T.C.
TRAKYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

VERİ SIKIŞTIRMADA YENİ YÖNTEMLER

Altan MESUT

Doktora Tezi

Bilgisayar Mühendisliği Anabilim Dalı

2006

EDİRNE

Danışman: Yrd. Doç. Dr. Aydın CARUS

ÖZET

Bu tezin amacı, günümüzde yaygın olarak kullanılmakta olan kayıplı ve kayıpsız veri sıkıştırma yöntemlerinin incelenmesi, eskiden geliştirilmiş olan yöntemler ile yakın zamanda geliştirilmiş olan yöntemler arasındaki farklılıkların belirlenmesi ve yeni yöntemlerin kendisinden önceki yöntemleri ne yönde geliştirdiğinin araştırılmasıdır. Tezde yapılan diğer bir çalışma ise, var olan sözlük tabanlı yöntemlere alternatif olabilecek yeni bir yöntemin geliştirilmesi üzerine olmuştur.

Kayıplı ve kayıpsız veri sıkıştırma yöntemleri, çalışma biçimlerine ve uygulama alanlarına göre sınıflandırılarak farklı bölümlerde anlatılmışlardır. Tezin ikinci ve üçüncü bölümlerinde olasılık tabanlı kayıpsız sıkıştırma ve sözlük tabanlı kayıpsız sıkıştırma yöntemleri anlatılmış, üçüncü bölümün sonunda kendi geliştirdiğimiz yaklaşım da dâhil olmak üzere tüm kayıpsız sıkıştırma yöntemleri, sıkıştırma testine tâbi tutularak karşılaştırılmıştır.

Ses, görüntü ve hareketli görüntü sıkıştırma yöntemleri, sırasıyla tezin dördüncü, beşinci ve altıncı bölümlerinde açıklanmışlardır. Bu yöntemlerin özellikle kayıplı olanları birçok sıkıştırma algoritmasının bir arada kullanılması ile oluşturulmuş karmaşık tekniklerdir. Bu yöntemler açıklanırken detaya girilmeden, yöntemin işleyişi ile ilgili genel bilgi verilmiştir. Her bölümün sonunda, hangi yöntemlerin hangi tip veriler üzerinde daha etkili olduğunun gösterilmesi amacıyla gerçekleştirdiğimiz karşılaştırma sonuçları yer almaktadır. Beşinci bölümde yer alan kayıpsız görüntü sıkıştırma yöntemlerinin karşılaştırılmasına geliştirdiğimiz algoritma da dâhil edilmiş ve karmaşıklığı az olan görüntü dosyalarında iyi sonuçlar verdiği görülmüştür.

2006, 156 sayfa

Anahtar Kelimeler: Veri sıkıştırma, Olasılık tabanlı kodlama, Sözlük tabanlı kodlama, Huffman kodlaması, LZ77, LZW, JPEG, MPEG, AAC, AVC.

ABSTRACT

The purpose of this thesis is to study lossy and lossless data compression methods which are widely used today, determine differences between formerly developed methods and recently developed methods and research how new methods enhance former ones. Another work in this thesis is to develop a new dictionary based method, which can be an alternative to current dictionary base methods.

Lossy and lossless data compression methods are classified according to their working types and application areas and described in different sections. In second and third chapters, probability-based lossless compression and dictionary-based lossless compression methods are explained. All of these lossless compression methods, including our new approach, are compared with a compression test at the end of the third chapter.

Audio, image and video compression methods are explained in fourth, fifth and sixth chapters respectively. Most of these methods, especially lossy ones, are complex techniques which are formed with many compression algorithms used together. These methods are not described in detail, only some general information about them is given. At the end of each chapter, the comparison results are given to show which methods are more effective on which type of data. Our compression algorithm is also included to the comparison of lossless image compression methods in the fifth chapter, and it is seen that it performs well in low complexity images.

2006, 156 pages

Key Words: Data Compression, Probability based coding, Dictionary based coding, Huffman coding, LZ77, LZW, JPEG, MPEG, AAC, AVC.

TEŞEKKÜR

Öncelikle, veri sıkıştırma alanında çalışmam için beni teşvik eden, bana yol gösteren, destek ve yardımlarını benden esirgemeyen danışman hocam Yrd. Doç. Dr. Aydın CARUS'a teşekkür ederim.

Ayrıca, tez izleme komitesi ve tez jürisinde yer alan Yrd. Doç. Dr. Ercan BULUŞ ve Yrd. Doç. Dr. Şaban AKTAŞ'a, tez jürisinde yer alan Prof. Dr. Mesut RAZBONYALI ve Yrd. Doç. Dr. Rembiye KANDEMİR'e, tez çalışmama yaptıkları olumlu eleştiriler ile katkıda bulundukları için teşekkürlerimi sunarım.

Son olarak, çalışabilmem için gerekli ortamın yaratılmasını sağlayan ve tüm yoğun zamanlarımda anlayışlı davranan Arş. Gör. Andaç ŞAHİN'e ve aileme teşekkür ederim.

İÇİNDEKİLER

KISALTMALAR LİSTESİ

3GPP	3rd Generation Partnership Project
AAC	Advanced Audio Coding
ADPCM	Adaptive Differential Pulse Code Modulation
ALS	Audio Lossless Coding
ASCII	American Standard Code for Information Interchange
ASO	Arbitrary Slice Ordering
ASP	Advanced Simple Profile
AVC	Advanced Video Coding
BWCA	Burrows-Wheeler Compression Algorithm
BWT	Burrows-Wheeler Transform
CAVLC	Context-Adaptive Variable Length Coding
CABAC	Context-Adaptive Binary Arithmetic Coding
CBR	Constant Bit Rate
CCITT	Consultive Committee on International Telephone and Telegraph
CIF	Common Interchange Format
CPB	Constrained Parameter Bitstream
CREW	Compression with Reversible Embedded Wavelets
CWT	Continuous Wavelet Transform
DCT	Discrete Cosine Transform
DPCM	Differential Pulse Code Modulation
DVD	Digital Versatile Disc (Digital Video Disc)
DWT	Discrete Wavelet Transform
EBCOT	Embedded Block-based Coding with Optimized Truncation
EBU	European Broadcasting Union
FFT	Fast Fourier Transform
FGS	Fine Granularity Scalability
FMO	Flexible Macroblock Ordering
GIF	Graphics Interchange Format
GMC	Global Motion Compensation
GOP	Group Of Pictures
GOV	Group Of VOPs (Video Object Planes)
HD-TV	High Definition TeleVision

