Skriptum zur VL

KOMPRESSIONSVERFAHREN

Mag.Dr. Andreas Uhl

RIST++ und Institut für Scientific Computing

Universität Salzburg

Adresse:

Andreas Uhl
Forschungsinstitut für Softwaretechnologie
Hellbrunnerstr.34
A-5020 Salzburg
Österreich
Tel.: ++43/(0)662/8044/5308
Fax: ++43/(0)662/8044/138
E-mail: uhl@cosy.sbg.ac.at

Inhaltsverzeichnis

Kapitel 1

Grundlagen

1.1 Was ist Kompression ?

Kompression bezeichnet die kompakte Darstellung von Daten. Wir beschränken uns hier auf die Kompression von *digitalen* Daten. Solche zu komprimierenden Daten sind z.B.

- Text
- Sourcecode
- Beliebige Dateien
- Bilder
- Video
- Audio
- Sprache

Klarerweise sind diese Daten von sehr unterschiedlicher Natur was deren Menge, Struktur, Verwendungszweck u.s.w. betrifft. Deshalb liegt es sehr nahe für bestimmte Datentypen spezielle Kompressionsverfahren zu entwickeln. Nicht nur die Art der Daten ist jedoch von Bedeutung sondern auch die Umgebung in der mit den Daten operiert werden soll (z.B. Übertragung der Daten in Netzwerken, Archivierung, interaktive Multimedia Anwendungen). Aus diesen Überlegungen ergeben sich verschiedene Möglichkeiten wie Kompressionsmethoden klassifiziert werden können:

- Nach der Art der zu verarbeitenden Daten
- Nach der Art der Einsatzumgebung
- Nach dem Grundtyp der zugrundeliegenden Algorithmen

Terminologie: Oft wird unter *Kompression* die "verlustbehaftete" Speicherung verstanden (lossy compression) wogegen die "verlustfreie" Speicherung oft als *Kodierung* bezeichnet wird. Diese Unterscheidung wird leider nicht einheitlich durchgezogen. Mehr zu dieser Unterscheidung folgt später. Die "Sprache der Kompression" ist Englisch, es gibt in diesem Bereich kaum brauchbare deutschsprachige Bücher oder Zeitschriften (was ja eigentlich für den gesamten Bereich der Informatik gilt !!). Der Begriff *Codec* bezeichnet ein vollständiges Kompressionssystem bestehend aus *Encoder* und *Decoder*. *Kompressionsrate* ist das Verhältnis zwischen der ursprünglichen Datenmenge und der Datenmenge nach dem Kompressionsvorgang und mißt somit die Kompressionsleistung.

1.2 Warum Kompression ?

Kompressionsverfahren werden angewendet um Speichermedien effizient zu nutzen, um Übertragungsbandbreite zu sparen und um Übertragungszeit zu reduzieren. Trotz der gewaltigen Fortschritte im Bereich Speichermedien und Netzwerktechnologien sind effiziente Kompressionsverfahren unumgänglich, man denke nur an den Anspruch "jedem Haushalt seinen MultimediaPC auf dem Information Superhighway mit interaktiven Multimediaanwendungen und Video on Demand". Über den bloßen Anspruch einer Datenmengenreduktion hinaus gehen die neueren Standards MPEG-4 und MPEG-7 die zusätzlich zu exzellenter Kompressionsleistung auch faszinierende Funktionalitäten bieten (siehe später). (für einige Beispiele siehe Tabellen 2A.1 - 2B.3).

Insbesondere drei wichtige Trends tragen dazu bei daß Kompressionverfahren immer wichtiger werden und daß diese Entwicklung den Umgang von digitalen Systemen in Hinblick auf Text, Sprache, Audio, Bildern und Video wesentlich verändern wird was zu neuen Produkten und Dienstleistungen führen wird:

- Die Entwicklung von effizienten Methoden zur Kompression aller möglichen Daten.
- Die Verfügbarkeit von schnellen und billigen single-chip, microprocessor, DSP und VLSI basierten Hardware Kompressoren.
- Die Konvergenz von Computer-, Kommunikation-, Consumer Electronics-, Publishing- und Unterhaltungsindustrie.

Kompression wird ermöglicht weil in digitalen Daten ein hoher Grad von Redundanz vorhanden ist.

- Statistische Redundanz: alle Symbole werden mit der gleichen Anzahl von Bits dargestellt unabhängig von ihrer Autrittshäufigkeit.
- Korrelation: nebeneinanderliegende Daten haben häufig ähnliche Werte, z.B. im Bild- und Videobereich:
  - Spatial Correlation.
  - Spectral Correlation.
  - Temporal Correlation.

Zusätzlich kommt dazu daß bei vielen Datentypen große Anteile irrelevant sind, weil die menschlichen Sinne und das Gehirn nicht das gesamte Ausmaß der Information verarbeiten können. Daher können solche

Teile weggelassen werden. Darüberhinaus gibt es in manchen Datentypen bestimmte Eigenschaften auf höherer Beschreibungebene die unabhängig von Zeit, Ort und Auflösung sind und somit effizient beschrieben werden können (fraktale Eigenschaften).

1.3 Geschichte der Kompressionsverfahren

- 1. JH v. Chr.: Stenographie
- 19. JH: Morse- und Braillealphabeth
- 1950er: statistische Kompressionsverfahren - Bit-muster verschiedener Länge werden verwendet um individuelle Symbole in Abhängigkeit von ihrer Auftrittshäufigkeit darzustellen.
- 1970er: dictionary Algorithmen - Symbolfolgen werden mit Hilfe von einem Wörterbuch ("dictionary") auf kürzere Folgen abgebildet.
- 1970er: Telefonnetzwerke wurden langsam digitalisiert $-->$ die Telekommunikationsindustrie begann Verfahren für Kompression zu entwickeln um mehr Stimmkanäle über ihre Leitungen zu bringen.
- frühe 1980er: FAX über analoge Telefonkanäle.
- mittlere 1980er: Anwendungen für digitale Bilder kommen vermehrt auf den Markt, die "digitale Revolution" bringt digitale Audiokompression.
- 1990er: Video Broadcasting, Video on Demand, ...

1.4 Entscheidungsgrundlagen zur Wahl eines Kompressionsverfahrens

Wenn es klar ist daß in einem digitalen System ein Kompressionsverfahren angewendet werden soll, muß entschieden werden, welche Art von Verfahren angewandt werden soll. Häufig wird die Auswahlfreiheit durch die Existenz von Standards oder de-facto Standards eingeschränkt - durch die zunehmende Entwicklung von offenen Systemen und die in Zukunft vielleicht abnehmende Bedeutung von Standards gewinnen diese Kriterien aber wieder an Bedeutung. Wichtige Entscheidungskriterien sind z.B.:

- Datentyp: 1-D, 2-D, 3-D, ...
- Lossy or Lossless: abängig von Datentypen, Ansprüchen an Qualität oder Kompressionsrate.
- Kompressionsleistung: welche Qualität benötige ich für meine Zielanwendung ?
- Algorithmen Komplexität, Geschwindigkeit, Delay: on- oder offline, real-time Applikation ?
- Hard- oder Software Lösung: Geschwindigkeit, Preis (MPEG-karten sind teuer !)
- Encoder/Decoder Asymmetrie: oftmaliges komprimieren (z.B. Videoconferencing) vs. einmaliges komprimieren (z.B. Bilddatenbanken, CD-ROM, VoD, ...)
- Fehlertoleranz: sind bei der Übertragung/Speicherung Fehler zu erwarten ?

