Parameter des Algorithmus dient. Im

allgemeinen Fall kann ein konstantes Komprimierungsverhältnis nur verlustbehaftete Datenkomprimierungsmethoden liefern.

Kompressionsverhältnis hängt vom verwendeten Programm, der Komprimierungsmethode und dem Typ von Quelldatei ab. Am besten könnten Bilddateien, Textdateien und Datendateien komprimiert werden, für die der Kompressionsverhältnis 5 - 40% erreichen kann. Etwa schlechter könnten Dateien aus ausführbaren Programmen und Lademodulen komprimiert werden, - nur 60 - 90%. Archivdateien werden fast nicht komprimiert. Archivierungsprogramme unterscheiden sich in den verwendeten Komprimierungsmethoden, was sich entsprechend auf den Komprimierungsgrad auswirkt.

## Irrelevanzreduktion

Das Hauptkriterium für den Unterschied zwischen Kompressionsalgorithmen ist es, ob sie einen Informationsverlust verursachen. Im Allgemeinen sind verlustfreie Komprimierungsalgorithmen universell und ihre Verwendung für Daten jeglicher Art unbedingt möglich ist, während die Verwendung verlustbehafteter Komprimierung gerechtfertigt sein muss. Bei einigen Datentypen ist eine Verzerrung grundsätzlich nicht zulässig. Dazu gehören:

- Programme und ihre Quelltexte, binäre Arrays usw.;
- Vitaldaten, Änderungen, die zu kritischen Fehlern führen können: zum Beispiel von medizinischen Messgeräten oder Steuergeräten von Flugzeugen, Raumfahrzeugen usw .;
- Zwischendaten, die während der mehrstufigen Verarbeitung von Grafik-, Audio- und Videodaten wiederholt komprimiert und wiederhergestellt werden.

## Systemanforderungen zu Algorithmen

Unterschiedliche Algorithmen erfordern möglicherweise unterschiedliche Rechenressourcen von Computersystem, auf denen sie implementiert sind:

- RAM (für Zwischendaten);
- permanenter Speicher (für Programmcode und Konstanten);
- CPU-Zeit.

Im Allgemeinen hängen diese Anforderungen von der Komplexität und „Intelligenz“ des Algorithmus ab. Die allgemeine Tendenz ist wie folgt: Je effizienter und universeller der Algorithmus ist, desto höher sind die Anforderungen an die Rechenressourcen. In bestimmten Fällen können jedoch einfache und kompakte Algorithmen genauso gut funktionieren wie komplexe und universelle. Die Systemanforderungen bestimmen ihre Verbraucherqualitäten: Je weniger anspruchsvoll der Algorithmus ist, desto einfacher und daher kompakter, zuverlässiger und billiger kann das System implementiert werden.

Da die Komprimierungs- und Wiederherstellungsalgorithmen paarweise arbeiten, ist das Verhältnis der Systemanforderungen für sie wichtig. Oft kann die

Komplizierung eines Algorithmus den anderen erheblich vereinfachen. Somit gibt es drei Möglichkeiten:

1) Der Komprimierungsalgorithmus erfordert mehr Rechenressourcen als der Wiederherstellungsalgorithmus.

Dies ist das häufigste Korrelationsmerkmal in Fällen, in denen einmal komprimierte Daten wiederholt verwendet werden. Ein Beispiel sind digitale Audio- und Videoplayer.

2) Komprimierungs- und Wiederherstellungsalgorithmen erfordern ungefähr die gleichen Rechenressourcen.

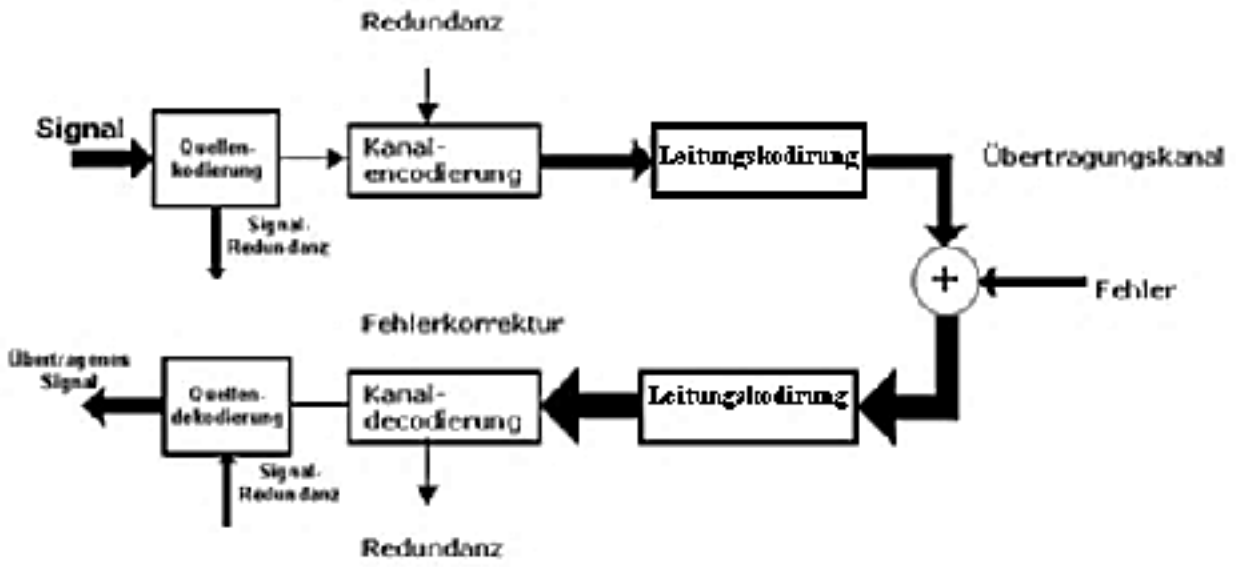
Die akzeptabelste Option für Kommunikationsleitungen, wenn die Komprimierung und Wiederherstellung einmal an beiden Enden erfolgt (beispielsweise bei der digitalen Telefonie).

3) Der Komprimierungsalgorithmus ist viel weniger anspruchsvoll als der Wiederherstellungsalgorithmus.

Diese Situation ist typisch für Fälle, in denen der Komprimierungsvorgang von einem einfachen, häufig tragbaren Gerät ausgeführt wird, für das die Menge der verfügbaren Ressourcen sehr kritisch ist, z. B. einem Raumfahrzeug oder einem großen verteilten Netzwerk von Sensoren. Es können auch Daten sein, die Sie in einem sehr kleinen Prozentsatz von Fällen entpacken müssen, z. B. zum Aufzeichnen von Überwachungskameras.

## 1.3 Anwendung in der Nachrichtentechnik

Bei der Datenübertragung wird häufig die zu übertragende Datenmenge durch Kompression reduziert. In so einem Fall spricht man dann auch von Quellenkodierung.[6][7] Die Quellenkodierung wird dabei häufig zusammen mit Kanalkodierung und Leitungskodierung verwendet, sollte aber nicht mit diesen verwechselt werden: Während die Quellencodierung überflüssige (redundante) Information einer Datenquelle reduziert, hat die Kanalcodierung die Aufgabe, durch zusätzlich eingebrachte Redundanz Übertragungs- bzw. Speicherfehler im Rahmen der Datenübertragung erkennen und korrigieren zu können. Die Leitungskodierung hingegen nimmt eine spektrale Anpassung des Signals an die Anforderungen des Übertragungskanals vor.[9]




*A*bb. 1.9: Nutzung von Qu*e*ll*e*n-, K*a*n*a*l- und L*ei*tungskod*ie*rung zur Übertr*a*gung *ei*n*e*s Sign*a*ls

## 1.4 Problemstellung

Im Hinblick auf die Literaturrecherche, in der Merkmale von modernen Telekommunikationssystemen untersucht wurden, kann man das Folgende schließen. Eine der effektivsten Methode, - um Bandbreite von Informationsquellen und Telekommunikationssysteme im Allgemeinen zu erhöhen, - ist die Verwendung von Kompressionstechniken und Hardware- oder Softwarekomponenten, die diese Techniken in Telekommunikationssysteme implementieren.

Bei relativ geringen Kosten einer solchen Implementierung ist es möglich, die durchschnittliche Geschwingichkeit von Datenübertragung signifikant zu erhöhen. Aus dieser Sicht vielversprechend ist die Verwendung der AV-Komprimierung, die die folgenden positiven Eigenschaften hat:

- Einfachheit ihrer technischen Umsetzung;
- Hohe Kompressionsgeschwindigkeit und Wiederherstellungsgeschwindigkeit von übertragenden Informationen;
- Kein Informationsverlust bei Datenkonvertierung;
- Allgemeingültigkeit für alle verarbeitenden Datenarten;

In diesem Zusammenhang sind folgende Aufgaben für den Forschungsteil des Diplomprojekts relevant:

- Entwicklung einen verallgemeinerten Algorithmen zum Informationscodieren und -Decodieren nach der *A*V-Komprimierungsmethode zur Verwendung in Telekommunikationssystemen;
- Bewertung wichtigsten Parameter der Effizienz der Algorithmen zur *A*V-Komprimierung und *A*V-Wiederherstellung.

Die Aufgaben für den technischen Teil dieses Projekts sind wie folgt formuliert:

- Konstruktion eines Strukturmodells eines Telekommunikationssystems mit Anwendung der *A*V-Komprimierung;
- Die Entwicklung eines Funktionsschemas eines Telekommunikationssystems

unter Verwendung der *A*V-Komprimierung, die (System) einen allgemeinen Charakter von praktischer Implementierung haben muss.

## 2 ADRESS-VEKTORMETHODE VON BINÄRER DATENKOMPRESSION

### 2.1 Definition und Beschreibung der Adress-Vektormethode

Es gibt eine Binärkodierung f: $A \to B$, die eine injektive Funktion ist. Und die Binärkodierung f ist eine Abbildung der Ausgangsmenge (oder Definitionsmenge) $A$ auf die Zielmenge B von Binärwörtern. Die injektive Funktion f muss die folgenden Anforderungen erfüllen:

1) Injektivität der Abbildung: für alle $a_i$ und $a_j$ aus $A$
   $f(a_i) \neq f(a_j); i, j = 1,...,|A|$
2) Dekodierung der Kodierung, nämlich muss eine Eins-zu-Eins-Zerlegung der Verkettung der Kodewörter in die ursprünglichen ausführen können.