HE-AAC	High Efficiency AAC (Advanced Audio Coding)
ICT	Irreversible Component Transform
IEC	International Electrotechnical Commission
ISO	International Organisation for Standardisation
ISSDC	Iterative Semi Static Digram Coding
ITU	International Telecommunication Union
JBIG	Joint Bi-level Image experts Group
JPEG	Joint Photographic Experts Group
JVT	Joint Video Team
LOCO-I	LOw COmplexity LOssless COmpression for Images
LPC	Linear Predictive Coding
LTP	Long Term Prediction
LZ77	Lempel Ziv 77
LZ78	Lempel Ziv 78
LZSS	Lempel Ziv Storer Szymanski
LZW	Lempel Ziv Welch
MAE	Mean Absolute Error
MDCT	Modified DCT (Discrete Cosine Transform)
MH	Modified Huffman
MMR	Modified Modified READ
MOPS	Millions of Operations Per Second
MP3	MPEG Layer III
MPEG	Moving Picture Experts Group
MS	Middle Side
MSE	Mean Squared Error
MSU	Moscow State University
MTF	Move-To-Front
MUSHRA	MUltiple Stimulus test with Hidden Reference and Anchors
MVP	Multi View Profile
PAE	Peak Absolute Error
PCM	Pulse Code Modulation
PCU	Processor Complexity Units
PEG	Photographic Experts Group
PNG	Portable Network Graphics

PPM	Prediction by Partial Matching
PSNR	Peak SNR (Signal-to-Noise Ratio)
QCIF	Quarter CIF (Common Interchange Format)
RAM	Random Access Memory
RCT	Reversible Component Transform
RCU	RAM Complexity Units
RDO	Rate Distortion Optimisation
RLE	Run Length Encoding
RMSE	Root MSE (Mean Squared Error)
ROI	Region of Interest
RS	Redundant Slice
RVLC	Reversible Variable Length Coding
SA-DCT	Shape-Adaptive DCT (Discrete Cosine Transform)
SBR	Spectral Band Replication
SNR	Signal-to-Noise Ratio
SSDC	Semi Static Digram Coding
TIFF	Tagged Image File Format
VBR	Variable Bit Rate
VCEG	Video Coding Experts Group
VLBV	Very Low Bitrate Video
VLC	Variable Length Coding
VO	Video Object
VOL	Video Object Layer
VOP	Video Object Plane
VS	Video Session
WMA	Windows Media Audio

1. GİRİŞ

Veri sıkıştırma yöntemleri, verilerin saklama ortamlarında daha az yer işgal etmeleri için ve bir iletişim ağı üzerinden daha hızlı transfer edilebilmeleri için yaygın olarak kullanılmaktadırlar. Son yıllarda disk kapasitelerinin hızlı bir şekilde artması, genel amaçlı sıkıştırma uygulamalarının kullanımını azalttıysa da, aslında sabit disklerimizde sakladığımız ses, görüntü ve hareketli görüntü dosyalarının tamamına yakını çeşitli yöntemlerle sıkıştırılmış haldedir.

Veri sıkıştırma yöntemleri, sıkıştırma biçimlerine göre kayıplı sıkıştırma yöntemleri ve kayıpsız sıkıştırma yöntemleri olmak üzere ikiye ayrılırlar. Kayıplı sıkıştırma yöntemleri, çıkartılması verinin bütünlüğünü en az düzeyde etkileyecek olan veri kümelerini çıkartarak, geriye kalan veri kümelerinin de kayıpsız sıkıştırmaya tâbi tutulması temeline dayanır. Bir veri kayıplı bir sıkıştırma yöntemi ile sıkıştırılırsa, verinin tamamı değil, sadece belirli bir kısmı geri getirilebilir. Veri birebir aynı şekilde geri getirilemediği için bu tür yöntemlere kayıplı yöntemler denir. Kayıplı veri sıkıştırma genellikle belirli bir miktar veri kaybının insan gözü ve kulağı tarafından hissedilemeyeceği durumlarda, örneğin fotoğraf görüntüleri, video ve ses için kullanılır. İnsan gözü ve kulağı yüksek frekans değerlerine daha az hassasiyet gösterdiği için, genellikle veri eleme işlemi yüksek frekans değerlerinin simgeleyen veriler üzerinde yapılır.

Kayıpsız sıkıştırma yöntemleri, orijinal veri ile sıkıştırıldıktan sonra geri getirilecek olan verinin tamamıyla aynı olmasının gerekli olduğu durumlarda kullanılır. Örneğin metin tipinde veriler kayıpsız olarak sıkıştırılmalıdırlar, çünkü geri getirildiklerinde kelimelerinde veya karakterlerinde eksiklikler olursa, metnin okunabilirliği azalacak ve hatta anlam kayıpları meydana gelebilecektir. Kısacası, insan gözünün ve kulağının hassasiyeti ile direkt olarak ilgisi bulunmayan, metin belgeleri, kaynak kodları, çalıştırılabilir program dosyaları gibi dosyalar kayıpsız sıkıştırılmak zorundadırlar. Fotoğraf görüntüleri, video ve ses için de daha hızlı çalışan kayıpsız sıkıştırma yöntemleri kullanılabilir, fakat sıkıştırma oranları kayıplı sıkıştırmaya göre oldukça düşük olacaktır.

Veri sıkıştırma yöntemlerini uygulama alanlarına göre sınıflandıracak olursak:

- *Metin ve ikili tabanlı veri sıkıştırma* alanında, her bit değerli olduğu için kayıpsız yaklaşımların kullanılması şarttır.
- *Ses verisi sıkıştırma* alanında, iyi sıkıştırma oranları sağladığı için genellikle kayıplı yaklaşımlar tercih edilirken, her bitin önemli olduğu profesyonel amaçlı uygulamalar için ise kayıpsız yaklaşımlar kullanılır.
- *Görüntü verisi sıkıştırma* alanında, şekiller ve taranmış metin görüntüleri gibi düşük frekanslı görüntülerde kayıpsız, fotoğraflarda ise genellikle kayıplı sıkıştırma kullanılır.
- *Hareketli görüntü (video) sıkıştırma* alanında, ses verisi sıkıştırma alanında olduğı gibi, profesyonel amaçlı bazı uygulama alanları haricinde genellikle kayıplı sıkıştırma kullanılır.