- Skalierbarkeit: dekodieren mit verschiedener Qualität ist möglich.
- Progressive Transmission: je mehr Daten übertragen werden, desto besser wird die Qualität.
- Wird Standard benötigt: ist es eine "geschlossene" Anwendung oder ist Datenaustausch über mehrere Systeme hinweg zu erwarten ?
- Mehrfaches encoding/decoding: wiederholte Anwendung von lossy Kompression, z.B. für Video-editing
- Adaptiv oder nicht adaptiv: sind stark unterschiedliche Eingangsdaten zu erwarten ?
- Zusammenspiel mit anderen Daten: Audio und Video sollten beide frame-basiert sein.
- Zusammenspiel mit anderen Systemen: können Daten in ein Format für andere Anwendung umgewandelt werden ? (z.B. MJPEG $-- >$ MPEG).

Kapitel 2

Grundlegende Methoden und Verfahren

2.1 Verlustfreie Kompression

Bei diesen Methoden können die Daten nach der Kompression exakt rekonstruiert werden, d.h. es geht keine Information verloren. Typische Anwendungen sind Kompression von Text oder Dateien, Fax, aber z.T. auch Bildmaterial wie z.B. in der medizinischen Bildverarbeitung. Bei all diesen Verfahren gibt es einen bekannten Tradeoff: Kodierungs Effizienz - Koder Komplexität - Kodierungs Delay.

Typischerweise handelt es sich um einen dreistufigen Prozeß:

- Das Modell: analysiert die Eingangsdaten bezüglich der Struktur und der Auftrittswahrscheinlichkeiten der einzelnen Symbole.
- Der Koder: produziert einen komprimierten Bitstream unter Benutzung der Information des Modells.
- Der Adaptor: benutzt Informationen aus den Daten um das Modell oder/und den Koder stetig an die Daten anzupassen.

Verlustfreie Kompression beruht auf der Technik Codewörter zu verwenden die kürzer sind als die ihnen entsprechenden Symbole wenn diese häufig sind. Dafür sind die Codewörter länger als die entsprechenden Symbole wenn diese selten vorkommen (there's no free lunch !).

Jeder Information generierende Prozeß kann betrachtet werden als Quelle die eine Folge von Symbolen emittiert die aus einem endlichen Alphabeth gewählt werden (im Fall von computergestützter Verarbeitung ist es letztlich immer das Alphabeth 0,1). Wir bezeichnen diese “Informationsquelle” als S mit dem zugehörigen Alphabeth $\{s_1, s_2, \ldots, s_n\}$ und den entsprechenden Auftrittswahrscheinlichkeiten $\{p(s_1), p(s_2), \ldots, p(s_n)\}$. Der Einfachheit halber betrachten wir diese Auftrittswahrscheinlichkeiten als relative Häufigkeiten. Es ist wünschenswert angeben zu können was die durchschnittliche Information einer gegebenen Quelle ist. Intuitiv sollte erfüllt sein, daß das Auftreten eines weniger häufigen Symbols mehr Information liefert als das Auftreten von sehr häufigen Symbolen. Die Information von unabhängigen Symbolen wie die Summe der Information der Einzelsymbole. Wir definieren

$$I(s_i) = \log \frac{1}{p(s_i)}$$

als die Information die durch das Auftreten des Symbols s_i "geliefert" wird, bezogen auf dessen Auftrittswahrscheinlichkeit $p(s_i)$. Die Basis des Logarithmus bestimmt die Einheit der Menge an Information, z.B. im Fall von Basis 2 handelt es sich um Bits.

Bilden wir nun den Durchschnitt über alle möglichen Symbole der Quelle gelangen wir zur durchschnittlichen Menge an Information pro Symbol, der *Entropie*:

$$H(S) = \sum_{i=1}^{n} p(s_i) I(s_i) = -\sum_{i=1}^{n} p(s_i) \log_2 p(s_i) \quad \text{bits/symbol}$$

Im wesentlichen liefert die Entropie eine Maßzahl wieviele Bits pro Symbol im Schnitt mindestens benötigt werden um eine verlustfreie Kompression durchführen zu können. Die nächsten Abschnitte behandeln Verfahren mit deren Hilfe es gelingt mehr oder weniger nahe an diese theoretische Grenze zu gelangen.

2.1.1 Einfluß der Samplingrate auf mögliche Kompressionsrate

Vergleicht man Daten die von ein und demselben analogen Signal stammen wobei y_i zehnmal so dicht gesampled ist wie x_i, so fällt auf daß benachbarte Werte von y_i wesentlich ähnlicher sind als benachbarte Werte von x_i (weil die abtastung des analogen Signals bei höherer samplingrate "stetiger" ist also zu mehr ähnlichen Werten führt – natürlich immer vorrausgesetzt daß das analoge Signal einen gewissen Grad an Stetigkeit aufweist). Die höhere Ähnlichkeit der y_i wirkt sich auf die Entropie und damit auf die erreichbare Kompressionsrate günstig aus.

2.1.2 Predictive Coding - Differential Coding

Dieses Verfahren ist eigentlich kein Kompressionsverfahren im eigentlichen Sinn sondern eine Art Preprocessing um Daten geeigneter für eine anschließende Kompression zu machen. Die Symbol Statistik wird verändert - aufgrund der Beobachtung, daß benachbarte Daten häufig hochgradig korreliert sind, wird anstelle der Folge der eigentlichen Werte $\{x_1, x_2, \ldots, x_N\}$ die Folge der Differenzen $y_i = x_i - x_{i-1}$ komprimiert. Während die x_i häufig nahezu gleichverteilte p_i aufweisen, haben die y_i deutliche Peaks in der Nähe von 0 (immer unter der Vorraussetzung daß die Annahme der Ähnlichkeit bei Nachbarschaft gilt, was v.a. in Audio-, Bild- und Videodaten erfüllt ist). Betrachten wir z.B. ein Bild mit 999 Bildpunkten und 10 verschiedenen Symbolen. Im Fall a) gilt $p(s_i) = 1/10$, $i = 1, \ldots, 10$, im Fall b) gilt $p(s_1) = 91/100$ und $p(s_i) = 1/100$ für $i = 2, \ldots, 10$. Fall a) entspricht dem "Normalbild", Fall b) dem Differenzenbild der y_i, wobei s_1 das Nullsymbol ist. Im Fall a) erhält man $H(S) = 3.32$, im Fall b) $H(S) = 0.72$. Die wesentlich geringere Entropie zeigt also daß das Differential Coding seine Berechtigung hat.

2.1.3 Runlength Coding – Lauflängen Kodierung

Grundprinzip: Folgen von aufeinanderfolgenden identischen Symbolen werden durch 3 Elemente ersetzt: ein einzelnes Symbol, ein Lauflängenzähler und ein Indikator der angibt wie die beiden anderen Symbole zu interpretieren sind.

Als Beispiel wird der Tape Drive für IBM AS/400 erklärt: Strings von aufeinanderfolgenden Blanks (2-63 byte) werden zu Control-byte komprimiert das "Blank" und Zähler enthält. Strings von aufeinanderfolgenden identischen Character ungleich Blanks (3-63 byte) werden zu 2 byte komprimiert. Control byte ("connsecutive char" und Zähler) und Character byte. Strings von nicht-wiederkehrenden Characters werden expandiert durch Control-byte ("nicht wiederkehrend").

z.B.: b Blank, * control byte; bbbbbbABCDEFbb33GHJKbMN333333 wird zu **ABCDEF**33GHJKbMN*3

<u>ACHTUNG</u>: bei unpassenden Daten kommt es leicht zu Datenexpansion !!