Gemäß der Problemstellung sind Qualitätsindikatoren für die Abbildung f - Redundanz, Kodierungszeit und Kostenvolumen. In den Arbeiten wird das Wesentliche der betrachteten Kompressionsmethode angegeben und werden ihre Redundanz und Anwendungsbedingungen dargestellt.

Betrachten wir eine getrennt genommene Informationsquelle $Y=\{a_j \in A_k, j=1,..., C_n^k\}$. Die Y besteht aus Elementen einer Äquivalenzklasse aus der Struktur der Bernoulli-Quelle $A$. Die bedingte Entropie einer solchen Quelle ist die mathematische Erwartung von $\log_2 C_n^k$ oder die Entropie einer kombinatorischen Quelle $A_k$:

$$H(A_k) = \log_2 C_n^k \tag{2.1}$$

Ein Merkmal der Quelle Y ist eine Abhängigkeit ihrer Entropie sowohl von der Wahrscheinlichkeitsverteilung $P_k$, als auch von $H(A_k)$, oder von der Potenz der Äquivalenzklasse $A$, deren Bereich der Veränderung von $C_n^0$ bis $C_n^{[k/2]}$ ist.

Für die Redundanz der ursprünglichen Vektorkodierung $f_v$ der Menge $A_k=\{a_j: r(a_j)=k, j=1,..., C_n^k\}$, wobei $r(a_j)$ die Anzahl der Einsen im Binärdatensatz $a_j$ ist, führt es das Folgende aus:

$$R(f_v, A_k) = n - H(A_k) = n - \log_2 C_n^k$$

oder durch relative Redundanz,

$$r(f_v, k) = 1 - \frac{\log_2 C_n^k}{n}. \tag{2.2}$$

Satz 1. *Sei* $A_k=\{a_j: r(a_j)=k, j=1,..., C_n^k\}$ *eine kombinatorische Quelle, die binäre Wörter* $a_j$ *mit einer gleichen Anzahl von k-Einsen erzeugt, k=0,...,n. Die relative Redundanz* r(fv, k) *einer solchen Quelle nimmt einen Minimalwert an:*

$$\min_{k=0,\dots,n} r(f_v, k) = 1 - \frac{\log_2 C_n^{[n/2]}}{n}, \qquad (2.3)$$

wo k=0 und k=n.

Der Funktionsgraph $r(f_v, k)$ und seine Funktionswerten sind, für n=256, in ***Abb. 1 (Anhang A)*** und in der Tabelle. 2.1 dargestellt. Das Verhalten von $r(f_v, k)$ zeigt abhängig vom Wert von k eine signifikante Ungleichmäßigkeit der Redundanzverteilung. Falls wenn k sich zu 0 und zu 256 nähert, nimmt die Redundanz der Vektorkodierung rapide zu.

Tabelle 2.1: Die Funktionswerten von $r(f_v, k)$ der Vektorcodierung der Äquivalenzklasse $A_k$ in Abhängigkeit von k

| K | $r(f_v, k)$ |
|---|---|
| 0 | 1,00 |
| 32 | 0,47 |
| 64 | 0,20 |
| 96 | 0,06 |
| 128 | 0,02 |
| 160 | 0,06 |
| 192 | 0,20 |
| 224 | 0,47 |
| 256 | 1,00 |

Die Redundanz bei k, die nahezu 0 und zu n sind, kann durch Adresscodierung reduziert werden. Die Adresscodierung wandelt einen binären Vektor in eine Folge um, die aus k Adressen von Nullen und Einsen besteht. Zu diesem Zweck wird jedem Bit des Vektors eine binäre Zahl zugewiesen, die mit einer Null beginnt, und dann werden die Adressen von Einzel- oder Null-Bits übertragen oder gespeichert. Es ist zu beachten, dass zur Reduzierung der Redundanz mit $k > n/2$ die Kodierung von Einsen durch Kodierung (n-k ) Nullen ersetzt werden muss. Somit sind die Kosten für die Adresscodierung $f_a$ der Quelle $A_k$, wodurch n-Bit-Binärvektoren mit einer konstanten Anzahl von k Einsen erzeugt werden:

$$C(f_a, A_k) = \begin{cases} k\lceil \log_2 n \rceil, 0 \le k \le [n/2] \\ (n-k)\lceil \log_2 n \rceil, [n/2] < k \le n \end{cases} \qquad (2.4)$$

Dabei ist $\lceil \log_2 n \rceil$ die kleinste ganze Zahl, und nicht kleiner als $\log_2 n$ ist.

Nach dem Ausdruck (2.1) für die Entropie einer kombinatorischen Quelle betrachten wir die Redundanz der Adresscodierung:

$$R(f_a, A_k) = \begin{cases} k\lceil \log_2 n \rceil - \log_2 C_n^k, 0 \le k \le \lceil n/2 \rceil \\ (n-k)\lceil \log_2 n \rceil - \log_2 C_n^k, \lceil n/2 \rceil < k \le n \end{cases}$$

Daher die relative Redundanz der Adresscodirung oder der Anteil der redundanten Zeichen pro Codewort:

$$r(f_a, k) = \begin{cases} 1 - \dfrac{\log_2 C_n^k}{k\lceil \log_2 n \rceil}, 1 \le k \le \lceil n/2 \rceil \\ 1 - \dfrac{\log_2 C_n^k}{(n-k)\lceil \log_2 n \rceil}, \lceil n/2 \rceil < k \le n \end{cases} \quad (2.5)$$

Satz 2. Sei $A_k = \{a_j : r(a_j) = k, j = 1, ..., C_n^k\}$ eine kombinatorische Quelle, die binäre Wörter $a_j$ mit einer konstanten Anzahl von k Einsen erzeugt, k = 1, ..., n -1. Die relative Redundanz $r(f_a, k)$ der Adresscodierung einer solchen Quelle nimmt den Mindestwert an:

$$\min_{k=1,\dots,n-1} r(f_a, k) = 1 - \frac{\log_2 n}{\lceil \log_2 n \rceil} \quad (2.6)$$

dabei k= 1 und k= n-1, und den Maximalwert:

$$\max_{k=0,\dots n-1} r(f_a, k) = 1 - \frac{\log_2 C_n^{[n/2]}}{\lceil n/2 \rceil \lceil \log_2 n \rceil} \quad (2.7)$$

wo k= [n/2].

Der Ausnahme aus dem Bereich von Änderung der k-Werte von 0 und n spiegelt die Tatsache wider, dass die Kodierung eines Alphabets, das aus einem einzelnen Zeichen besteht, keinen Sinn macht, da das Maß der Ungewissheit hier 0 ist (für k = 0 und k = n ist Potenz der $A_k$ Äquivalenzklasse = 1).

In ***Abb. 2*** (Anhang B) wird eine grafische Darstellung der relativen Redundanz r ($f_a$, k) gezeigt, wenn n = 256 ist, und die Tabelle. 2.2 fasst seine Funktionswerte zusammen.

Tabelle 2.2: die Funktionswerte r ($f_a$, k) zur Adresscodierung der Äquivalenzklasse $A_k$ in Abhängigkeit von k:

| K | r($f_a$, k) |
|---|---|
| 1 | 0,00 |
| 32 | 0,47 |
| 64 | 0,60 |
| 96 | 0,69 |
| 128 | 0,75 |
| 160 | 0,69 |
| 192 | 0,60 |

| K | $r(f_a, k)$ |
|---|---|
| 224 | 0,47 |
| 255 | 0,00 |

Kombinieren wir die Grafiken $r(f_v, k)$ und $r(f_a, k)$ für die Vektor- und Adresscodierung im gleichen Koordinatensystem (***Abb. 3*** (Anhang C)). Im Vergleich zur Vektor- ist das Adresscodierung offensichtlich durch eine wesentlich geringere Redundanz für $k$ nahe zu 1 und zu $n-1$, und umgekehrt eine größere Redundanz für $k$ nahe $[n/2]$, gekennzeichnet.

Nach der Betrachtung des Falls einer separaten Äquivalenzklasse $A_k$ betrachten wir den allgemeinen Fall, in dem die gesamte Nachricht $A$ codiert ist. Gemäß dem Satz 1 kann die Quelle $A=\{a_i: |a_i| = n, i=1,...,|A|\}$ in zwei Quellen umgewandelt werden: eine Quelle $X = \{k = 0,1, ..., n\}$, die eine Anzahl $k$ - Einsen mit der Wahrscheinlichkeiten $P_k$ erzeugt und eine Quelle $Y = \{aj \in Ak, j = 1, ...,\}$ $C_n^k$, die Wörter $aj$ aus der Menge $Ak = \{aj: r(aj) = k, j = 1,\ldots,\}$ $C_n^k$ erzeugt.

Nun kann der Schluss gezogen werden, dass aus Sicht der rationalen redundanten Codierung die Hauptaufmerksamkeit auf die Quelle $Y$ der Äquivalenzklassen zu richten ist. Eine vergleichende Analyse von Vektor- und Adresscodierungsmethoden (***Abb. 3 (Anhang C)***, Satz 1 und 2) zeigt, dass für die Minimierung der Kosten für die Codierung der Quelle $Y$ das Auswahlverfahren für die betrachteten Codierungsmethoden enthalten sein muss, dessen Kriterium der niedrigere Redundanzwert ist. Dies wiederum bedeutet, dass es notwendig ist, Bedingungen zu finden, unter denen der Wert der Kosten $C(f_v, X)$ der Vektorkodierung der Quelle $X$ und der Kosten $C(f_a, Y)$ der Adresskodierung der Quelle $Y$ geringer als die Kosten von $C(f_v, A)$ Quellvektorcodierung der Quelle $A$ sind.