İki tip kayıpsız sıkıştırma yöntemi vardır. Bunlardan ilki değişken uzunluklu kodlama olarak ta bilinen olasılık (veya istatistik) tabanlı kodlama, ikincisi ise sözlük tabanlı kodlamadır. Olasılık tabanlı kodlamada, sıkıştırılan verinin bütünü içinde daha sık kullanılan sembollere bit adedi olarak daha küçük boyutta kodlar atanması prensibi ile sıkıştırma yapılır. En çok kullanılan olasılık tabanlı teknikler; *Huffman Kodlaması* ve *Aritmetik Kodlama*'dır. Tezin ikinci bölümünde bu teknikler açıklanmıştır. Sözlük tabanlı kodlamada ise, sık tekrarlanan sembol grupları için tek bir sembol kullanılması ile sıkıştırma yapılır. En çok kullanılan sözlük tabanlı teknikler; *LZ77*, *LZ78* ve *LZW* yöntemleridir. Bu yöntemler ise tezin üçüncü bölümünde açıklanmıştır. Hem olasılık tabanlı kodlamayı, hem de sözlük tabanlı kodlamayı bir arada kullanan algoritmalar, daha yüksek sıkıştırma oranları sağlarlar. *DEFLATE*, Huffman ve LZ77 sıkıştırma tekniklerini bir arada kullanan bir algoritmadır ve bu algoritma da tezin üçüncü bölümünde açıklanmıştır. Üçüncü bölümde ayrıca, sözlük tabanlı kayıpsız sıkıştırma yöntemlerine yeni bir alternatif olması için tezin bir parçası olarak geliştirilen, *SSDC* ve tekrarlamalı biçimi olan *ISSDC* açıklanmıştır.

Dördüncü bölümde önce kayıpsız ses sıkıştırma yöntemlerine değinilmiş, sonra günümüzde en çok kullanılan kayıplı ses sıkıştırma yöntemi olan MP3 standardı anlatılmış ve daha sonra da MP3'ün yerini alması beklenen AAC standardına yer

verilmiştir. Bu bölümün sonunda, yaygın olarak kullanılan ses sıkıştırma yöntemlerinin uzman kuruluşlarca yapılan karşılaştırma sonuçları yer almaktadır.

Beşinci bölümde yaygın olarak kullanılan kayıpsız görüntü sıkıştırma yöntemleri ve dosya formatları anlatıldıktan sonra, hem görüntü sıkıştırma için özel tasarlanmış bu yöntemlerin, hem de içinde ISSDC'nin de bulunduğu diğer genel amaçlı kayıpsız sıkıştırma yöntemlerinin, belirlenen görüntü dosyaları üzerinde sıkıştırma oranları ölçülmüştür. Bu bölümde ayrıca, kayıplı görüntü sıkıştırma denilince akla ilk gelen standart olan JPEG, ve daha yeni bir standart olan JPEG2000 genel hatları ile açıklanmış, bu iki standardın düşük sıkıştırma oranlarındaki etkinliğini ölçmek amacıyla yapılan bir karşılaştırmaya da yer verilmiştir.

Genellikle hareketli görüntü sıkıştırmada kayıplı yaklaşımlar tercih edildiği için, altıncı bölümde kayıpsız hareketli görüntü sıkıştırmaya çok az değinilmiştir. Bu bölümde, kayıplı hareketli görüntü sıkıştırma alanında en yaygın olarak kullanılan MPEG standartları temel özellikleri ile anlatılmıştır. Bu alandaki yeni standartlar olan MPEG-4 ve AVC'nin, eski standartlar olan MPEG-1 ve MPEG-2 ile ortak olan özelliklerinden bahsedilmemiş, sadece kullandıkları yeni özelliklere değinilmiştir. Bu bölümün sonunda, yaygın olarak kullanılan kayıplı hareketli görüntü sıkıştırma yöntemlerinin uzman kişiler tarafından yapılan karşılaştırmaları, sonuçları ile birlikte yer almaktadır.

2. OLASILIK TABANLI TEKNİKLER

Sıkıştırılması istenen mesajın (semboller kümesinin) tek tek tüm sembollerinin veya birkaç sembolün bir araya getirilmesi ile oluşturulan alt sembol kümelerinin olasılıklarının bulunması, ve bu olasılık dağılımlarını temel alarak mesajın tekrar kodlanmasına *olasılık kodlaması* ve buna dayalı tekniklere de *olasılık tabanlı teknikler* adı verilir.

ASCII kodlamasında her sembolü temsil etmek için 8-bit uzunluğunda bir kod kullanıldığından, bu kodlama türüne *sabit uzunluklu kodlama* denir. Eğer bir mesajın daha az bit ile ifade edilmesi isteniyorsa, o mesajı oluşturan birbirinden farklı sembollerin, farklı sayıda bit ile ifade edilmesi, yani *değişken uzunluklu kodlama* (variable length coding - VLC) yapılması gerekir. Eğer mesajda daha sık kullanılan semboller daha az bit ile ifade edilirse, mesajın tamamı için sembol başına kullanılan bit sayılarının ortalaması (ortalama uzunluk) azaltılmış olunur. Örneğin Mors alfabesinde kullanılan kodlamada, daha sık kullanılan karakterler daha kısa kodlar ile, daha seyrek kullanılan karakterler ise daha uzun kodlar ile gösterilirler. Örneğin ‘E’ karakterinin kodu ‘▪’ iken ‘Z’ karakterinin kodu ‘▬ ▬ ▪ ▪’ şeklindedir. Bu açıdan bakıldığında Mors kodlaması bir veri sıkıştırma örneği olarak görülebilir. Çünkü bu kodlamanın amacı veri sıkıştırmanın amacı ile aynıdır: *Bir mesajı karşı tarafa daha çabuk ulaştırmak*. Günümüzde telgraf haberleşmesi yerini e-posta haberleşmesine bırakmış olsa da, aslında bir e-posta’nın alıcısına daha çabuk ulaştırılması için, arka planda telgrafta kullanılan Mors kodlamasına benzer bir yaklaşımın kullanıldığı söylenebilir.

En çok kullanılan olasılık tabanlı teknikler Huffman Kodlaması ve Aritmetik Kodlama’dır. Bu tekniklerin anlatılmasına başlanmadan önce, entropi (entropy) ve önek kodu (prefix code) kavramlarını açıklamak gerekir.

2.1. Entropi

İstatistiksel fizikte kullanılan entropi kavramı, bilişim teorisine, bu teorinin yaratıcısı kabul edilen Claude Elwood Shannon tarafından uyarlanmıştır [Shannon, 1948]. Shannon öz bilgi (self information) olarak adlandırdığı bir nicelikten bahseder.

Eğer bir A olayının gerçekleşmesi olasılığı P(A) ise, A ile ilgili öz bilgi $i(A)$ ile ifade edilir ve aşağıdaki formül ile bulunur;

$$i(A) = \log_x \frac{1}{P(A)} = -\log_x P(A) \qquad (2.1)$$

Bilginin birimi logaritmanın tabanına bağlıdır. Eğer taban 2 ise birim bit'tir, *e* ise birim nat'tır, 10 ise birim hartley'dir. Bilgisayar dünyasında 0 ve 1'lerle yani bit'lerle çalıştığımız için, logaritmanın tabanını 2 olarak kabul edeceğiz:

$$i(A) = -\log_2 P(A) \qquad (2.2)$$

Bu formülde verilen öz bilgi, o bilgiyi ifade edebilmek için kaç bit kullanılması gerektiğini gösterir. Bu eşitlik bize, yüksek olasılığa sahip mesajların düşük bilgi içerdiğini, düşük olasılığa sahip mesajların ise yüksek bilgi içerdiğini gösterir. Örneğin, "Edirne'de 19 Mayıs günü kar yağacak" mesajı düşük olasılıklıdır, fakat "Edirne'de 19 Mayıs günü güneşli bir hava olacak" mesajına göre daha fazla bilgi vericidir.