2.1.4 Variable length Coding und Shannon-Fano Coding

Wir bezeichnen $l(s_i)$ als die Länge des Codewortes (in bits) für das Symbol s_i. Die durchschnittliche Codelänge ist definiert als: $L(S) = \sum_{i=1}^{n} p(s_i)l(s_i)$. Um diesen Ausdruck zu minimieren muß man also kurze Codewörter für Symbole mit hoher Auftrittswahrscheinlichkeit verwenden. Interessant ist hier die nahe Verbindung zur Entropie: $H(S) = \sum_{i=1}^{n} p(s_i)I(s_i)$. Klarerweise ist es also notwendig, $l(s_i) = I(s_i) = -\log_2 p(s_i)$ zu setzen um die Entropiegrenze zu erreichen. Dies ist allerdings nur möglich falls $\log_2 p(s_i)$ eine ganze Zahl liefert. Da das kaum der Fall sein wird, rundet man auf die nächste ganze Zahl auf ($->$ Hinweis auf Suboptimalität des Verfahrens !) Genau diese Vorgangsweise wird bei Shannon-Fano Coding gewählt. Als Kode-effizienz wird der Ausdruck $\frac{H(S)}{L}$ bezeichnet. Werte nahe bei 1 sind also wünschenswert.

z.B.: $S = \{s_1, s_2, s_3, s_4\}$ mit $p(s_1) = 0.6$, $p(s_2) = 0.3$, $p(s_3) = 0.05$ und $p(s_4) = 0.05$. $H(S) = 1.4$ bits/Symbol. Wählt man eine fixed length Codierung, erhält man beispielsweise 00, 01, 10, 11 als Codewörter mit einer durchschnittlichen Codelänge von 2 bits/Symbol. Bestimmt man $l(s_i)$ so kommt man auf Werte von 1, 2 und 5 und kommt zu folgendem variable length Code: 0, 10, 110 und 111. s_3 und s_4 wären mit 5 bits zu kodieren, es genügen aber 3 bit um einen eindeutigen Code zu erzeugen. Die durchschnittlichen Codelänge beträgt 1.5 bits/Symbol.

Ein häufig angewandtes Verfahren ist es, das Alphabeth bestehend aus s_i zu ersetzen durch die Erweiterung auf $s_i s_j$ (mit den entsprechenden Auftrittshäufigkeiten). Die bessere durchschnittlichen Codelänge von z.B. 1.43 bits/Symbol wird erkauft durch eine deutlich erhöhte Rechen- und Speicherkomplexität.

2.1.5 Huffman Kodierung

Angenommen es sei gegeben eine Quelle S mit einem Alphabeth der Größe n. Die zwei am wenigsten häufigen Symbole werden vereinigt, dem neuen Symbol die Summe der Auftrittshäufigkeiten zugewiesen und dann das neue Alphabeth der Größe $n-1$ analog betrachtet. Angenommen man kennt die Codewörter der neuen Symbole, so werden die beiden alten generiert durch Anhängen von 0 und 1 an der rechten Seite des neuen Symbols. Dieses Verfahren wird rekursiv angewendet. Das Beispiel auf der Folie hat Entropie $H(S) = 2.15$, der erzeugte Huffman Code hat durchschnittlichen Codelänge von 2.2 bits/Symbol, ein ad-hoc generierter Code (wie im letzten Abschnitt) , z.B. $\{0, 10, 110, 1110, 1111\}$ hat durchschnittlichen Codelänge von 2.25 bits/Symbol.

Modifikation: Wenn viele Symbole kleine Wahrscheinlichkeiten haben steigt der Codewortbedarf stark an. Die Symbole mit geringer Auftrittshäufigkeit werden in eine eigene Klasse eingeteilt, die im Huffmancode mit "ELSE" bezeichnet wird und dann getrennt kodiert wird. Bezeichnung: "modified Huffamn Code".

Probleme:

1. Hat eine Quelle ein Symbol mit $p(s)$ ca. 1 und viele andere mit kleiner Auftrittshäufigkeit $-->$ durchschnittlichen Codelänge von 1 bit/Symbol weil die kleinste Codewortlänge 1 ist obwohl die Entropie wesentlich kleiner ist (vergleiche das Kodieren von Differenzen !!!).
2. Bei wechselnder Quellstatistik kann ein fixer Code leicht zu Datenexpansion führen.
3. Da ein fixes Codebook schlecht ist $-->$ zweistufiger Algorithmus: Statistik aufbauen, Code generieren.
4. Adaptivität ist aufwendig, es wird ja immer der gesamte Baum modifiziert !

2.1.6 Arithmetische Kodierung

Während es bei der Huffman Kodierung eine Entsprechung von einem Quellensymbol zum kodierten Symbol gibt, verwendet die Arithmetische Kodierung ein Codeword für eine ganze Folge von Quellensymbolen s_m der Länge m. Damit kann das Problem der Einschränkung auf ganzzahlige bit/Symbol Werte elegant umgangen werden. Bei der Arithmetischen Kodierung können sehr ähnliche Quellfolgen zu völlig verschiedenen Kodewörtern führen.

Sei s_m eine Folge von binären Quellensymbolen mit $p(0) = p$ und $p(1) = 1 - p$. Weiters sei $I = [0, 1)$ das halboffene Intervall und $\sum p(s_m) = 1$ bei Summierung über alle 2^m möglichen Folgen der Länge m. Man kann mit diesen Angaben Intervalle I_l mit $l = 1, 2, \ldots, 2^m$ konstruieren die alle in I liegen und jeweis einer Folge s_m zugeordnet werden wobei $|I_l| = p(s_m)$ und die I_j leeren Durchschnitt haben. Das folgende Verfahren heißt "Elias Code" und dient in erster Linie der Illustration (weil es unpraktikabel für Implementierungen ist).

Das Intervall I wird geteilt entsprechend der Auftrittshäufigkeiten p und 1-p in die Intervalle [0,p) (falls 0 auftritt) und [p,1) (falls 1 auftritt). Rekursiv werden diese Intervalle (je nach Auftreten des nächsten Symbols) aufgeteilt in $[0, p^2)$ und $[p^2, p)$ bzw. in $[p, 2p - p^2)$ und $[2p - p^2, 1)$. Der Folge 01 würde z.B. das Intervall $[p^2, p)$ zugeordnet. Nach $j - 1$ Quellensymbolen wurde der Folge s_{j-1} das Intervall $[L^{j-1}, R^{j-1})$ zugewiesen.

- folgt 0: $L^j = L^{j-1}$ und $R^j = L^{j-1} + p(R^{j-1} - L^{j-1})$.
- folgt 1: $L^j = L^{j-1} + p(R^{j-1} - L^{j-1})$ und $R^j = R^{j-1}$.

Man kann nun zeigen, daß für jede Folge s_m das entsprechende Teilintervall $[L^j, R^j)$ die gleiche Länge hat wie $p(s_m)$. Sobald nun das einer Folge ensprechende Teilintervall generiert wurde, ist das Codewort für s_m schnell gefunden: es muß nur der Punkt L^j in binärer Form dargestellt und die $-\log_2 p(s_m)$ bits (aufgerundet) nach der Dezimalstelle behalten werden. Das genügt um das Teilintervall und damit die Symbolfolge eindeutig zu identifizieren.