Die Kosten $C(f_v, X)$ werden im Allgemeinen durch die Anzahl von Ränge bestimmt, die für die binäre Darstellung der Anzahl von $k$-Einsen benötigt wird. Die Kosten $C(f_a, Y)$ werden gemäß dem (2.4) berechnet. So lösen wir in Bezug auf $k$ das folgende System der Ungleichungen:

$$\begin{cases} \lceil \log_2(n+1) \rceil + k \lceil \log_2 n \rceil < n \\ 0 \le k \le \lceil n/2 \rceil \end{cases}$$

$$\begin{cases} \lceil \log_2(n+1) \rceil + (n-k) \lceil \log_2 n \rceil < n \\ \lceil n/2 \rceil < k \le n \end{cases},$$

Daher finden wir, dass das erste System der Ungleichungen gilt, wenn

$$0 \le k < \frac{n}{\lceil \log_2 n \rceil} - \frac{\lceil \log_2(n+1) \rceil}{\lceil \log_2 n \rceil}$$

und das zweite, wenn

$$n - \frac{n}{\lceil \log_2 n \rceil} + \frac{\lceil \log_2 (n+1) \rceil}{\lceil \log_2 n \rceil} < k \le n$$

Erstens, berücksichtigen wir, dass für n - nicht ganzzahlige Potenzen von Zwei: $\lceil \log_2 n \rceil = \lceil \log_2 (n+1) \rceil$,

zweitens, wenn n ziemlich groß ist:

$$\frac{\lceil \log_2 (n+1) \rceil}{\lceil \log_2 n \rceil} \approx 1$$

Systemlösungen können dann mit einer relativ kleinen Abweichung dargestellt werden als:

$$0 \le k < \frac{n}{\lceil \log_2 n \rceil} - 1 \quad , \qquad n - \frac{n}{\lceil \log_2 n \rceil} + 1 < k \le n$$

Durch die Kombination beider Ungleichungen erhalten wir die allgemeine rationale Codierungsbedingung:

$$\begin{cases} 0 \le k < \dfrac{n}{\lceil \log_2 n \rceil} - 1 \\ n - \dfrac{n}{\lceil \log_2 n \rceil} + 1 < k \le n \end{cases} \qquad (2.8)$$

Im Hinblick auf ihre Sperrigkeit bezeichnen wir

$$\alpha = \frac{n}{\lceil \log_2 n \rceil},$$

dann wird der Ausdruck (2.8) in die Form konvertiert:

$$\begin{cases} 0 \le k < \alpha - 1 \\ n - \alpha + 1 < k \le n \end{cases} \qquad (2.9)$$

Wenn daher der Wert von k die Bedingung (2.9) erfüllt, werden alle Codeelemente der entsprechenden äquivalenten Klasse $A_k$ der Quelle Y in binäre Folgen von k - Adressen umgewandelt. Wenn in diesem Fall die erste Ungleichung (2.9) erfüllt ist, die als untere Kompressionsbedingung bezeichnet wird, erfolgt eine Transformation durch die Adressen von k-logischen Einsen, und falls die zweite Ungleichung erfüllt ist, die als obere Kompressionsbedingung bezeichnet wird, erfolgt die Transformation durch die Adressen der (n-k)-logischen Nullstellen. Wenn die Zahl k die Bedingung (2.9) nicht erfüllt, bleiben die ursprünglichen n-Bit-Vektoren unverändert.

D*efinition*. Das Codierungsverfahren $f_{av}$: $A \rightarrow B$, - wobei $A = \{a_i : |a_i| = n, i = 1,..., |A|\}$ -eine ursprüngliche und $B = \{f_{av}(a_i); i = 1,..., |A|\}$ - eine resultierende Menge von binären Wörtern, - das im Übergang vom Vektor- zum Adressierungscodierungsverfahren in Abhängigkeit von der Anzahl k-Einsen im

Quellwort gemäß Bedingung (2.9) besteht, wird als Verfahren der Adressvektorkompression bezeichnet, und der resultierende Code wird der Vektoradresscode genannt, dessen Elemente gemäß der Regel gebildet werden:

$$f_{av}(a_i) = \begin{cases} \{\mathrm{Bin}\, k, VU(a_i)\}, 0 \le k < \alpha - 1 \\ \{\mathrm{Bin}\, k, VZ(a_i)\}, n - \alpha + 1 < k \le n \\ \{\mathrm{Bin}\, k, a_i\}, a - 1 \le k \le n - \alpha + 1 \end{cases} \quad (2.10)$$

Wobei $VU(a_i)$ und $VZ(a_i)$- Adressvektoren von logischen Einsen und Wortnullen $a_i \in A$.

Der Binärdatensatz $\mathrm{Bin}\ k$ spielt die Rolle eines Servicewortes, das Informationen über die Anzahl der $k$ - logischen Einsen des Quellworts enthält, sodass wir Folgendes bestimmen können:

1)Codierungsverfahren einer binären Vektor- oder Adressekombination;

2) Ihren Anfang und ihre erwartete Länge.

**Satz 3.** Die Adressvektorkodierung $f_{av}$: $A \to B$ hat die Eigenschaften der Surjektivität und der Injektivität.

**Satz 4.** Der Adressvektorcode ist ein dekodierender Code.

**Satz 5.** Relative Redundanz der Adressvektorkomprimierungsmethode $f_{av}$ des Quellens $Y = \{a_j \in A_k, j = 1,..., C_n^k\}$, wobei $n$-Bit-Binärwörter $a_j$ mit einer konstanten Anzahl $k$-Einsen erzeugt werden:

$$r(f_{av}, k) \le 1 - \frac{\log_2 C_n^{[\alpha]}}{n} \quad (2.11)$$

wobei

$$\alpha = \frac{n}{\lceil \log_2 n \rceil}.$$

Gemäß (2.2, 2.5) und der Bedingung der Adressvektorkodierung der Äquivalenzklasse $A_k$:

$$\begin{cases} 0 \le k < \alpha \\ n - \alpha < k \le n \end{cases}$$

ist die relative Redundanz der Adressvektor-Codierung auf der Menge $A_k$:

$$r(f_{av}, k) = \begin{cases} 1 - \dfrac{\log_2 C_n^k}{k\lceil \log_2 n \rceil}, 0 < k < \alpha \\ 1 - \dfrac{\log_2 C_n^k}{n}, a \le k \le n - \alpha \\ 1 - \dfrac{\log_2 C_n^k}{(n-k)\lceil \log_2 n \rceil}, n - \alpha < k < n \end{cases} \quad (2.12)$$

In ***Abb. 4 (Anhang D)*** ist der Graph $r(f_{av}, k)$ für das Adressvektor-Codierungsverfahren bei $n=256$, - dessen Werte in der Tabelle 2.3 zusammengefasst sind, - dargestellt.

Die Verwendung der Adresscodierung zusammen mit der Vektorcodierung impliziert das Vorhandensein einer Adressvektordatenkomprimierung in dem simulierten System einer zusätzlichen Quellenadresse $Z=\{\text{Bin } z,\ z=0,...,n\}$. Die Umschaltung zwischen den Quellen Y und Z hängt vom k -Wert ab, der von der Quelle X erzeugt wird und durch einen Kommutator ausgeführt werden muss. Hier gibt es die folgende Logikschalterfunktion:

$$F = \bar{u}\cdot Y + u\cdot Z, \tag{2.13}$$

wobei ist u eine logische Funktion der Bedingung (2.9).

Tabelle 2.3: Funktionswerte $r(f_{av}, k)$ der Adressenvektorcodierung der Äquivalenzklasse $A_k$ in Abhängigkeit von k

| K | $r(f_{av}, k)$ |
|---|---|
| 1 | 0,00 |
| 16 | 0,35 |
| 32 | 0,47 |
| 48 | 0,32 |
| 64 | 0,20 |
| 80 | 0,12 |
| 96 | 0,06 |
| 112 | 0,03 |
| 128 | 0,02 |
| 144 | 0,03 |
| 160 | 0,06 |
| 176 | 0,12 |
| 192 | 0,20 |
| 208 | 0,32 |
| 224 | 0,47 |
| 240 | 0,35 |
| 255 | 0,0 |

Wenn wir $u = 1$ nehmen und die Bedingung (2.9) für die Komprimierung erfüllt ist, können wir das Folgende beschreiben:

$$u = \bigcup_{k=0}^{[\alpha-2]} m_k + \bigcup_{k=[n-\alpha+2]}^{n} m_k = \bigcup_{\substack{k<\alpha-1,\\ k>n-\alpha+1}} m_k$$

wobei $m_k$ ein binäres Argument ist, das der Zahl k entspricht und dessen Anwesenheit oder Abwesenheit anzeigt.

Zur Dekodierung der Adress-Vektor-Code-Kombinationen müssen die entsprechenden Binärsätze Bin k analysiert werden, nach deren Ergebnis die Kodierungsmethode bestimmt werden. Im Falle der Adress-Kodierungsmethode - adressierbare Logikebene. Die Rücktransformation zu den Originalwörtern wird durchgeführt. Folglich muss das System der Adressvektorkomprimierung auf der Empfangsseite die Analysatornummer k und den inversen Wandler enthalten.

Abb. 5 (Anhang E) stellt das Strukturmodell des Systems der Adressvektorkomprimierung als Teilsystem des automatisierten Steuerungssystems dar. Das automatisierte Steuerungssystem besteht aus einer Quelle $A=\{a_i: |a_i|=n, i=1,...,|A|\}$, die in die Quelle $X=\{0,1,...,n\}$ von Vorzeichen k umgewandelten ist, aus einer Quelle $Y=\{a_j \in A_k, j=1,..., C_n^k\}$ von Binärwörtern $a_j$ der Äquivalenzklasse $A_k$, aus einer zusätzlichen Quelle $Z=\{\text{Bin } z, z=0,...,n\}$ von Adressen, aus einem von der Quelle X gesteuerten Schalter, aus einem Analysator der Zahl k, einem Wandler für die Quellwörter $a_i$ und einem Informationsempfänger.

Die Erzeugung einer Nachricht durch die Quelle $A$ kann als Auswahl der Klasse $A_k$ auf der Grundlage der Äquivalenz k und ferner als Suche und Erzeugung des zu dieser Klasse gehörenden binären Wortes $a_i$ dargestellt werden. Der Wert von k ist durch die Quelle der X-Indikationen gegeben. Gleichzeitig mit der Erzeugung $a_i$ erzeugt die zusätzliche Quelle Z eine Folge von k-Adressen, die die Position der Einsen im ursprünglichen Vektor angeben. Abhängig vom k-Wert wird die Quelle der Y-Elemente der Äquivalenzklasse oder die entsprechende Quelle der Z-Adressen durch einen Schalter mit dem Kommunikationskanal verbunden.

Die Quellenumschaltung wird durch Bedingung (2.9) eingestellt. Zusammen mit dem Vektor $a_i$ oder der Folge von Adressen $VU(a_i)$ oder $VZ(a_i)$ wird die Binärzahl Bin k über den Kommunikationskanal übertragen. Nach Erhalt wird der Äquivalenzzeichen k vom Analysator verarbeitet und die logische Ebene der Adressen und die erwartete Länge der Informationsnachricht analisiert. Nach der Analyse des Ergebnises bringt der Wechselrichter die empfangene Nachricht in die ursprüngliche Form $a_i$, wonach sie an den Informationsempfänger geht.