Entropi, her sembolün veya semboller kümesinin öz bilgisinin ağırlıklı ortalamasıdır:

$$\mathrm{H} = -\sum_{i=1}^{n} P(s_i) \log_2 P(s_i) \qquad (2.3)$$

Bu formülde yer alan *n*, kodlanacak mesajda yer alan her bir farklı sembolün sayısıdır. $P(s_i)$ ise *i*. sembolün mesajda bulunma olasılığıdır. Formülden anlaşıldığı gibi mesajın içerdiği bilgi fazlalaştıkça, entropi büyüyecektir.

Örnek 2.1: "karapara" mesajının entropisini hesaplayalım. Mesajda kullanılan sembollerin olasılık dağılımı aşağıdaki gibidir:

$$P(a) = \frac{1}{2},\ P(r) = \frac{1}{4},\ P(k) = P(p) = \frac{1}{8}$$

Bu olasılıkları 2.3 formülüne uygularsak, "karapara" mesajının entropisini 1,75 olarak kolaylıkla bulabiliriz:

$$H = -\left[\left(\frac{1}{2}\log_2\frac{1}{2}\right)+\left(\frac{1}{4}\log_2\frac{1}{4}\right)+\left(\frac{1}{8}\log_2\frac{1}{8}\right)+\left(\frac{1}{8}\log_2\frac{1}{8}\right)\right]$$

$$H = -\left[\left(-\frac{1}{2}\right)+\left(-\frac{2}{4}\right)+\left(-\frac{3}{8}\right)+\left(-\frac{3}{8}\right)\right] = \frac{14}{8} = 1{,}75$$

Entropi kayıpsız sıkıştırma için limiti belirler. Yani bu örnekten çıkarılabilecek sonuç; sembolleri gruplamadan kodlama yapan en iyi kayıpsız sıkıştırma kodlayıcısının "karapara" mesajını sembol başına en az 1,75 bit kullanarak kodlayabileceğidir.

2.2. Önek Kodu

Sabit uzunluklu kodları değişken uzunluklu hale getirerek sıkıştırma yapılırken karşılaşılabilecek en büyük sorun, kodlama (encoding) sırasında uzunlukları değiştirilen semboller arka arkaya eklenerek sıkıştırılmış mesaj oluşturulduğu için, daha sonra kod çözme (decoding) yapılırken hangi bit'in hangi sembolün bir parçası olduğunun bilinememesidir. Örneğin 4 karaktere sahip bir alfabede a, b, c ve d sembollerine karşılık gelen değişken uzunluktaki kod değerleri {(**a=1**), (**b=01**), (**c=101**), (**d=011**)} şeklinde seçilirse, **1011** bit serisine kod çözme işlemi yapıldığında **aba**, **ca** veya **ad** sonuçlarından herhangi biri çıktı olarak üretilebilir. Bu belirsizlik durumundan kaçınmak için iki ardışık sembolü birbirinden ayıran belirli bir sembol kullanılabilir, veya her sembolden önce o sembolün uzunluğunu bildiren bir ekstra sembol yer alabilir. Ne var ki, bu çözümler fazladan veri eklenmesi gerekliliği sebebiyle kullanışlı değildirler. Bu yöntemler yerine, her bit serisi için tek bir çıktı üreten *yalnız bir şekilde çözülebilir* (*uniquely decodable*) kodlar tasarlamak daha etkili bir çözüm olacaktır.

Önek kodu, hiçbir sembolün bir diğerinin öneki olmadığı özel bir yalnız bir şekilde çözülebilir kod tipidir. Bir önceki örneği önek koduna uygun olacak şekilde {(**a=1**), (**b=01**), (**c=000**), (**d=001**)} biçiminde tekrar kodlarsak, **1011** bit serisine kod çözme işlemi yapıldığında sadece **aba** çıktı olarak üretilebilecektir.

Önek kodunun diğer yalnız bir şekilde çözülebilir kodlara göre iki önemli avantajı vardır:

- Önek kodu ile kodlanan mesajların ortalama uzunlukları, diğer kodların ortalama uzunluklarına göre daha küçük, dolayısıyla entropiye daha yakın olur. Yani bu mesajlar önek kodu ile daha fazla sıkıştırılabilir.
- Önek kodu ile kodlanan bir mesajın bit serisi içinde, bir bitin bir sembolün son biti olduğu bir sonraki sembole gelmeden de anlaşılabilir.

Örnek 2.2: Verilen tanımların daha iyi anlaşılabilmesi için örnek olarak daha önce entropisini hesapladığımız "karapara" mesajını üç farklı biçimde değişken uzunluklu kodlayarak inceleyelim:

Çizelge 2.1. "karapara" için oluşturulan değişken uzunluklu kodlar

Karakterler	***1. Kod***	***2. Kod***	***3. Kod***
a	0	0	0
r	1	10	01
k	01	110	011
p	10	111	0111
Ortalama Sembol Uzunluğu	**1.25**	**1.75**	**1.875**

Ortalama uzunluk hesaplanırken kullanılan formül aşağıda verilmiştir;

$$\text{Ortalama Uzunluk} = \sum_{i=1}^{4} P(s_i)N(s_i)$$

Bu denklemdeki N değeri, sembol (örneğimizde karakter) için kullanılan bit sayısıdır. Ortalama uzunluk hesabının doğruluğunu "karapara" mesajını her üç kodlama ile kodlayıp, oluşan bit adedini mesajdaki karakter adedine bölerek sınayabiliriz.