Die tatsächlichen Implementierungen verwenden wegen des Problems der Rechengenauigkeit eine Renormalisierung auf $[0, 1)$ in jedem Schritt mit zusätzlicher Rundung.

Bonuspunkte:

1. Adaptivität ist sehr leicht machbar ! Es muß nur die Statistik mitgeführt werden und die Intervallgrenzen entsprechen angepaßt werden.
2. Schon mit wenig codierten Symbolen kann eine Dekodierung beginnen (erstes Intervall). Dann kommt es mit jedem zusätzlichen Symbol zur schrittweisen Verfeinerung.
3. Einschränkung auf ganzzahlige bits/Symbol Werte überwunden !

2.1.7 Dictionary Kompression

Grundidee aller Dictionary Algorithmen: die Redundanz von wiederholt vorkommenden Phrasen in einem Text wird eliminiert (vergleiche dazu Runlength coding: da ging es nur um wiederholt vorkommende Symbole). Allg.: wiederholt vorkommenden Symbol Strings in bel. Dateien. Ein Frontend wählt Strings aus und die Information für Kompression und Dekompression wird in einem Dictionary gespeichert, von dem der Kompressor und der Dekompressor je eine Kopie haben. Bei diesen Verfahren werden ganze Strings durch Tokens ersetzt. Gut: lookup-table Operationen sind sehr schnell !

LZ77 Dictionary Kompression

Name: von Lempel und Ziv (1977); der hier beschriebene Algorithmus ist die LZSS (Storer/Szymanski) Variante von 1982. Typisch ist ein "sliding window" bestehend aus 2 Teilen: der History buffer (das Dictionary) – enthält einen Block gerade kodierten Textes (einige 1000 Symbole) – und der Lookahead buffer – enthält zu kodierende Strings (10-20 Symbole).

Entscheidend ist die richtige Wahl der Größe der jeweiligen buffer:

- History buffer klein − > Match ist unwahrscheinlich.
- History buffer groß − > hoher Zeitbedarf und schlechte Kompressionsrate durch großen Adreßraum für die Pointer.
- Lookahead buffer klein − > kein Match für lange Strings.
- Lookahead buffer groß − > Zeit wird für Suche nach matches für lange Strings vergeudet.

Beispiel: *Text out* < − ——DATA COMPRESSION—— COMPRESSES DATA—— < − *Text in.* "DATA COMPRESSION" ist der Inhalt des History buffers, " COMPRESSES DATA" ist der lookahead buffer. Das folgende Beispiel benutzt den QIC-122 Standard für Bandlaufwerke mit History buffer 2048 bytes.

Am Beginn wird jedes Symbol in "DATA COMPRESSION" als 0<ASCII Char. Wert> kodiert. " COMPRESS" ist ein String der Länge 9 der 12 Symbole zurück im History buffer vorkommt. Kodierung: 1<offset=12><length=9>. Dann werden "E" und "S" wieder als 0<ASCII Char. Wert> kodiert und "DATA" als String der Länge 4 der 28 Symbole zurück im History buffer vorkommt. Kodierung: 1<offset=28><length=4>.

Man sieht:

- Algorithmus hat start-up Zeit bis der History buffer gefüllt ist.
- Asymmetrischer Algorithmus - die Sucheffizienz im History buffer ist sehr wichtig. Intelligente Datenstrukturen nötig !

LZ78 Dictionary Kompression

Der hier beschriebene Algorithmus ist die Variante LZW(elch) Coding von 1984. Dieser Algorithmus löst das Problem des relativ kleinen History buffers. Phrasen fallen nicht aus dem Dictionary und der Lookahead buffer hat keine bestimmte Größe. Es werden keine Pointer auf frühere Substrings verwendet sondern ein Verfahren namens "Phrasing". Ein Text wird dabei in Phrasen zerlegt, wobei eine Phrase der längste bisher gefundene String + 1 neuer Character ist.

Alle Symbole werden durch Tokens ersetzt die auf Einträge im Dictionary zeigen. Die ersten 256 Dictionary Einträge sind die üblichen 8 bit ASCII code Symbole. Das Dictionary wird dann schrittweise durch neue Phrasen erweitert die aus dem zu kodierenden Text entstehen.

Wesentliche/kritische Punkte:

- Wie speichert man eine variable Anzahl von Strings variabler Länge effizient ab ?
- Wie groß soll das Dictionary werden ? Was ist wenn es voll ist (z.B. neues beginnen, oder nur die Strings behalten die oft verwendet wurden) ?

2.1.8 Welcher Algorithmus für welche Anwendung ?

Diese Frage ist allgemein nicht zu beantworten, letztlich kann nur ein Testen mit den in Frage kommenden Daten diese Frage lösen !

2.2 Verlustbehaftete Kompression

Bei Verfahren zur verlustbehafteten Kompression gibt es zusätzlich zum Tradeoff zwischen Kodierungs Effizienz - Koder Komplexität - Kodierungs Delay noch den zentralen Faktor Kompressions Qualität. Dieser Faktor ist extrem schwierig zu beurteilen, weil hier der einzig wichtige Maßstab die menschliche Wahrnehmung ist. Trotz zahlreicher Versuche v.a. im Bereich der Bild- und Videoverarbeitung ist es nicht gelungen eine wirklich überzeugende Maßzahl für die Qualität eines Signales zu entwicklen, die gut mit der menschlichen Wahrnehmung von "Qualität" korreliert. Insbesondere ist zu bedenken, daß der Begriff "Qualität" intersubjektiv verschieden ist ! Die einzige Methode um wirklich verläßliche Aussagen bezüglich menschlicher Wahrnehmung treffen zu können ist eine Menge von Versuchspersonen die Signalqualität beurteilen zu lassen. Offensichtlicher Nachteil: **AUFWENDIG !**. Die klassische Maßzahl die verwendet wird ist der Signal/Rauschabstand ((P)SNR) der auf dem root-mean-squared error (RSME) beruht:

$$RMSE = \sqrt{1/N \sum_{i=1}^{N} (f(i) - f'(i))^2}$$

wobei $f'(i)$ das rekonstruierte Signal ist. Peak-signal-to-noise ratio (gemessen in DeciBel dB) ist definiert wie folgt (wobei MAX der maximal mögliche Signalwert im Originalsignal $f(i)$ ist):

$$PSNR = 20 \log_{10}(\frac{MAX}{RMSE}).$$

Je höher der Wert der PSNR ist, desto höher ist die Qualität des rekonstruierten Signals. Mit zunehmender Kompressionsrate ist also abnehmende PSNR zu erwarten (rate-distortion tradeoff). Nochmals sei darauf hingewiesen, daß ein so einfaches Fehlermaß keinesfalls auf HVS Eigenschaften oder verschiedene Arten von Störungen eingehen kann. Große Qualitätsunterschiede werden allerdings meist richtig erkannt.

2.2.1 Quantisierung

Quantisierung ist der Übergang von der Beschreibung eines Signals durch ein großes Alphabeth durch die Beschreibung durch ein kleineres Alphabeth. Notwendigerweise kommt es bei der Umkehrung zu Fehlern, häufig sind genau das die einzigen Fehler die in der verlustbehafteten Kompression gemacht werden.

z.B.: ein Grauwertbild sei mit 8 bit/Pixel gespeichert. d.h. fuer jedes Pixel stehen 256 Grauwerte zur Auswahl. Benutzt man nun eine Quantisierung auf 4 bit/Pixel (bpp) stehen nur noch 16 Grauwerte zur Verfügung. Umgekehrt kann man das 4 bpp Bild nicht mehr exakt in das Ausgangsbild zurückverwandeln.