Da die Anzahl der Vektoren mit der gleichen Anzahl $C_n^k$ von Einsen gleich ist und die Wahrscheinlichkeit ihrer Erscheinung $p(a_i)=p^k(1-p)^{n-k}$ ist, werden die Kosten der vektoriellen Vektorkomprimierung $f_{av}$ für Quelle $A$ auf der Basis von (2.9, 2.10) in der folgenden Form berechnet:

$$C(f_{av}, A) = \sum_{k=0}^{\alpha-2} C_n^k p^k (1-p)^{n-k} (k+1)\lceil \log_2 n \rceil + \sum_{k=\alpha-1}^{n-\alpha+1} C_n^k p^k (1-p)^{n-k} (n+\lceil \log_2 n \rceil) + \sum_{k=n-\alpha+2}^{n} C_n^k p^k (1-p)^{n-k} (n-k+1)\lceil \log_2 n \rceil \quad (2.14)$$

Und wenn $P_\kappa = C_n^k p^k (1-p)^{n-k}$ ist die Wahrscheinlichkeit der Erscheinung von k-Einsen, gilt das Folgende:

$$C(f_{av}, A) = \sum_{k=0}^{\alpha-2} P_k (k+1)\lceil \log_2 n \rceil + \sum_{k=\alpha-1}^{n-\alpha+1} P_k (n + \lceil \log_2 n \rceil) + \\ + \sum_{k=n-\alpha+2}^{n} P_k (n-k+1)\lceil \log_2 n \rceil \tag{2.15}$$

Redundanz der Adressvektorkomprimierung $f_{av}$ für Quelle $A$ unter Verwendung des Ausdruckes (2.15):

$$R(f_{av}, A) = \sum_{k=0}^{\alpha-2} P_k (k+1)\lceil \log_2 n \rceil + \sum_{k=\alpha-1}^{n-\alpha+1} P_k (n + \lceil \log_2 n \rceil) + \\ + \sum_{k=n-\alpha+2}^{n} P_k (n-k+1)\lceil \log_2 n \rceil - \\ - \left( \sum_{k=0}^{n} P_k \log_2 \frac{1}{P_k} + \sum_{k=0}^{n} P_k \log_2 C_n^k \right) \tag{2.16}$$

Zusammenfassend der obigen Eigenschaften des Adressvektorkomprimierungsverfahrens können wir daher Folgendes schließen:

1. Das Adressvektor-Codierungsverfahren ermöglicht es, die Redundanz zu reduzieren und somit binäre Nachrichten zu komprimieren, wenn sie von logischen Einsen oder Nullen dominiert werden.
2. Das Adressvektorkomprimierungsverfahren basiert auf der Darstellung der Informationsquelle als Gesamtkomplex miteinander verbundener Quellen: aus einer Quelle binärer Einsen und einer Quelle von Elementen der Äquivalenzklasse, was wiederum zu einer Verkürzung der Zeit für die vorläufige Bewertung von Informationen zum Zweck ihrer Komprimierung führt.
3. Die in der Adressvektormethode verwendeten Prozeduren, wie das Berechnen von Binäreinsen, das Auswählen der Codierungsmethode und das Erzeugen von Adressen, sind aus Sicht der praktischen Implementierung ziemlich einfach, was möglicherweise zu einer hohen Geschwindigkeit der Komprimierungs- und Reparaturalgorithmen führt.

## 2.2 Der allgemeine Algorithmus zur Adress-Vertorkomprimierung und -Wiederherstellung von Binärinformationen

### 2.2.1 Der allgemeine Algorithmus zur Adressvektorkomprimierung von Binärinformationen

Der Algorithmus ist ein strenges Regelsystem, das die Reihenfolge Operationen über bestimmte Objekte festlegt. Entsprechend der Definition von Algorithmus muss er folgende Eigenschaften haben:

1. Endlichkeit.
2. Wirksamkeit der verwendeten Operationen.
3. Eingabe und Ausgabe haben.

Von allen möglichen sind die schnellsten Algorithmen zum Codieren und Decodieren von binären n-Bit-Wörtern Algorithmen, die vollständige Nachschlagetabellen verwenden. Sie erhalten das Ergebnis während nur einer einzelnen Operation, aber die Kosten für eine solche Geschwindigkeit ist eine verschwenderische Speichernutzung ($2^n$-Zellen). Und es macht solche Algorithmen bei großen n -Werten praktisch ungeeignet.

Daher ist es angesichts der bestehenden Beschränkungen der Software- und Hardwarekosten erforderlich, Algorithmen für die Adressvektorcodierung und -decodierung zu finden, zu untersuchen und zu implementieren. Diese Algorithmen sollen in algorithmische Software eines automatisierten Kontrollsystems enthalten und ausreichend schnell sein und keine große Speicherkapazität erfordern.

Falls der Adressvektorkomprimierung wird der Algorithmus durch die mathematische Funktion $f_{av}$ (2.10) bestimmt. Diese Funktion übereinstimmt eine binäre n-Bit-Sequenz $a_i$ eindeutig mit einem entsprechenden Adressvektor-Codewort $f_{av}(a_i)$variabler Länge.

Der allgemeine AVK-Algorithmus, der die Adressvektorcodierung durchführt $f_{av}$: $A \rightarrow$ B, wobei $A=\{a_i: / a_i / = n, \ i=1,..., / A / \}$ - eine ursprüngliche Informationsquelle ist, $B=\{f_{av}(a_i), \ i=1, \ ..., \ / A / \}$ - eine resultierende Menge von Adressvektor-Codewörtern, sieht folgendermaßen aus:

1. Empfang und Anfangsdeskription eines binären Eingangs-n-Bit-Vektors $a_i$ in einen Eingangsregister.

2. Berechnung der Anzahl von k-Einsen im Vektor $a_i$.

3. Falls die Anzahl k der logischen Einsen nicht dem Ungleichungssystem (2.9) erfüllt, bleibt der Binärvektor unverändert und springt es zum Schritt 7. über. Andernfalls geht es zum nächsten Schritt weiter.

4. Wenn die Anzahl der k-Einheiten größer oder gleich 0 und kleiner als die untere Kompressionsbedingung (2.9) ist, springt es zum Schritt 6. über. Andernfalls geht es zum nächsten Schritt weiter.

5. Bildung des Adressvektors $VZ(a_i)$ von logischen Nullen und weiter zum Schritt 7.

6. Bildung des Adressvektors $VU(a_i)$ von logischen Einsen und weiter zum Schritt 7.

7. Bildung eines Serviceworts Bin k.

8. Das Adressvektor-Codewort $f_{av}(a_i)$ wird in das Ausgangsregister übergetragen und der Algorithmus wird gestoppt.

Das Blockschaltbild des allgemeinen Algorithmus von AVK ist in Abb. 6 (Anhang F) dargestellt.

Bei diesem Algorithmus ist die Kodierungszeit oder die Anzahl von ausgeführten Operationen von oben durch die Länge n des Quellbinärworts begrenzt. Als Eingabe hat der AVK-Algorithmus ein ursprüngliches Wort $a_i \in A$ und als Ausgabe ein Adressvektorwort $f_{av}(a_i) \in B$.

Dieser Komprimierungsalgorithmus zeichnet sich durch Bestimmtheit und Effizienz von verwendeten Operationen aus, da er erstens eine endliche Menge relativ einfache mathematische Operationen enthält und zweitens alle Operationen mit Primzahlen ausgeführt werden.

Lassen wir die Korrektheit des AVK-Algorithmus sich anhand der „Proof-by-Exhaustion“-Methode belegen, d.h. alle möglichen Fälle seiner Funktionsweise in Abhängigkeit von der Art von Eingabedaten auflisten. Dazu muss natürlich die Gültigkeit der wichtigsten Schritte 3,4 und 5 ermittelt werden.

Als Ergebnis von Schritt 3 wird einer der drei Übergänge zum Schritt 4, 5 oder 6 ausgeführt. Diese Übergänge stellen erstens eine vollständige Gruppe von Ereignissen dar, zweitens jeder Übergang entspricht der Operation, die gemäß der Funktion (2.10) der Adressvektorkomprimierung ausgeführt wird.

Tatsächlich Schritt 4. Bildung Adressen von Einsen, - wird nur dann durchgeführt, wenn die k - Anzahl die untere Bedingung (2.9) erfüllt. Schritt 5. Bildung Adressen von Nullen, - nur wenn die Zahl k die obere Bedingung (2.9) erfüllt. Und die Übersprung zum Schritt 6. passiert ohne die Schritte 4 oder 5 erfolgt ausführt sind im Falls, wenn k keine der Bedingungen erfüllt, wie die Definition des Adress-Vektorkompression fordert.

### 2.2.2 Der allgemeine Algorithmus zur Wiederherstellung von Binärinformationen

Falls der Adressvektor-Decodierung wird der entsprechende Algorithmus durch die mathematische Funktion $g=[f_{av}(a_i)]$ bestimmt, die die Inverse der Funktion $f_{av}(a_i)$ ist.

Der *a*llg*e*m*ei*n*e* *A*VD-*A*lgor*i*thmus, der die Adressvektor-Decodierung durchführt, $g_{av}$: $B \rightarrow A$, wobei $B=\{f_{av}(a_i);\ i=1,.../A/\}$ ist, die ursprüngliche Menge von Adressvektor-Codewörtern, $A=\{a_i:/a_i/=n,\ i=1,.../A/\}$ ist die resultierende Menge von Binärvektoren:

1. Anfangseinstellung der Eingangs- und Ausgangsregister auf Null und der Empfang eines ursprünglichen Adressvektorworts $f_{av}(a_i)$ vom Eingangsregister.
2. Analyse des Serviceworts Bin k.
3. Wenn die Anzahl k der logischen Einsen 0, ist, dann ist der Nullinhalt des Ausgangsregisters der resultierende Binärvektor $a_i$ und geht es zum Schritt 10 über.
4. Wenn die Anzahl k der logischen Einsen n ist, geht es zum Schritt 9 weiter.
5. Wenn k das Ungleichungssystem (2.9) nicht erfüllt, springt es zum nächsten Schritt über. Andersfalls geht es zum Schritt 7 fort.
6. Der Inhalt des Eingangsregisters wird ohne das Servicewort Bin k in das Ausgangsregister übergetragen und weiter geht es zum Schritt 10 fort.
7. Entsprechend der Adressen des Vektors $VU(a_i)$ oder des Vektors $VZ(a_i)$ werden Bits im Ausgangsregister gemäß den entsprechenden Adressennummern mit logischen Einsen und alle anderen Bits mit Nullen gefüllt.
8. Wenn die Anzahl k der logischen Einsen kleiner als die untere Komprimierungsbedingung (2.9) ist, gehen wir beim Schritt 9 fort.
9. Der Inhalt des Ausgangsregisters wird invertiert.
10. Der Algorithmus wird gestoppt.