1. Kodlama ile oluşan kod: **`0101010010`** (bit # / karakter # : 10/8 = 1.25)

2. Kodlama ile oluşan kod: **`11001001110100`** **(**bit # / kr # : 14/8 = 1.75)

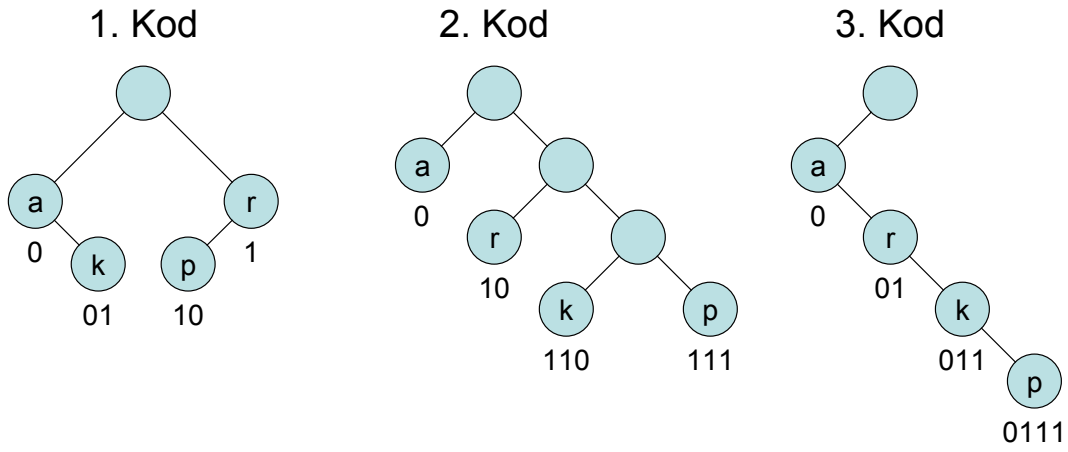
3. Kodlama ile oluşan kod: **`011001001110010`** **(**bit # / kr # : 15/8 = 1.875)

Birinci kod yalnız bir şekilde çözülebilir değildir, çünkü “kara” kodlanırken oluşturulan **01010** bit sırası, kod çözme aşamasında **arka**, **arara**, **kka**, **apra**, **app** veya **kap** gibi birçok farklı şekilde çözülebilir.

İkinci kod bir önek kodudur, dolayısıyla yalnız bir şekilde çözülebilir bir koddur. Hiçbir karakterin kodu bir diğer karakterin kodunun öneki değildir.

Üçüncü kod da ikinci kod gibi yalnız bir şekilde çözülebilir bir koddur. “kara” kodlanırken oluşturulan **0110010** bit sırası geri kodlamada sadece “kara” olarak üretilebilir. Fakat bu kod, ikinci kod gibi bir önek kodu değildir, çünkü a’nın kodu r’nin kodunun, r’nin kodu k’nın kodunun ve k’nın kodu da p’nin kodunun önekidir.

Bir kodun bir önek kodu olup olmadığını görebilmek için ikili ağaç yapısı ile temsil ederek, her sembolün ağacın yaprakları olup olmadığını kontrol edebiliriz. Her sembolün kodunun ağaç üzerinde kökten yaprağa doğru bir yolu takip ettiği, 0 gördüğünde sol dala, 1 gördüğünde ise sağ dala gittiği düşünüldüğünde, bu üç kod için ağaç yapıları Şekil 2.1’deki gibi oluşacaktır:



Şekil 2.1. “karapara” örneği için oluşturulan kodların ikili ağaç ile gösterimi

Şekil 2.1’de görüldüğü gibi ikinci kodda tüm semboller yapraklardadır. Bu durum ikinci kodun önek kodu olduğunu gösterir. Önek kodu olmayan birinci ve üçüncü kodlarda bazı semboller düğümlerde yer alır. Bu tür bir ağaç yapısı oluşturulması önek kodlarının çözülmesinde de yardımcı olur. Kod çözücü okuduğu

bit'in değerine göre sola ya da sağa giderek bir yaprağa ulaşmaya çalışır. Yaprağa ulaşınca ilgili sembolü üretir ve köke geri dönerek yeni bir bit okur. Bu işlemi bit serisi bitene kadar yapar.

Verilen örnekte önek kodunun iki önemli avantajı rahatlıkla görülebilir:

- Önek kodu olan ikinci kodun ortalama uzunluğu, önek kodu olmayan üçüncü kodun ortalama uzunluğundan daha azdır.
- Üçüncü kod ile oluşturulan "karapara" mesajında ilk üç bit okunduğunda (**011**) bu üç bitin 'k' karakterini temsil ettiği dördüncü bit okunmadan anlaşılamaz, bu üç bit 'p' karakterinin ilk üç bitini temsil ediyor da olabilir. İkinci kodda böyle bir durum söz konusu değildir.

Bu örnekte kullanılan ikinci kodun ortalama sembol uzunluğu daha önce hesaplanan entropiye eşit çıkmıştır. Bu durumda bu kodun bu mesaj için en iyi kod olduğunu söyleyebiliriz. Fakat her zaman entropiye eşit çıkacak kod üretmenin olanağı yoktur. Gerçekte bu eşitliği sağlayabilecek kodlar sadece sembollerin olasılıkları 2^{-k} (1/2, 1/4, 1/8, ...) biçiminde olduğunda bulunabilir.

2.3. Huffman Kodlaması

MIT'de Robert Fano tarafından kurulan - ve bilişim teorisi alanında bir ilk olan - sınıfta öğrenci olan David Huffman tarafından verilen bir ödev üzerine geliştirilmiştir [Huffman, 1952]. GZIP, JPEG, MPEG gibi yaygın olarak kullanılan sıkıştırma yöntemlerinde son işlem olarak kullanılır ve muhtemelen sıkıştırma algoritmalarında en yaygın olarak kullanılan bileşendir. CCITT'nin (Consultive Committee on International Telephone and Telegraph) faks iletimi için geliştirdiği 1-boyutlu kodlama, tipik bir Huffman kodlamasıdır.

Huffman kodlaması verilen bir model (olasılık kümesi) için en uygun önek kodunu oluşturur. Huffman kodlamasının nasıl çalıştığını bir örnek ile açıklayalım:

Örnek 2.3: Beş harfli bir $A = \{a_1, a_2, a_3, a_4, a_5\}$ alfabemiz olsun ve karakterlerin bulunma olasılıkları $P(a_1) = P(a_3) = 0{,}2$ ve $P(a_2) = 0{,}4$ ve $P(a_4) = P(a_5) = 0{,}1$ olsun. Bu

kaynak için entropi 2,122 bit/sembol'dür. Öncelikle karakterler Çizelge 2.2'de görüldüğü gibi azalan sırada sıralanmalıdır. Bu çizelgede $c(a_i)$ a_i'nin kodunu temsil etmektedir.