Betrachtet man einzelne Werte eines Signals und deren entsprechende quantisierte Werte so spricht man von *skalarer* Quantisierung. *Vektor*quantisierung bezeichet eine Gruppe von Verfahren bei denen n-tupel von Signalwerten gemeinsam auf einen quantisierten n-tupel abgebildet werden. In jedem Fall kann Quantisierung auch als Approximation betrachtet werden.

Man spricht von *uniform* quantization wenn der Wertebereich auf den ein Signal hin abgebildet wird gleichmäßig in Teilintervalle aufgeteilt wird. Das ist insbesondere dann sinnvoll, wenn die Auftrittshäufigkeiten im Originalsignal gleichmäßig verteilt sind. Der Wert “0” spielt in der Kompression immer eine besondere Rolle. Oft wird das Teilintervall das auf 0 abgebildet wird größer als alle anderen gewählt - man spricht dann von “uniform quantization with deadzone around 0”.

Sind die Auftrittshäufigkeiten im Originalsignal nicht gleichmäßig verteilt und man kennt diese Verteilung so ist es sinnvoll in den Regionen eine feinere Aufteilung zu verwenden in denen viele Signalwerte liegen (mehr bits dort verwenden wo es Sinn macht Detail zu zeigen !). In diesem Fall spricht man von “non-uniform quantization”. Kennt man diese Verteilung nicht, kann man zuerst das Signal analysieren und aufgrund der ermittelten Auftrittshäufigkeiten dann adaptiv die Quantisierung festlegen - “adaptive quantization”. Insbesondere im Transformationsbereich kennt man von typischen Signalen die Auftrittshäufigkeiten die sich oft sehr ähneln - in diesem Fall kann die Quantisierung gut im Vorraus modelliert werden.

2.2.2 DPCM

Differential pulse code modulation (DPCM) ist das allgemeine Konzept hinter dem differential/predictive Kodieren. An Stelle der Differenz zweier aufeinanderfolgender Signalwerte kann auch eine Linearkombination mehrerer aufeinanderfolgender Signalwerte verwendet werden.

z.B.: $y_i = x_i - x'_i$ mit $x'_i = \alpha x_{i-1} + \beta x_{i-2} + \gamma x_{i-3}$. Die Werte α, β, γ heißen Prediktor Koeffizienten, x'_i ist die Prediction für x_i. Es handelt sich hier um einen *linearen* Prediktor der *Ordnung 3*. Abgespeichert wird die Folge der y_i.

Die Prediktorkoeffizienten können für bekannte Signalklassen fixiert werden oder aber an jedes neue Signal angepaßt werden. Die optimalen Koeffizienten α, β, γ die beispielsweise die Gesamtentropie von y_i minimieren können natürlich berechnet werden. Weiters können für jedes beliebige Signalteil eigene Koeffizienten verwendet bzw. optimiert werden ("lokal adaptive prediction"). Es ist natürlich zu beachten daß hier die jeweiligen Koeffizienten (im Gegensatz zu fix gewählten Koeffizienten) auch gespeichert werden müssen.

Die Entscheidung ob es sich bei DPCM um ein verlustfreies oder verlustbehaftetes Kompressionsverfahren handelt fällt erst mit der Entscheidung in welcher Art die y_i abgespeichert werden. Werden diese mit einem verlustfreien Verfahren abgespeichert (z.B. Huffamn kodiert) so erhält man auch insgesamt ein verlustfreies Verfahren. Wird nun *Quantisierung* auf das Differenzsignal y_i angewendet so erhält man ein verlustbehaftetes Verfahren.

Eine häufig verwendete Variante von DPCM verwendet eine fix vorgegebene Anzahl von Prediktor und Quantisierer Kombinationen die dann in Abhängigkeit von der lokalen Signalstatistik verwendet werden.

2.2.3 Transform Coding

Das klassische transformationsbasierte Kompressionsverfahren besteht aus drei Teilen:

1. Transformation
2. Quantisierung: alle vorher beprochenen Varianten kommen zum Einsatz um die Transformationskoeffizienten auf ein möglichst kleines Alphabeth abzubilden. Dieser Schritt ist nicht reversibel, hier passiert der Fehler.
3. Verlustfreie Kodierung der Transformationskoeffizienten: alle vorher beprochenen Varianten kommen zum Einsatz um die quantisierten Transformationskoeffizienten möglichst nahe an der Entropiegrenze zu kodieren. Meist eine Kombination aus Lauflängen (viele 0) und Huffman Kodierung oder Arithmetische Kodierung.

Die Transformation

Die Transformation (häufig werden sog. "Integraltransformationen" verwendet) dient dazu die Daten des Signals in eine besser zu komprimierende Form zu bringen. Meistens wird das Konzept verwendet ein Signal mit Hilfe von orthogonalen Basisfunktionen darzustellen.

z.B.: Vektoren im 2 dimensionalen Raum können durch einen Satz von orthogonalen (Def.: inneres Produkt ist 0) Basis-Vektoren dargestellt werden (Orthogonalbasis):

$$(x, y) = \alpha(1, 0) + \beta(0, 1).$$

$\{(1, 0), (0, 1)\}$ sind die orthogonalen Basisvektoren. α und β sind die Koeffizienten die angeben, wie stark jeder Basisvektor verwendet werden muß um den Vektor (x, y) darstellen zu können. Nur die Orthogonalitätseigenschaft ermöglicht eine minimale Anzahl von Basisvektoren.

Dieses Konzept kann auf Funktionen bzw. Signale übertragen werden.

$$f(x) = \sum_n < f(x), \psi_n(x) > \psi_n(x)$$

Die $\psi_n(x)$ sind die orthogonalen Basisfunktionen, $< f(x), \psi_n(x) >$ sind die Transformationskoeffizienten die angeben, wie stark jede Basisfunktion verwendet werden muß um das Signal "gut" darstellen zu können. Für eine Kompressionsanwendung werden die $< f(x), \psi_n(x) >$ berechnet und dann gespeichert. Da die $\psi_n(x)$ orthogonal sind, ist die entsprechende Anzahl minimal.

z.B.: im Falle der Fouriertransformation sind die $\psi_n(x) = e^{-\pi i n x} = \cos(nx) - i\sin(nx)$. In diesem Fall werden Frequenzen von periodischen Signalen betrachtet. Ein Fourierkoeffizient $< f(x), \psi_n(x) >$ gibt die Stärke des Frequenzanteils n in einem Signal an. Es ist klar, daß nicht alle Signale auf diese Art am effizientesten dargestellt werden können.

Für ein Transformationsbasiertes Verfahren wird nun also das Signal $f(x)$ in die Form $f(x) = \sum_n < f(x), \psi_n(x) > \psi_n(x)$ gebracht. Die Transformationskoeffizienten $< f(x), \psi_n(x) >$ werden geeignet quantisiert und dann verlustfrei gespeichert. Sind die $\psi_n(x)$ gut gewählt, sind die Transformationskoeffizienten nach der Quantisierung häufig 0 und müssen nicht gespeichert werden. Bei der Rekonstruktion ergibt sich daher nur ein geringer Fehler.

Daher ist es ganz offensichtlich, dass es wichtig ist für gegebene Signale die geeigneten Basisfunktionen zu verwenden !