Das Blockschaltbild des allgemeinen *A*VD-Algorithmus ist in Abb.7 (Anhang G) dargestellt.

Bei diesem Decodieralgorithmus ist die Laufzeit oder die Anzahl der Ausführungszyklen oben durch die Anzahl der Adressen der Vektoren $VU(a_i)$ und $VZ(a_i)$ begrenzt, die nicht mehr als $\alpha$-1 sein können. Als Eingabe hat der *A*BD-

Algorithmus ein binären Adressvektorwortes $f_{av}(a_i) \in B$ und als Ausgabe ein in seine ursprüngliche Form umgewandelten binären Wort $a_i$.

Der betrachtete Algorithmus hat auch Bestimmtheit und Effizienz von verwendeten Operationen, da er erstens eine endliche Menge von ziemlich einfachen mathematischen Operationen enthält und zweitens alle Operationen mit Primzahlen ausgeführt werden.

Bevor wir mit der Begründung des *A*VD-Algorithmus anfangen, lassen wir den folgenden Satz bewiesen werden.

**Satz 12**. Es gibt eine Adressvektorabbildung $f_{av}: A \to B$ und das Vorbild eines binären Vektors $a_i$ sei der Vektor $VU(a_i)$ von Einsen-Adressen, und das Vorbild des Vektors $a_j$ ist der Vektor $VZ(a_j)$ von Nullen-Adressen, wobei $|a_i| = |a_j| = n$, und $i,j = 1,.../A/$ sind. Wenn $VU(a_i) = VZ(a_i)$, dann ist $a_i = \overline{a}_j$ und umgekehrt, wenn $a_i = \overline{a}_j$, dann ist $VU(a_i) = VZ(a_i)$.

**Beweis**. Beweisen wir durch Widerspruch. Nehmen wir an, dass die obigen Bedingungen erfüllt werden, dann sei es $a_i \neq \overline{a}_j$. Und da $VU(a_i) = VZ(a_i)$ sind, können sich die Binärwörter $a_i$ und $\overline{a}_j$ nur dann unterscheiden, wenn ihre Längen unterschiedlich sind: Ausgehend aus der Anweisungsbedingung: $|a_i| \neq |\overline{a}_j|$ sind und die Inversionsoperation ändert jedoch nicht die Länge des Binärworts. Daher ist $|a_i| = |a_j| = |\overline{a}_j|$. Unsere Annahme ist also falsch ist und dann $a_i = \overline{a}_j$ ist. Der Beweis und die umgekehrte Aussage werden auf ähnliche Weise ausgeführt, wobei die Gleichheit der Längen der binären Wörter $a_i$ und $a_j$ berücksichtigt wird.

Um die Korrektheit des Dekodierungsalgorithmus zu belegen, verwenden wir dieselbe Methode „Proof-by-Exhaustion“. Wenn das Servicewort Bin k in das Eingaberegister eintritt, wobei k=0 ist, springt der Algorithmus aus dem Schritt 3 zum Schritt 9 über und stoppt.

In diesem Fall enthält das Ausgangsregister (zuvor in Schritt 1 auf Null gesetzt) einen binären Vektor von Nullen, der der Funktion $g_{av}$ der Adressvektor-Decodierung entspricht. Im Falle des Empfangs des Serviceworts Bin k, wobei k=n, wird von dem Schritt 4 die Führung zum Schritt 8 übergegeben, die die Invertierung des Ausgangs-Null-Registers durchführt.

Infolgedessen erscheint am Ausgang ein Vektor, der aus einigen logischen Einheiten besteht, was gemäß der Dekodierungsgrafik erforderlich ist. Wenn ferner ein Wort des Typs fav $(a_i)$ = {Bin k, $a_i$} decodiert wird, schreibt der Algorithmus gemäß Schritt 5 den Inhalt der Eingabe in die Ausgabe um und stoppt.

In diesem Fall enthält das Ausgangsregister das Binärwort $a_i$, das benötigt wird. Wenn der Eingabewert des *A*VD-Algorithmus irgendein Adressvektorwort vom Typ $f_{av}(a_i)=\{Bin\ k, VU(a_i)\}$ oder $f_{av}(a_i)=\{Bin\ k, VZ(a_i)\}$ ist, dann nach dem Ausführen der Schritte 1-5 die Führung wird an den Schritt 6 übergegeben. Im Schritt 6 werden die logischen Einsen in die Bits des Ausgangsregisters geschrieben, die durch die Adressen des Vektors $VU(a_i)$ oder $VZ(a_i)$ angegeben sind.

Die restlichen Ziffern (Nullen nach der Erstinstallation) bleiben unverändert. Die Zuverlässigkeit dieses Ergebnisses wird durch die Funktion $g_{av}=[f_{av}(a_i)]$ und die Anweisung 12 gerechtfertigt. Nachdem wir also alle möglichen Eingabedaten durchgegangen sind, schließen wir daraus, dass der Decodierungsalgorithmus korrekt funktioniert.

Gemäß der strukturierten Programmiermethode setzt die Weiterentwicklung der allgemeinen *A*VK- und *A*VD-Algorithmen voraus, dass jeder Schritt ein eigenes Regelwerk hat. Die nachfolgende algorithmische Strukturierung hängt von den Anforderungen an die Komplexitätseigenschaften ab: der Anzahl der Ausführungsschritte, der Länge des Programms und der Menge der verbrauchten Speichkapazität.

Unter dem Gesichtspunkt der Komplexität und der eingeführten Beschränkungen besteht das größte Interesse darin, die Anzahl der logischen Einsen und die Bildung von Adressen zu zählen - Schritte 2 und 6 des Codierungsalgorithmus und die Wiederherstellung des Binärvektors durch Adressen - Schritt 7 des Decodierungsalgorithmus.

## FAZIT

In dieser Studie wurde eine Literaturrecherche durchgeführt, wobei Fragen im Zusammenhang mit Informationskomprimierung behandelt werden, und zwar:

- Definierte Grundkonzepte;
- In Abhängigkeit davon,- ob Datenkomprimierungsmethoden einen Informationsverlust enthalten,- wurde eine Klassifizierung von Datenkomprimierungsmethoden durchgeführt.
- Das Prinzip einiger bekannten Datenkomprimierungsmethoden wurde betrachtet und beschrieben.
- Hauptkriterien für die Bewertung der Kompressionsqualität sind definiert;
- Methodenvergleich und Analyse von Qualitätsbewertungskriterien.

Im Hinblick auf die Literaturrecherche, in der Merkmale von modernen Telekommunikationssystemen untersucht wurden, kann man das Folgende schließen. Eine der effektivsten Methode, - um Bandbreite von Informationsquellen und Telekommunikationssysteme im Allgemeinen zu erhöhen, - ist die Verwendung von Kompressionstechniken und Hardware- oder Softwarekomponenten, die diese Techniken in Telekommunikationssysteme implementieren.

Bei relativ geringen Kosten einer solchen Implementierung ist es möglich, die durchschnittliche Geschwingichkeit von Datenübertragung signifikant zu erhöhen. Aus dieser Sicht vielversprechend ist die Verwendung der *AV*-Komprimierung, die die folgenden positiven Eigenschaften hat:

- Einfachheit ihrer technischen Umsetzung;
- Hohe Kompressionsgeschwindigkeit und Wiederherstellungsgeschwindigkeit von übertragenden Informationen;
- Kein Informationsverlust bei Datenkonvertierung;
- Allgemeingültigkeit für alle verarbeitenden Datenarten;

In dieser Bachelor Arbeit sind folgende Aufgaben im Forschungsteil erledigt:

1. Es wurde einen verallgemeinerten Algorithmus zum Informationscodieren und -Decodieren nach der *AV*-Komprimierungsmethode zur Verwendung in Telekommunikationssystemen entworfen;
4. Es wurde eine Analyse von wichtigsten Parameter der Effizienz der Algorithmen zur *AV*-Komprimierung und *AV*-Wiederherstellung und denen Auswertung durchgeführt.

Im technischen Teil dieses Projekts sind das Folgende entworfen:

- Konstruktion eines Strukturmodells eines Telekommunikationssystems mit Anwendung der *AV*-Komprimierung;
- Entwicklung eines Funktionsschemas eines Telekommunikationssystems unter Verwendung der *AV*-Komprimierung, die (System) einen allgemeinen Charakter von praktischer Implementierung haben muss.

# QUELLENVERZEICHNIS


1. „Datenkompression“ - Maciej Li´skiewicz[1], Henning Fernau[2] Berlin 2014; [1)]Institut für Theoretische Informatik, Medizinische Universität zu Lübeck Wallstr. 40, D-23560 Lübeck und [2)]Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübinge.
2. „Definition.Was ist Datenkompression?“ - Autor / Redakteur: Tina Billo / Rainer Graefen; 08.10.18; https://www.storage-insider.de/was-ist-datenkompression-a-764167/
3. Tilo Strutz: Bilddatenkompression – Grundlagen, Codierung, Wavelets, JPEG, MPEG, H.264, HEVC. Springer Vieweg, Wiesbaden 2017, ISBN 978-3-8348-1427-2, S. 421.
4. Matthew V. Mahoney: Fast Text Compression with Neural Networks. In: AAAI (Hrsg.): Proceedings of the Thirteenth International Florida Artificial Intelligence Research Society Conference. 2000, ISBN 1-57735-113-4, S. 5.
5. „Verlustfreie Kompression in rekonfigurierbarer Hardware. Diplomarbeit“ - Herr Maximilian Buder; 23. Oktober 2007
6. Stefan Brunthaler: Quellen- und Leitungscodierung. (PDF; 528 KB) In: TH Wildau. 2018, abgerufen am 12. August 2018 (Vorlesungsscript Kommunikationstechnik in der Telematik SS2018).
7. Peter Maluck, Jürg Scheidegger: Quellencodierung – Gelenktes Entdeckendes Lernen. (PDF; 776 KB) In: SwissEduc. 24. August 2009, abgerufen am 12. August 2018 (Seminar Kommunikationstechnik).
8. Bericht zum Seminar „Informatik für Biologen“, „Datenkompression“ - Andreas Wieland; Curando. Universität ULM. Sciendo Docendo.
9. www.wikipedie.de