Çizelge 2.2. Başlangıçtaki beş harfli alfabe

Karakter	***Olasılığı***	***Kodu***
a_2	0,4	$c(a_2)$
a_1	0,2	$c(a_1)$
a_3	0,2	$c(a_3)$
a_4	0,1	$c(a_4)$
a_5	0,1	$c(a_5)$

En düşük olasılığa sahip iki sembol a_4 ve a_5'tir. Bu durumda, bu iki sembol ağacın en alt seviyedeki yaprakları olacaklar ve dolayısıyla en uzun koda sahip olacaklardır. İkisinin içerdiği bit sayıları (uzunlukları) eşit olacak, birinin son biti 0 ve diğerinin ki 1 olarak seçilecek, diğer tüm bitleri de birbirine eşit olacaktır. Bu iki sembolü birleştirerek yeni bir sembol haline getirirsek ve bu yeni sembolü b_1 olarak gösterirsek, $c(b_1)$ son bit haricindeki diğer tüm bitlerden oluşan kodu temsil edecektir. Bu durumda a_4 ve a_5 sembollerin kodları aşağıdaki gibi olacaktır:

$$c(a_4) = c(b_1)\ \&\ 0$$
$$c(a_5) = c(b_1)\ \&\ 1$$

Böylelikle beş harfli A alfabesi, harfleri a_1, a_2, a_3 ve b_1 olan dört harfli A' alfabesine dönüştürülmüş oldu. b_1 sembolünün olasılığı içerdiği iki sembolün olasılıklarının toplamına yani 0,2'ye eşit olacaktır. Yeni alfabe azalan sırada sıralanarak Çizelge 2.3 elde edilir.

Çizelge 2.3. Azaltılmış dört harfli alfabe

Karakter	***Olasılığı***	***Kodu***
a_2	0,4	$c(a_2)$
a_1	0,2	$c(a_1)$
a_3	0,2	$c(a_3)$
b_1	0,2	$c(b_1)$

Bu alfabede a_3 ve b_1 sıralanan listenin en altında yer aldıkları için birleştirilecek olan en düşük olasılığa sahip iki sembol olarak seçilirler. Bu iki sembolün birleşimine b_2 dersek, b_2'nin olasılığı da 0,2 + 0,2 = 0,4 olacaktır. Bu iki sembolün kodları da;

$$c(a_3) = c(b_2)\ \&\ 0$$
$$c(b_1) = c(b_2)\ \&\ 1$$

şeklinde olacaktır. Bu durumda a_4 ve a_5 sembollerinin kodları da;

$$c(a_4) = c(b_2)\ \&\ 10$$
$$c(a_5) = c(b_2)\ \&\ 11$$

şeklinde tekrar yazılabilir.

Bu birleşim sonucunda dört harfli *A*' alfabesi bir harf daha azalarak *A*" haline gelmiş, ve bu yeni alfabenin azalan sırada sıralanmış hali Çizelge 2.4'te gösterilmiştir.

Çizelge 2.4. Azaltılmış üç harfli alfabe

Karakter	***Olasılığı***	***Kodu***
a_2	0,4	$c(a_2)$
b_2	0,4	$c(b_2)$
a_1	0,2	$c(a_1)$

A" alfabesinin son iki sırasındaki a_1 ve b_2 sembolleri birleştirilerek, olasılığı 0,6 olan b_3 elde edilip alfabe küçültme işlemine devam edilirse;

$$c(b_2) = c(b_3)\ \&\ 0$$
$$c(a_1) = c(b_3)\ \&\ 1$$

elde edilir. Bu durumda a_3, a_4 ve a_5 sembollerinin kodları da;

$$c(a_3) = c(b_3)\ \&\ 00$$
$$c(a_4) = c(b_3)\ \&\ 010$$
$$c(a_5) = c(b_3)\ \&\ 011$$

şeklinde tekrar yazılabilir.

Yeni oluşan iki harfli A''' alfabesi sıralanmış hali ile Çizelge 2.5'te görülmektedir.

Çizelge 2.5. Azaltılmış iki harfli alfabe

Karakter	***Olasılığı***	***Kodu***
b_3	0,6	$c(b_3)$
a_2	0,4	$c(a_2)$

Kalan son iki sembol için kodlama aşağıdaki gibi seçilebilir:

$$c(b_3) = 0$$
$$c(a_2) = 1$$

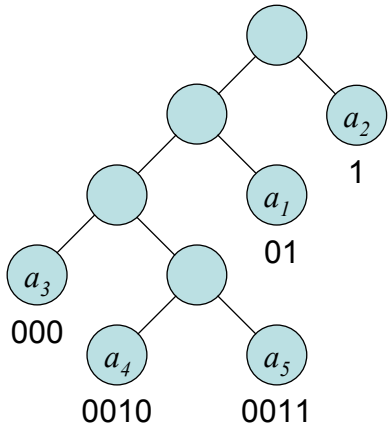
Bu durumda, b_3 sembolünün açılımını yaptığımız zaman, ilk alfabemiz olan A alfabesindeki a_2 haricindeki diğer harflerin kodları da aşağıdaki gibi olacaktır:

$$c(a_1) = 01$$
$$c(a_3) = 000$$
$$c(a_4) = 0010$$
$$c(a_5) = 0011$$

Yapılan bu işlemler neticesinde oluşturulan, A alfabesi için en uygun önek kodu olan Huffman kodu Çizelge 2.6'da, ikili ağaç şeklindeki gösterimi de Şekil 2.2'de verilmiştir.

Çizelge 2.6. beş harfli A alfabesi için Huffman kodu

Karakter	***Olasılığı***	***Kodu***
a_2	0,4	1
a_1	0,2	01
a_3	0,2	001
a_4	0,1	0010
a_5	0,1	0011



Şekil 2.2. *A* alfabesi için Huffman kodu ikili ağaç ile gösterimi

Bu kod için ortalama uzunluk 2,2 bit/sembol olarak aşağıdaki gibi hesaplanabilir:

$$l = 0{,}4 \text{ x } 1 + 0{,}2 \text{ x } 2 + 0{,}2 \text{ x } 3 + 0{,}1 \text{ x } 4 + 0{,}1 \text{ x } 4 = 2{,}2 \text{ bit/sembol}$$

Olasılıklar 2'nin negatif üsleri olarak dağılım göstermediği için ortalama uzunluk entropiye eşit çıkmamıştır, ama aradaki fark sadece 0,078 bit/sembol'dür.

2.3.1. Minimum Değişimli Huffman Kodları

Eşit olasılığa sahip olan sembolleri sıralarken birleştirilmiş sembolleri yukarıya alırsak farklı bir Huffman kodu oluşacaktır. Önceki örneğimizde Çizelge 2.3'teki sıralamayı Çizelge 2.7'deki gibi yapacak olursak sonraki adımlar bu durumdan etkilenecektir.

Çizelge 2.7. Azaltılmış dört harfli alfabe

Karakter	***Olasılığı***	***Kodu***
a_2	0,4	$c(a_2)$
b_1	0,2	$c(b_1)$
a_1	0,2	$c(a_1)$
a_3	0,2	$c(a_3)$

Bu durumda a_1 ve a_3 birleştirilecek ve Çizelge 2.8'deki gibi bir alfabe oluşturulacaktır.

Çizelge 2.8. Azaltılmış üç harfli alfabe

Karakter	*Olasılığı*	*Kodu*
b_2	0,4	$c(b_2)$
a_2	0,4	$c(a_2)$
b_1	0,2	$c(b_1)$

Bir sonraki adımda ise a_2 ve b_1 birleştirilecek ve Çizelge 2.9 oluşturulacaktır.