Eine weitere Interpretationsmöglichkeit von Integraltransformationen ist es sie als Rotation der Koordinatenachsen zu intepretieren. Die Daten werden in ein Koordinatensystem transformiert, das eine bessere Kompression erlaubt. Als Beispiel betrachten wir die Paare $X = (x_1, x_2)$ die Paare benachbarter Pixelwerte darstellen. Trägt man diese Paare in einem entsprechenden Koordinatensystem auf so liegen die meisten Punkte auf der Hauptdiagonale. Die Varianz der x_j ist definiert als

$$\sigma_{x_j}^2 = 1/M \sum_{i=1}^{M} (x_{ji} - \mathbf{x_j})^2$$

wobei M die Anzahl der Paare und $\mathbf{x_j}$ der Mittelwert der x_j über alle Paare ist. Eine einfache Methode um Kompression zu erreichen ist es in jedem Paar eine der Komponenten durch den entsprechenden Mittelwert $\mathbf{x_j}$ zu ersetzen. Der MSE dieser Kompressionsmethode ist genau $\sigma_{x_j}^2$. Klarerweise ist aber die Varianz in beide möglichen Richtungen groß und daher das Ergebnis dieser Kompressionsart schlecht.

Betrachten wir nun die Transformation von X nach $Y = (y_1, y_2)$ durch eine Rotation der Koordinatenachsen um 45 Grad (Y = AX wobei A eine Rotationsmatrix ist). Wesentliche Eigenschaft einer geeigneten Transformation ist daß

$$\sigma_{x_1}^2 + \sigma_{x_2}^2 = \sigma_{y_1}^2 + \sigma_{y_2}^2$$

Während die Varianzen im Originalkoordinatensystem ungefähr gleich waren, ist die Varianz im neuen Koordinatensystem sehr ungleich verteilt, d.h. $\sigma_{y_1}^2 >> \sigma_{y_2}^2$. Daher macht es in dieser Darstellung viel mehr Sinn die Komponente y_2 durch ihren Mittelwert $\mathbf{y_2}$ da der Fehler durch diese Kompression genau $\sigma_{y_2}^2$ entspricht.

Beispiele für beliebte Transformationen (sollen können: Daten dekorrelieren, datenunabhängige Basisfunktionen, schnelle Algorithmen):

1. Karhunen-Loeve Transformation: optimale Transformation da in Abhängigkeit von den Daten die Basisfunktionen bestimmt werden. Nachteil: Komplexität $O(N^3)$ und keine allgemein nutzbaren Funktionen. Einsatz höchstens bei "multiple-spectral-band images" oder sehr ähnlichen Daten.

2. Fourier Transformation: der Klassiker der allerdings für Kompressionszwecke wegen der komplexen Werte und ungünstiger Periodizität weniger geeignet ist. $F(u) = 1/n \sum_{j=0}^{n-1} f(j) e^{\frac{2\pi i u j}{n}}$.
3. Discrete Cosinus Transformation: beliebt für Bildkompression, verwendet auf Blöcken. $F(u) = 1/n \sum_{j=0}^{n-1} f(j) \cos \frac{(2j+1)u\pi}{2n}$.
4. Wavelet Transformation: Verwendet Basisfunktionen die von 2 Parametern abhängen: eine Art Frequenz und Zeit. $W_{a,b}$. Näheres im Kapitel über lossy Kompression.

Die Quantisierung

Eine wesentliche Frage für die Quantisierung ist es wie die Transformationskoeffizienten zur Weiterverarbeitung ausgewählt werden.

1. Vordefinierte Zonen im Transformationsbereich: nur niederfrequente Koeffizienten werden quantisert, der Rest ignoriert (d.h. auf Null quantisiert).
2. Threshold Auswahl: Koeffizienten unterhalb einer bestimmten Schranke werden ignoriert (Nachteil: die Lokation muß gespeichert werden), oder nur die L größten Koeffizienten werden bearbeitet, u.s.w.

Wichtige Vertreter der Transformationsbasierten Kompressionsverfahren sind DCT-basierter JPEG, Wavelet Verfahren, Suband Coding.

2.2.4 Codebook-basierte Kompression

Es wird ein Dictionary verwendet um Teile des Signals durch Teile des Dictionarys darzustellen. Also konkret: Wähle einen zu komprimierenden Signalteil der Größe n und suche das im Dictionary befindliche Datenstück mit gleicher Größe das dem Signalteil am ähnlichsten ist (hier haben wir wieder die schon diskutierte Frage nach dem Ähnlichkeitsmaß/Qualitätsmaß !). Anstelle des Signalteils wird nur der Index des Datenstücks im Dictionary gespeichert. Natürlich müssen hier Coder und Decoder im Besitz des gleichen Dictionarys sein.

Wichtige Vertreter sind Vector Quantisierung und Fraktale Kompression.

Kapitel 3

Bildkompression

Wir betrachten Bilder als zweidimensionale Arrays die als Arrayelemente die jeweiligen Helligkeitsstufen beinhalten. Im Fall von Graustufenbildern hat ein Bild mit der "Farbtiefe" von 8bit $2^8 = 256$ verschiedene Graustufen, ein bilevel Bild (=Schwarz-Weiß Bild) benötigt daher 1bpp (bpp = bit per pixel). Farbbilder können in verschiedenen Darstellungen gegeben sein. Klassisch ist die RGB Darstellung, bei der jedem der Farbkanäle rot, grün und blau ein Array mit entsprechenden Helligkeitswerte ebendieser Farbe zugeordnet ist. Ein Farbbild mit 24 bit Farbtiefe entsteht so aus drei 8 bit tiefen Farbkanälen. Das menschliche Auge ist aber aus physiologischen Gründen gegenüber den drei genannten Farben nicht gleich sensitiv - am besten können wir Grüntöne wahrnehmen, am schlechtesten Blautöne. Aus diesem Grund (und einigen anderen) wurde eine alternative Darstellung entwickelt: YUV. Y ist ein Helligkeitskanal (engl. luminance channel), U und V sind die Farbkanäle (engl. chrominance channels). Y wird gewonnen aus einer Gewichtung von RGB (ca. Y = 0.3 R + 0.5 G + 0.2 B), ebenso können U und V berechnet werden - beide Farbdarstellungen können ohne wesentliche Einschränkung ineinander umgewandelt werden. Vorteil von YUV: Farb/SW TV und Subsampling der Chroma Kanäle. Alle weiteren Farbdarstellungen die im Bild- und Videobereich verwendet werden sind Varianten dieser Darstellungen.

3.1 Verlustfreie Bildkompression

Wie schon bei der abstrakten Betrachtung der verlustfreien Kompression klar wurde, ist die erreichbare Kompressionsrate sehr beschränkt. Es werden daher in der Bildverarbeitung nur dann verlustfreie Verfahren verwendet, wenn das notwendig/vorgeschrieben ist, z.B. medizinische Bildgebung, Speicherung von Plänen, wenn es um das Speichern von Texten in Form von Bildern geht (die ja lesbar sein sollen), etc.