# ANHANG A

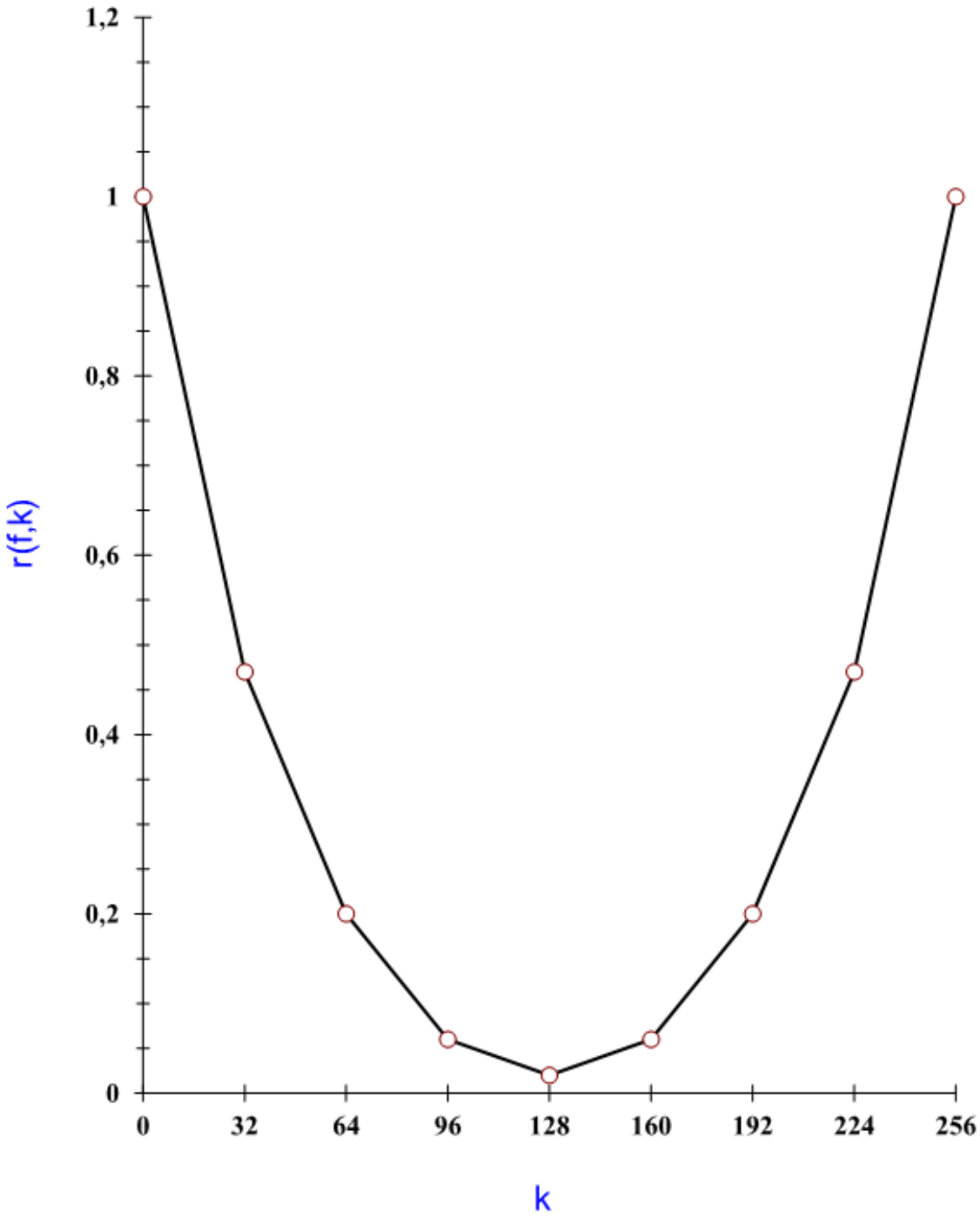



Abb. A: Graph der relativen Redundanz $r(f_v, k)$ der Vektorkodierung der Äquivalenzklasse $A_k$ in Abhängigkeit von k Einsen.

| | | | | | |
|---|---|---|---|---|---|
| | | | | | ЕлІТ 6.050903.263 ПЗ |
| Змін | Аркуш | № докум | Підпис | Дата | |
| Розробив | | Мощна І.Б. | | | Adress-Vektormethode von binärer Datenkompression. *Пояснювальна записка* |
| Перевірив | | Кулик І.А. | | | |
| Реценз. | | | | | |
| Н. Керівник | | Кулик І.А. | | | |
| Затвердж. | | Опанасюк А.С. | | | |

| Літ. | Аркуш | Аркушів |
|---|---|---|
| | | |

СумДУ, гр. ТК-51

# ANHANG B

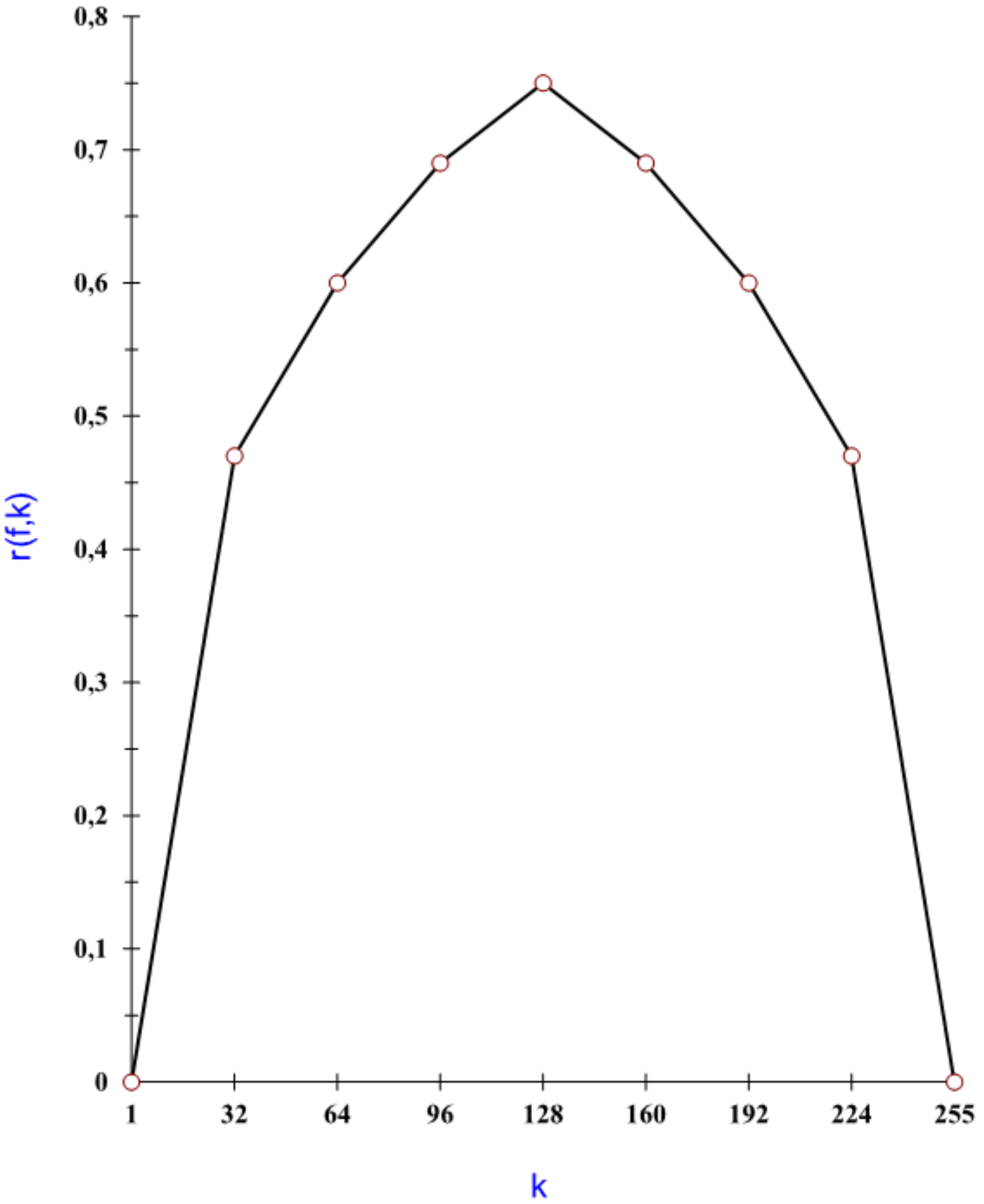



Abb. B: Graph der relativen Redundanz r ($f_a$, k) Adresscodierung der Äquivalenzklasse $A_k$ in Abhängigkeit von k Einsen.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ЕлІТ 6.050903.263 ПЗ | | |
| Змін | Аркуш | № докум | Підпис | Дата | | | |
| Розробив | | Мощна І.Б. | | | Adress-Vektormethode von binärer Datenkompression. Пояснювальна записка | Літ. | Аркуш / Аркушів |
| Перевірив | | Кулик І.А. | | | | | |
| Реценз. | | | | | | СумДУ, гр. ТК-51 | |
| Н. Керівник | | Кулик І.А. | | | | | |
| Затвердж. | | Опанасюк А.С. | | | | | |