Çizelge 2.9. Azaltılmış iki harfli alfabe

Karakter	*Olasılığı*	*Kodu*
b_3	0,6	$c(b_3)$
b_2	0,4	$c(b_2)$

Böylece Huffman kodu Çizelge 2.10'daki gibi gerçekleşmiş olacaktır.

Çizelge 2.10. Minimum değişimli Huffman kodu

Karakter	*Olasılığı*	*Kodu*
a_2	0,4	00
a_1	0,2	10
a_3	0,2	11
a_4	0,1	010
a_5	0,1	011

Bu kod için ortalama uzunluk hesaplanırsa yine 2,2 bit/sembol sonucu bulunacaktır:

$$l = 0{,}4 \times 2 + 0{,}2 \times 2 + 0{,}2 \times 2 + 0{,}1 \times 3 + 0{,}1 \times 3 = 2{,}2 \text{ bit/sembol}$$

İki Huffman kodu aynı ortalama uzunluğa sahip olsa da, kod uzunluklarının değişimi çok farklıdır. Daha önce oluşturulan kodda, uzunluklar 1 ile 4 arasında 3 birim değişirken, yeni oluşturulan kodda 2 ile 3 arasında sadece 1 birim değişmektedir.

Her iki kodun da sıkıştırma oranı aynı olacaktır, fakat sabit bant genişliğine ve tampon büyüklüğü sınırlamalarına sahip modem sıkıştırması gibi bir uygulama söz konusu ise minimum değişimli Huffman kodu daha elverişlidir. Örneğin saniyede 57.600 bit veri gönderme kapasitesine sahip bir modemle 20.000 adet a_4 veya a_5 sembolünün bir saniyede gönderilmek istendiği düşünülürse, minimum değişimli kod

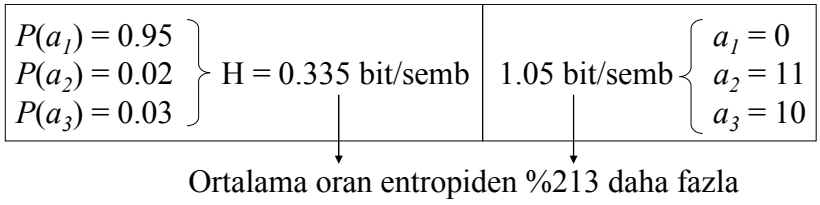
60.000 bit/s büyüklüğünde bir bant genişliğine, diğer kod ise 80.000 bit/s bant genişliğine ihtiyaç duyacaktır. Bu durumda minimum değişimli kod tampon bellekte sadece 2.400 bit veri tutarken, diğer kod 22.400 bit veri tutmak isteyecektir. Minimum değişimli kod tüm veriyi yaklaşık olarak 1,04 saniyede gönderirken, diğer kod ise 1,38 saniyede gönderebilecektir.

2.3.2. Uyarlanır Huffman Kodlaması

Huffman kodlaması, kodlanacak kaynağın olasılık bilgisine ihtiyaç duyar. Eğer bu bilgi yoksa, Huffman kodlaması iki geçişli bir prosedür haline gelir: ilk geçişte olasılık bilgileri elde edilir, ikinci geçişte kaynak kodlanır. Önceden karşılaşılan sembollerin istatistiği temel alınarak, *Uyarlanır (Dinamik) Huffman Kodlaması* olarak bilinen yaklaşım ile, Huffman kodlaması tek geçişli hale getirilebilir. Fakat her bir sembol kodlanırken, daha önce kodlanmış olan sembollerin istatistiğini çıkarmak ve bu istatistiğe göre Huffman kodlamasını yapmak çok masraflı olacaktır. Zamanla, daha az hesaplama yaparak uyarlanır kodlama yapabilen birçok farklı yöntem geliştirilmiştir [Faller, 1973] [Gallagher, 1978] [Knuth, 1985] [Vitter, 1987] [Lin, 1991].

2.3.3. Huffman Kodlamasının Etkinliği

Alfabenin küçük olduğu ve karakterlerin belirme olasılığında büyük farklar olduğu durumlarda, Huffman kodlamasının etkinliği azalır. Olasılıklar arasında büyük farkların olduğu aşağıdaki örneğe bakacak olursak, Huffman kodlamasının 1.05 bit/sembol olan ortalama uzunluğunun, entropinin (0.335 bit/sembol) yaklaşık üç katı olduğu görülmektedir.



Birkaç sembolü bir araya getirerek entropi ile ortalama uzunluk arasındaki farkı azaltabiliriz.

$P(a_1a_1) = 0.9025$	
$P(a_1a_2) = 0.0190$	Genişletilmiş alfabe için ortalama
$P(a_1a_3) = 0.0285$	oran = 1.222 bit/sembol (Orjinal
$P(a_2a_1) = 0.0190$	alfabe cinsinden = 0.611 bit/sembol)
$P(a_2a_2) = 0.0004$	
$P(a_2a_3) = 0.0006$	Entropi (H) = 0.335 bit/sembol
$P(a_3a_1) = 0.0285$	
$P(a_3a_2) = 0.0006$	Halen entropiden %82 fazla
$P(a_3a_3) = 0.0009$	

Sembolleri birleştirmeye devam ederek 8 sembollük bloklar oluşturduğumuzu düşünürsek, daha kabul edilebilir değerlere ulaşırız. Fakat bu durumda alfabe büyüklüğü 6561 (3^8) olacaktır ki, bu büyüklük saklama kapasitesini arttırıp, işlem süresini de yavaşlatacağı için elverişli değildir.

Alfabenin küçük olduğu ve karakterlerin belirme olasılığında büyük farklar olduğu durumlarda, belirli sıralamaya sahip sembollere kod atayarak aynı uzunluktaki tüm olası sıralamalara kod oluşturma zorunluluğu getirmeyen bir yönteme ihtiyaç vardır. Aritmetik kodlama bu ihtiyacı karşılamıştır.