3.1.1 Faxkompression

Es gibt hier die weitverbreiteten Standards ITU-T T.4 und T.6 für Gruppe 3 und 4 Faxgeräte. Die Kompressionsraten die hier erreicht werden können leicht 15 und mehr erreichen (allerdings wirklich nur für Dokumente, die ganz anders strukturiert sind als z.B. Bilder nach Halftoning Bearbeitung). Alle diese Standards verwenden Runlength und Huffman Coding. Eindimensionales T.4 kodieren verwendet nur die Pixel in der monentanen Zeile - zuerst werden die Runs bestimmt, der Output wird mit einem fixen (modifizierten)

Huffamn Code komprimiert. Jede Zeile wird durch ein EOL Zeichen abgeschlossen um Resynchronisierung nach Übertragungsfehlern zu garantieren. Um die Kompressionsrate zu steigern, wurde auch eine zweidimensionale Option entwickelt: nur manche Zeilen werden kodiert wie oben beschrieben, die anderen werden relativ zu den jeweils eindimensional kodierten dargestellt. Ein zusätzliches Bit nach dem EOL zeigt an ob die nächste Zeile 1-D oder 2-D kodiert wurde. Resynchronisierung nach Übertragungsfehlern ist hier nur nach 1-D kodierten Zeilen möglich. Der T.6 Standard ist für garantiert fehlerfreie Übertragung, hier gibt es keine EOL Zeichen und nur zweidimensionales Kodieren, die Kompressionsrate wird dadurch maximiert.

3.1.2 JBIG

Joint Binary Image Experts Group wurden von ITU-T and ISO mitgetragen. Es ist ein Standard zum Kodieren von Binärbildern, jedoch nicht beschränkt auf Fax (d.h. schriftliche Dokumente) sondern auch im Hinblick auf Haltoned Images entwickelt. Heißt auch ITU-T T.82 Recommendation. Ein bearbeitetes Pixel wird über 3 Zeilen "predicted", das eigentliche Kodieren übernimmt ein adaptiver Arithmetischer Koder (der IBM Q-Coder) um auf verschiedene Datentypen reagieren zu können. Daraus ergibt sich eine bessere Kompressionsrate – 5.2 und 48 versus 1.5 und 33 für Halftone Image und Dokument – aber auch 2-3 fache (Software)Verarbeitungszeit im Vergleich zu ITU-T T.4 und T.6. Darüberhinaus erlaubt JBIG hierarchische Kodierung: zuerst wird ein Bild mit geringer Auflösung kodiert, dann wird Schritt für Schritt die horizontale und Vertikale Auslösung verdoppelt.

JBIG kann aber auch (wie alle anderen Verfahren zur Kompression von Binärbildern) benutzt werden um Bilder mit größerer Bittiefe verlustfrei zu komprimieren. Dies wird erreicht durch sukzessives Kodieren von einzelnen Bitplanes (das sind ja genau Binärbilder) der entsprechenden Bilder, z.B. man beginnt mit der Bitplane die dem most significant Bit (MSB) entspricht und führt das Verfahren solange durch bis man bei der Bitplane für das least significant Bit (LSB) angekommen ist. Da die typischen Kompressionsverfahren große einheitliche Flächen gut komprimieren können, sollten diese Bitplanes möglichst so beschaffen sein. Leider führt die gewöhnliche Binärdarstellung aber genau zum entgegengesetzten Effekt. Betrachtet man z.B. ein Bild das zwischen den Werten 127 und 128 wechselt - in Binärdarstellung ist das 01111111 und 10000000. Die Zahlen unterscheiden sich in jeder Bitposition, daher sind auch die entsprechenden Bitplanes schlecht zu komprimieren. Sie sind zwar untereinander korreliert, dies wird jedoch meist nicht ausgenutzt. Eine effektive Abhilfe schafft der *Gray Code.* Das ist eine Methode eine Menge von Zahlen in ein binäres Alphabeth zu mappen sodaß sich benachbarte Zahlenwerte um genau ein Bit in der Binärdarstellung unterscheiden.

Binary Code $\longrightarrow$ Gray Code:

1. Beginnend mit dem MSB der Binärdarstellung, alle 0 werden belassen bis eine 1 kommt.
2. Die 1 wird belassen, alle folgenden Bits werden invertiert bis eine 0 kommt.
3. Die 0 wird invertiert, alle folgenden Bits werden belassen, bis eine 1 kommt.
4. Goto Step 2.

Gray Code $\longrightarrow$ Binary Code:

1. Beginnend mit dem MSB der Binärdarstellung, alle 0 werden belassen bis eine 1 kommt.

2. Die 1 wird belassen, alle folgenden Bits werden invertiert bis eine 1 kommt.
3. Die 1 wird invertiert, alle folgenden Bits werden belassen, bis eine 1 kommt.
4. Goto Step 2.

3.1.3 Lossless JPEG

Bis zu einer Bittiefe von 6 bpp gibt zukzessives JBIG kodieren gute Ergebnisse, bei einer höheren Bittiefe sollte man zu einem anderen Verfahren greifen. Hier bietet sich insbesondere der Standard zur verlustfreien Bildkomprimierung der Joint Photographic Experts Group (JPEG) an. Lossless JPEG ist ein klassischer lossless DPCM coder. Aufgrund einer Prediction für ein aktuelles Pixel wird ein Residual erstellt (das geringere Entropie aufweist) welches dann mit einem verlustfreien Verfahren kodiert wird, in diesem Fall mit einem fixen Huffman Code. Es stehen weiters 7 verschiedene Prediktoren zur Auswahl, die allerdings manuell ausgewählt werden müssen. In (nicht standardisierten) Erwieterungen wird automatisch der beste Prediktor ausgewählt - sei es für das ganze Bild, auf Blockbasis oder stetig. Das Residual ist nicht direkt Huffman kodiert: es wird als ein Symbolpaar dargestellt - Kategorie und Größe. Nur die Kategorie wird Huffman kodiert. Weiterentwicklungen gibt es im Rahmen des JPEG-LS Algorithmus.

3.1.4 GIF, PNG, TIFF

Der GIF (Graphics Interchange Format) Standard ist ein proprietärer CompuServe Standard (was zu großer Aufregung geführt hat). Aus historischen Gründen (d.h. 1987 glaubte man daß die Darstellung von 16 Farben auf einem Computerbildschirm Zukunftsmusik sei) ist die Anzahl der möglichen Farben auf 256 beschränkt (aus einer möglichen Auswahl von 16 Millionen). Das heißt das für höhere Farbtiefe ein Quantisierungseffekt auftritt (d.h. in diesem Fall verlustbehaftete Kompression). Neben einer Vielzahl von Headerinformation (u.a. für die Möglichkeit von animated GIFs und der (lokalen) Farbbeschreibungen) werden die tatsächlichen Bilddaten mit dem LZW Verfahren komprimiert (Achtung: patentrechtlich geschützt !).

PNG ist das *Portable Network Graphics* Format das als lizenzfreie Antwort auf GIF geschaffen wurde ("**P**NG **N**ot **G**IF"). PNG bunutzt eine Variante von LZ77, ist vollständig lossless und wird vom wordwide Web Konsortium (W3C) unterstützt.

TIFF (Tagged Image File Format) unterstützt verschiedene Farbdarstellungen und Bildauflösungen. Es ist jedoch **KEIN** Kompressionsstandard sondern ein Fileformat das unkomprimierte Bilder und 6 verschidene Kompressionsmethoden (5 davon lossless und lossy JPEG) unterstützt. Die verlustfreien Methoden beinhalten die Faxkompressionsmethoden ITU-T T.4 und T.6 und eine LZW Variante.

3.2 Verlustbehaftete Bildkompression

3.2.1 JPEG

JPEG ist das klassische Bildkompressionsverfahren des vergangenen Jahrzehnts. Es besteht aus einer DCT die auf 8×8 Bildblöcke angewendet wird, anschließender Quantisierung, zig-zag Umordnung und abschließender Kodierung (Mischung aus Runlength und Huffman Kodierung).