# ANHANG C

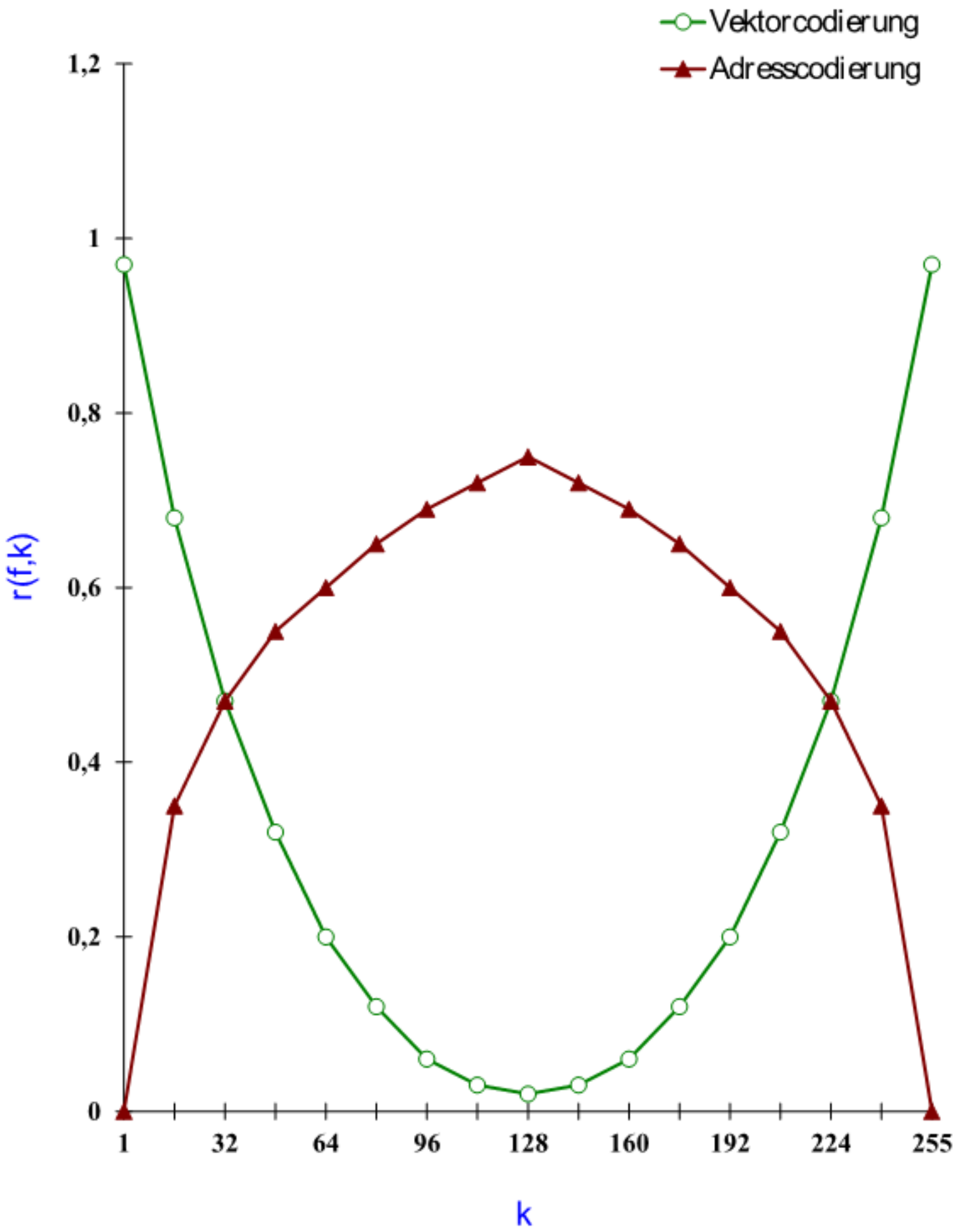



Abb. C: Funktionsgraphen $r(f_v, k)$ und $r(f_a, k)$ für die Vektor- und Adresscodierung in einem Koordinatensystem.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ЕлІТ 6.050903.263 ПЗ | | |
| Змін | Аркуш | № докум | Підпис | Дата | | | |
| Розробив | | Мощна І.Б. | | | Adress-Vektormethode von binärer Datenkompression. *Пояснювальна записка* | Літ. | Аркуш / Аркушів |
| Перевірив | | Кулик І.А. | | | | | |
| Реценз. | | | | | | СумДУ, гр. ТК-51 | |
| Н. Керівник | | Кулик І.А. | | | | | |
| Затвердж. | | Опанасюк А.С. | | | | | |

# ANHANG D

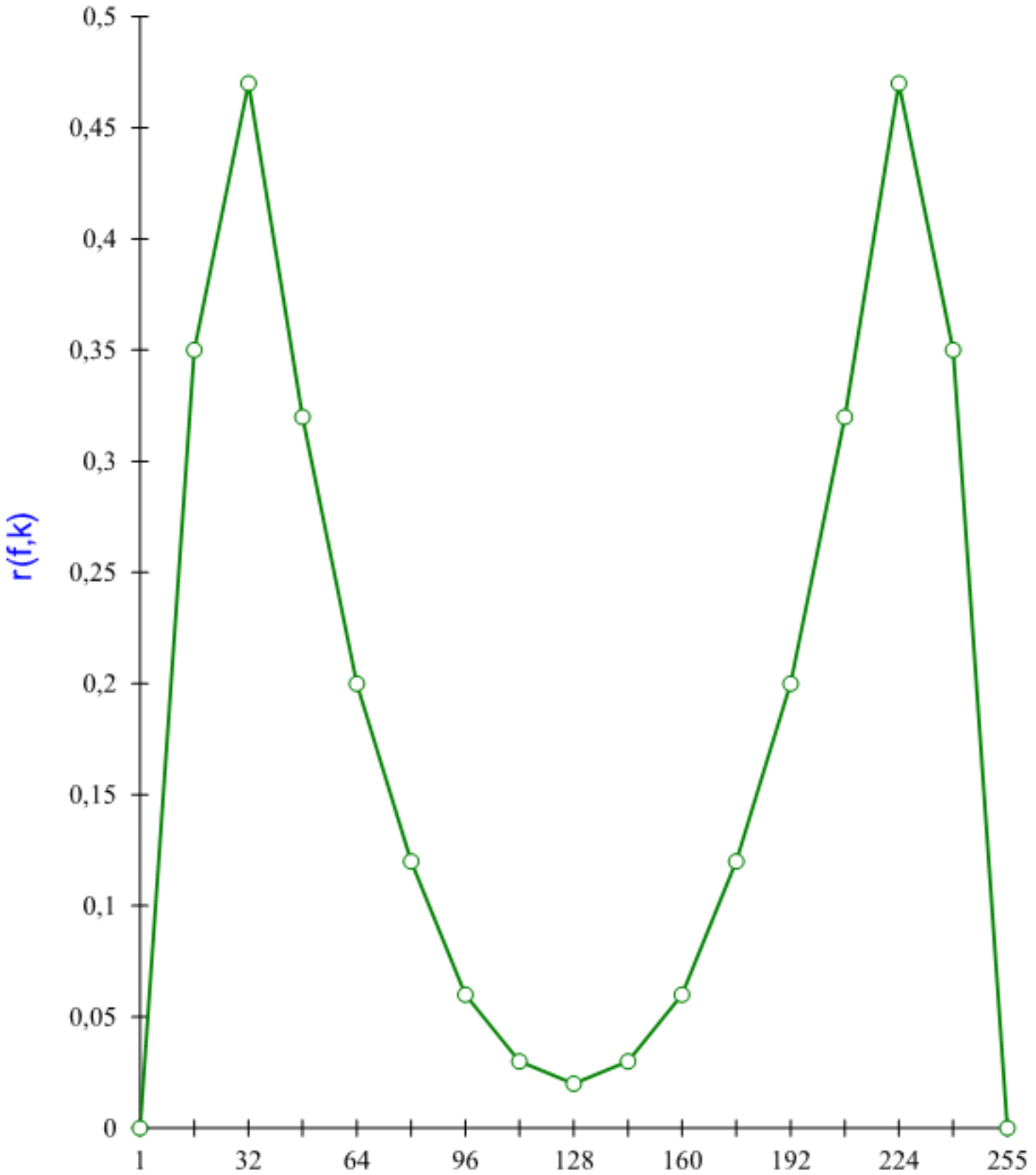



Abb. D: Der relative Redundanzgraph $r(f_{av}, k)$ der Äquivalenzklasse $A_k$ der Adressenvektorcodierung in Abhängigkeit von der Anzahl k-Einsen.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ЕлІТ 6.050903.263 ПЗ | | |
| Змін | Аркуш | № докум | Підпис | Дата | | | |
| Розробив | | Мощна І.Б. | | | Adress-Vektormethode von binärer Datenkompression. Пояснювальна записка | Літ. | Аркуш / Аркушів |
| Перевірив | | Кулик І.А. | | | | | |
| Реценз. | | | | | | СумДУ, гр. ТК-51 | |
| Н. Керівник | | Кулик І.А. | | | | | |
| Затвердж. | | Опанасюк А.С. | | | | | |

# ANHANG E

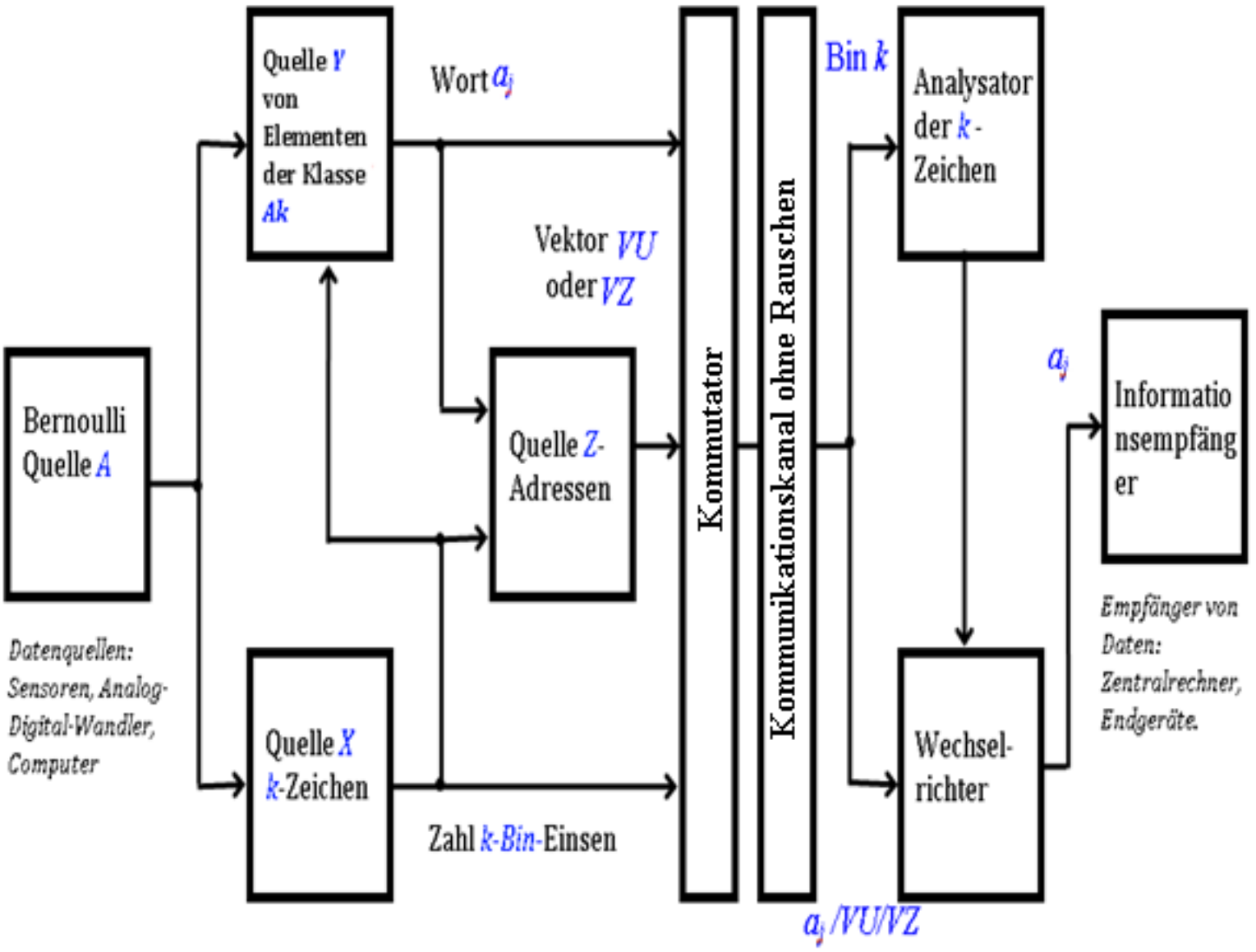



Abb. E: Strukturmodell des Systems der Adressvektorkomprimierung als Teilsystem von Automatisiertes Steuersystem.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ЕлІТ 6.050903.263 ПЗ | | |
| Змін | Аркуш | № докум | Підпис | Дата | | | |
| Розробив | | Мощна І.Б. | | | Adress-Vektormethode von binärer Datenkompression. *E1* | Літ. | Аркуш / Аркушів |
| Перевірив | | Кулик І.А. | | | | | |
| Реценз. | | | | | | ТК-51 СумДУ | |
| Н. Керівник | | Кулик І.А. | | | | | |
| Затвердж. | | Опанасюк А.С. | | | | | |