3.2.2 Wavelet Kompression, JPEG2000

Im Gegensatz zur Fouriertransformation ist die Wavelettransformation von zwei Parametern abhängig:

$$W_{a,b}(f) = |a|^{-1/2} \int_{-\infty}^{\infty} f(t)\, \psi\left(\frac{t-b}{a}\right)\, dt$$

Die Funktionen $\psi_{a,b}(s) = |a|^{-1/2}\, \psi\left(\frac{s-b}{a}\right)$ werden Wavelets genannt, $\psi(s)$ heißt oft *mother wavelet*.

Wenn sich a ändert, beschreiben $\psi_{a,b}$ unterschiedliche Frequenzbereiche, wird auch der Parameter b geändert verschiebt sich das Zeit-Lokalisierungszentrum (das in $s = b$ liegt). Alle Wavelets sind also verschobene und skalierte Versionen des Mother Wavelets.

Der Hauptunterschied zur gefensterten Fouriertransformation liegt in der Form der Basisfunktionen. Während alle Fensterfunktionen im Fourierfall gleiche Breite haben, haben die $\psi_{a,b}$ eine Breite die von der Frequenz abhängen: hochfrequente $\psi_{a,b}$ haben geringe Breite während niedefrequente $\psi_{a,b}$ breiter sind.

Beispiele für $\psi(x)$, das Mother Wavelet:

$$\psi(x) = \left(1 - x^2\right)\, e^{\frac{-x^2}{2}} \qquad \text{"Mexican hat"}$$

$$\psi(x) = \frac{\sin(2\pi x) - \sin(\pi x)}{\pi x} \qquad \text{"Shannon wavelet"}$$

$$\psi(x) = \begin{cases} 1 & : \; 0 \leq x \leq 1/2 \\ -1 & : \; 1/2 < x < 1 \\ 0 & : \; otherwise \end{cases} \qquad \text{"Haar wavelet"}$$

Diskretisierung: Beschränkung der Parameter (a, b) auf diskrete Werte: $a = a_0^m$, $b = nb_0a_0^m$ mit $m, n \in \mathbf{Z}$ und $a_0 > 1$, $b_0 > 1$. Gern nimmt man $a_0 = 2$ and $b_0 = 1$.

$$W_{m,n}(f) = 2^{-m/2} \int_{-\infty}^{\infty} f(t)\psi(2^{-m}t - n)\, dt$$

Die schnelle Wavelettransformation beruht auf einer einfachen Subbandfilterung die höherdimensional separabel angewendet werden kann.

In Analogie zu JPEG bestehen einfache Kompressionsalgorithmen aus Transformation, Quantisierung und Kodierung. Fortgeschrittene Verfahren beruhen auf dem Konzept der Zerotrees und benutzen die Lokalität der Transformationsbereiches der Wavelettransformation.

3.2.3 Fraktale Kompression

Fraktale Kompression versucht Selbstähnlichkeiten in einem Bild zur effizienten Beschreibung auszunutzen. Die Grundidee beruht auf folgender Beobachtung: komplexe geometrische Stukturen wie etwa Farne, Apfelmännchen etc. lassen sich durch eine kleine Menge von Formeln beschreiben, obwohl der Detailreichtum unendlich groß sein kann ("Fraktale"). Um so eine Struktur platzsparend zu speichern ist es besser diese Formeln zu speichern als das entsprechende Bild. Dies soll auch mit gewöhnlichen Bildern gemacht werden. Hier gibt es allerdings zwei Probleme:

1. Gewöhnlichen Bilder sind keine Fraktale und weisen nur in einem sehr eingeschränkten Sinn Selbstähnlichkeiten auf. Daher wird das Bild unterteilt und die globale Selbstähnlichkeit wird auf Ähnlichkeiten von Blöcken reduziert.
2. Es ist sehr rechenaufwendig, zu einem gegebenen Fraktal die zugehörigen Formeln zu finden ("inverses Problem").

Fraktale Kompression bleibt in jedem Fall sehr zeitaufwendig, wogegen die Dekompression sehr schnell ist. Diese extreme Asymmetrie macht dieses Verfahren für spezielle Anwendungen sehr geeignet, für andere völlig ungeeignet (gut für off-line Anwendungen, Photo-CDROM, VoD, schlecht für VC u.s.w.).

3.2.4 Vektor Quantisierung

Das zu komprimierende Bild wird zuerst in n-dimensionale Bildvektoren zerlegt (z.B. $n = l \times m$ Pixelblock formt n-dimensionalen Vektor oder 3-dimensionaler Vektor durch RGB Komponenten eines Pixels). Jeder Bildvektor X wird mit den Codevektoren X_i, $i = 1, \ldots, N_c$ verglichen die aus einem zuvor generierten oder definierten Codebook stammen. Diese Vektoren haben auch Dimension n. Der best-passendste Vektor wird durch Distortion - Minimierung ermittelt. Nach dieser Wahl wird nur der Index des Codevektors gespeichert (mit $\log_2 N_c$ bits).

Der Decoder ist extrem schnell, mit einem look-up table Verfahren wird der Index wieder durch den entsprechenden Vektor ersetzt. Kompressionsrate = $\log_2 N_c/n$ bpp (d.h. gut ist ein kleines Codebook und eine große Vektordimension).

Probleme:

1. Wie erzeugt man das Codebook ?
   - LBG Algorithmus
   - Codebook Initialisierung
2. Wie soll Codebook strukturiert sein um effizient suchen zu können ?
   - Baumstruktur
   - Product Codes

Weitere Spielarten von Vektorquantisierung sind die folgenden:

- Mean/Residual VQ
- Interpolative/Residual VQ
- Klassifizierte VQ
- Hierarchische VQ
- Transform VQ

3.2.5 Block Truncation Coding

Nicht überlappende $n \times n$ Pixelblöcke werden mit einem one-bit Quantizer kodiert, der in Abhängigkeit von der Blockstatistik designed wird. Es wird der Mittelwert des Blockes berechnet, alle Pixel mit größerem Wert werden auf 1 gesetzt, der Rest auf 0. Die entstehende Bitmap wird kodiert. Ebenso wird der Mittelwert der beiden entstehenden Regionen berechnet und kodiert. Bei der Rekonstruktion werden die Werte 0 und 1 durch die jeweiligen Mittelwerte der Regionen ersetzt. Um eine höhere Qualität zu erzielen gibt es eine hierarchiches BTC Verfahren: liegen die Mittelwerte der beiden Regionen zu weit auseinander und ist der Fehler zu groß wird der Block aufgeteilt und das Verfahren auf die kleineren Teilblöcke angewendet.

3.2.6 Hierarchical Coding

Bilder sind so komprimiert, daß man ein Bild mit verschieden hoher Auflösung oder Qualität dekomprimieren kann. Anwendung:

- Progressive Transmission
- Multiuser Environments

Gemeinsamer Aspekt: benutzt werden Bildhierarchien mit verschiedenen Stufen, die dem rekonstruierten Bild mit verschiedener Auflösung bzw. Qualität entsprechen.

- Fixed resolution Hierarchies: Rekonstruiertes Bild ist gleich groß wie Original und der Wert eines konkreten Pixels wird Stufe für Stufe verfeinert. z.B.: Bit-Planes, Transform Based - wichtige Koeffizienten zuerst
- Variabel resolution Hierarchies: Pyramidenstruktur, Basis ist Originalbild, auf höheren Stufen nimmt die Auflösung ab. z.B.: Subsampling Pyramid, Mean Pyramid, Prediction/residual Pyramid (Gaussian-Laplacian Pyramid)