# ANHANG F

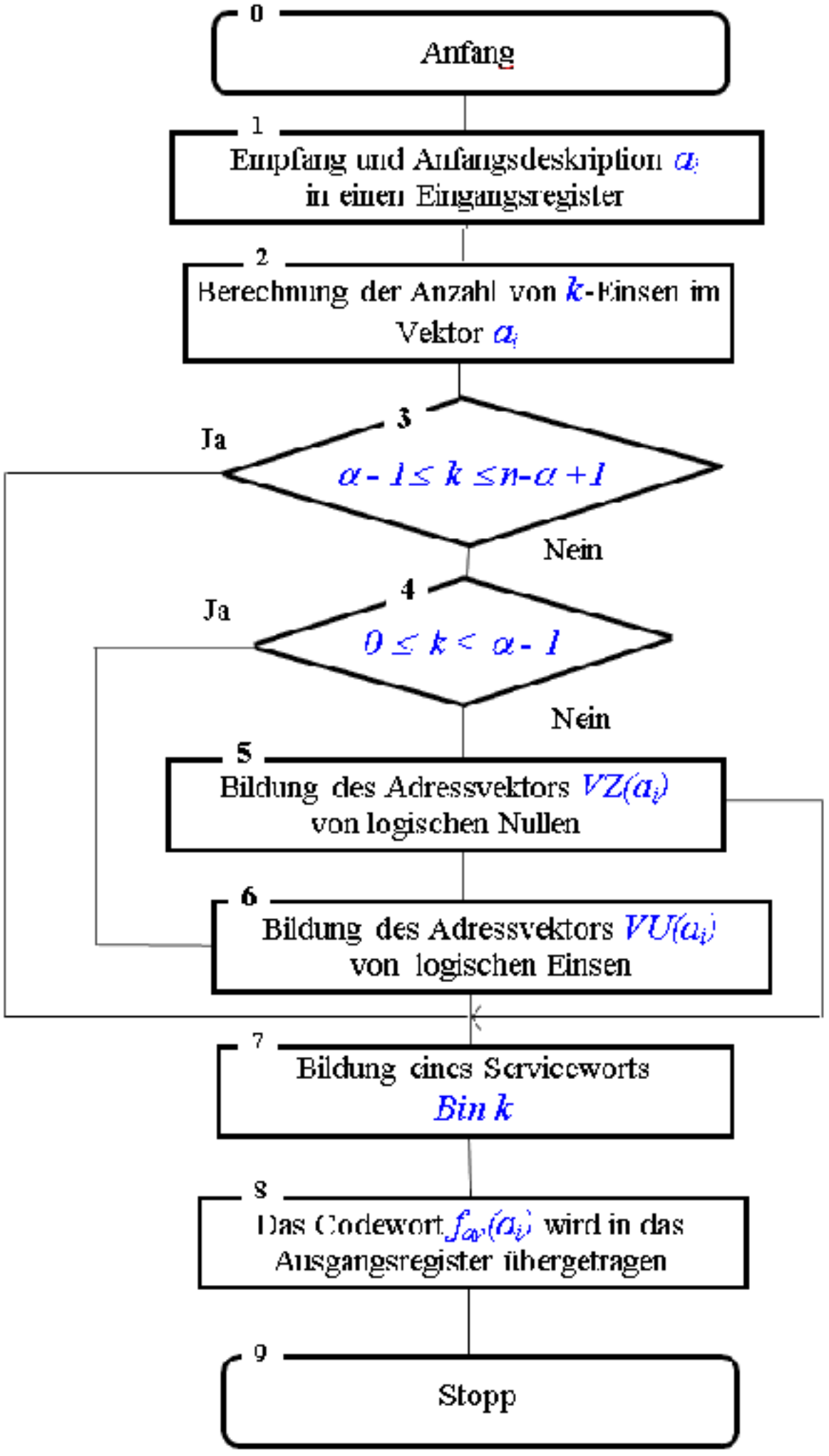



Abb. F: das Blockschaltbild des allgemeinen AVK-Algorithmus

| | | | | | |
|---|---|---|---|---|---|
| | | | | | ЕлІТ 6.050903.263 ПЗ |
| Змін | Аркуш | № докум | Підпис | Дата | |
| Розробив | | Мощна І.Б. | | | Adress-Vektormethode von binärer Datenkompression. *CA* |
| Перевірив | | Кулик І.А. | | | Літ. / Аркуш / Аркушів |
| Реценз. | | | | | |
| Н. Керівник | | Кулик І.А. | | | СумДУ, гр. ТК-51 |
| Затвердж. | | Опанасюк А.С. | | | |

# ANHANG G

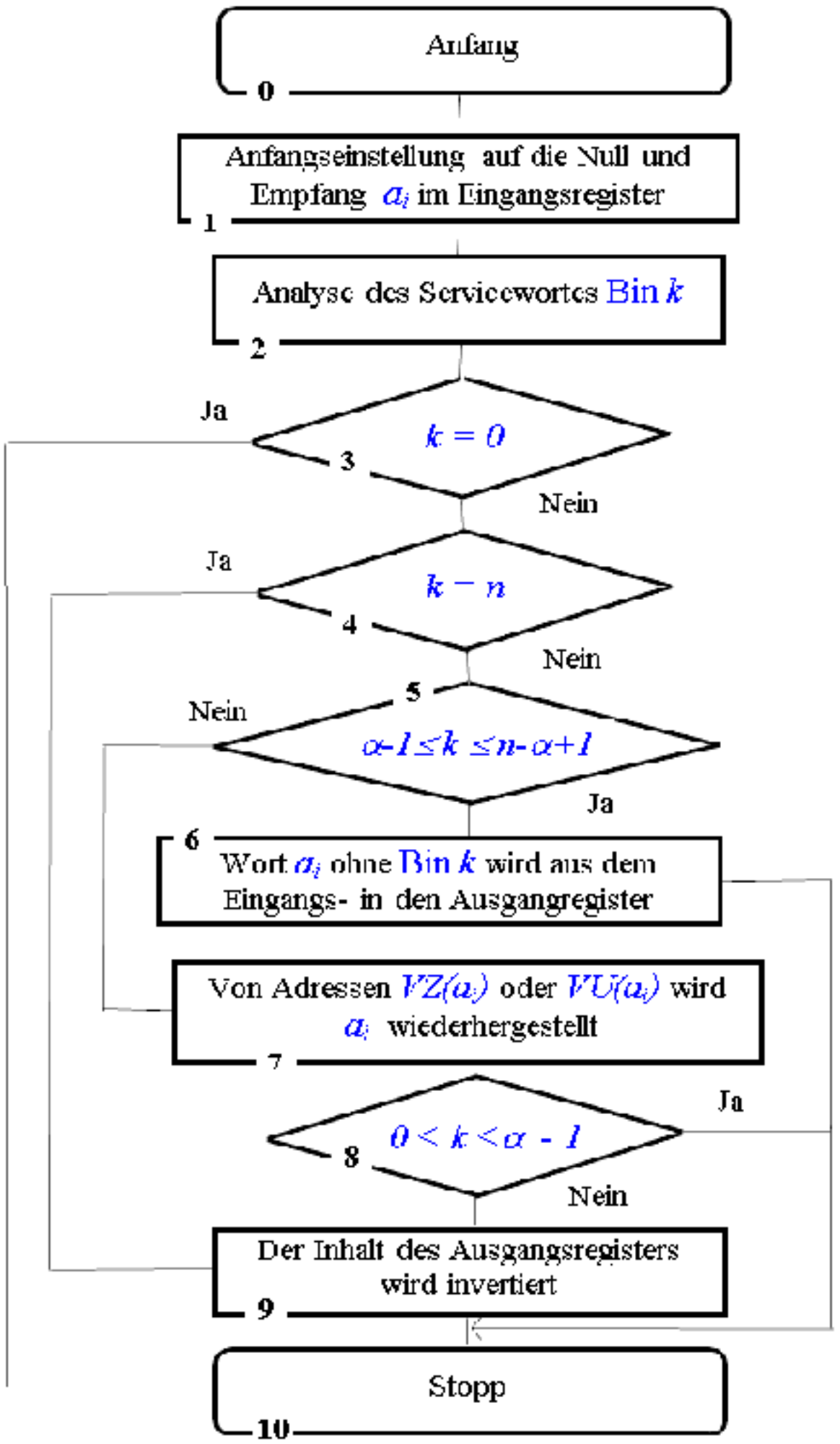



Abb. G: Blockschaubild des allgemeinen Algorithmus AVD

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | ЕлІТ 6.050903.263 ПЗ | |
| Змін | Аркуш | № докум | Підпис | Дата | | |
| Розробив | | Мощна І.Б. | | | Adress-Vektormethode von binärer Datenkompression. *CA* | Літ. / Аркуш / Аркушів |
| Перевірив | | Кулик І.А. | | | | |
| Реценз. | | | | | | СумДУ, гр. ТК-51 |
| Н. Керівник | | Кулик І.А. | | | | |
| Затвердж. | | Опанасюк А.С. | | | | |