2.4. Aritmetik Kodlama

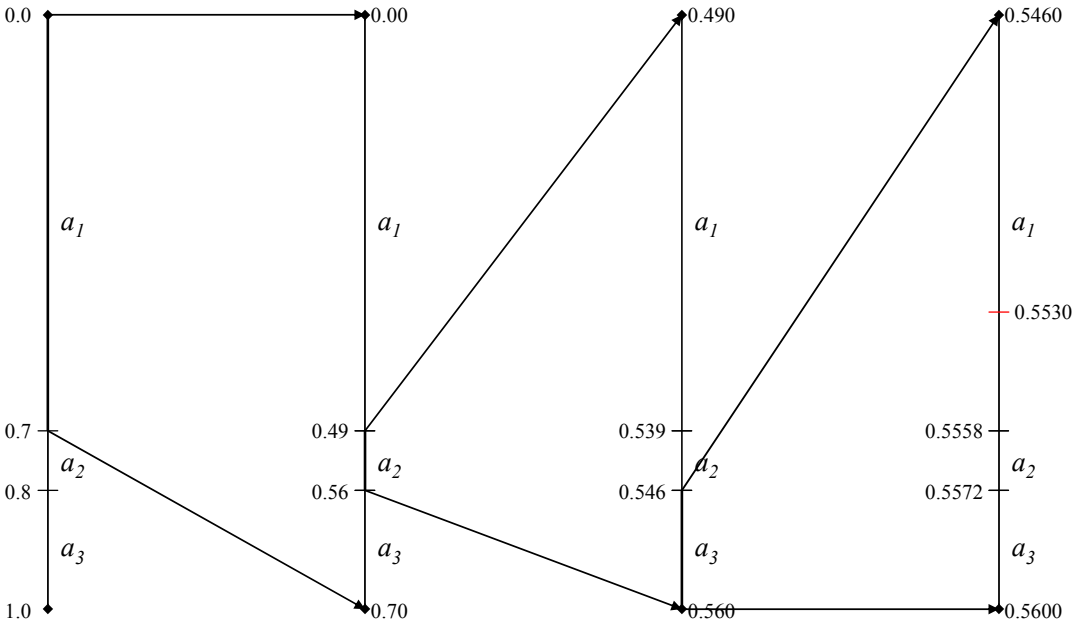
Shannon 1948'deki makalesinde, aritmetik kodlama teorisinin ispatına çok yakın bir kavramdan bahsetmiştir. Fano'nun MIT'deki bilişim teorisi sınıfının bir başka öğrencisi olan Peter Elias ise bu fikrin öz yinelemeli bir uyarlamasını gerçekleştirmiştir. Ama kendisi bu çalışmasını hiç yayınlamadığı için, biz bunu Abramson'un bilişim teorisi üzerine yayınladığı kitabından [Abramson, 1963] bilmekteyiz. Daha sonra Jelinek tarafından yazılan bir başka kitabın [Jelinek, 1968] ekler kısmında ise, aritmetik kodlama fikri değişken uzunluklu kodlamanın bir örneği olarak yer almıştır. Sonlu duyarlık sorununun çözülmesi, modern aritmetik kodlamanın başlangıcı olmuştur [Pasco, 1976] [Rissanen, 1976]. Pratik aritmetik kodlama algoritmaları [Rissanen ve Langdon, 1979] ve veri sıkıştırmada kullanılması [Witten vd., 1987] [Moffat vd., 1995] [Howard ve Vitter, 1994] hakkında birçok makale yazılmıştır.

Aritmetik kodlamanın temel fikri, *n* adet mesajın her olası serisini temsil etmek için 0 ile 1 arasındaki bir sayı aralığını (örneğin 0.2 ile 0.5 aralığı gibi) kullanmaktır.

Alfabenin küçük olduğu ve karakterlerin belirme olasılığında büyük farklar olduğu durumlarda, Huffman Kodlaması etkinliğini yitirirken, Aritmetik kodlama bu durumlarda daha başarılıdır.

Aritmetik kodlamada, belirli bir sembol serisini diğer sembol serilerinden ayırmak için, her serinin tekil bir belirleyici ile etiketlenmesi gerekir. Bu etiket (tag), genellikle 0 ile 1 arasında bir sayı şeklinde belirlenir.

Örnek 2.4: Üç harfli bir $A = \{ a_1, a_2, a_3 \}$ alfabesini ele alalım. Harflerin olasılık dağılımları $P(a_1) = 0.7$, $P(a_2) = 0.1$ ve $P(a_3) = 0.2$ olsun. Eğer a_1, a_2, ve a_3 harflerinin ardı ardına geldiği bir seri için etiketin bulunması istenirse, öncelikle 0 ile 1 sayıları aralığı olasılık dağılımları baz alınarak 3 farklı parçaya bölünür. 0 ile 0.7 arası a_1, 0.7 ile 0.8 arası a_2, ve 0.8 ile 1 arası da a_3 için tahsis edilir. Serinin ilk karakteri a_1 olduğu için, ikinci karaktere geçilirken, a_1'e tahsis edilen aralık yani 0 ile 0.7 aralığı genişletilir. Yeni aralık tekrar olasılık dağılımları baz alınarak 3 farklı parçaya bölünür. 0 ile 0.49 arası a_1, 0.49 ile 0.56 arası a_2, ve 0.56 ile 0.7 arası da a_3 için tahsis edilir. Serinin ikinci karakteri a_2 olduğu için 0.49 ile 0.56 aralığı seçilerek işleme devam edilir. Serinin üçüncü karakteri olan a_3 karakteri 0.546 ile 0.56 arasını tahsis edecektir, ve serinin etiketi olarak bu aralıkta yer alan herhangi bir sayısal değer seçilebilir.



Şekil 2.3. *A* alfabesinin a_1, a_2, a_3 sırası için aritmetik kodlamada etiketin bulunması

Şekil 2.3'te bu işlemin adımları soldan sağa doğru gösterilmektedir. Şekilde görüldüğü gibi bu örnekte en son aralığın orta noktası olan 0.553 değeri etiket olarak seçilmiştir. İkili sayı sistemindeki 0.100011 sayısı, onluk tabanda 0.546875 sayısına karşılık gelir, ve en son aralıkta (0.546 ile 0.56 aralığı) yer alan bu değer tam orta nokta olmasa bile etiket olarak kodlanabilir. Sadece noktadan sonraki basamakların (100011) alıcıya gönderilmesi yeterlidir.

Etiket olarak son aralığın orta noktasının alındığı bir aritmetik kodlama sistemi için, $T(x_i)$ etiketinin matematiksel olarak gösterimi aşağıdaki gibi yapılabilir:

$$T(x_i) = \sum_{y<x_i} P(y) + \frac{1}{2} P(x_i) \qquad (2.4)$$

Örneğin bir zarı iki defa attığımız zaman önce 1 sonra 3 gelmesi olasılığı için etiket oluşturmak istersek. Öncelikle her zarın gelme ihtimalini eşit kabul ederek, 0 ile 1 arasını 36 eşit parçaya bölebiliriz. Sonra ilk parçayı 1 ve 1 gelme olasılığına ($P(x_{11})$), ikinci parçayı 1 ve 2 gelme olasılığına ($P(x_{12})$), üçüncü parçayı 1 ve 3 gelme olasılığına ($P(x_{13})$), … vererek bu 36 eşit parçayı paylaştırabiliriz. 1'den sonra 3 gelmesi olasılığı için etiket değerinin hesaplanması aşağıdaki gibi olacaktır:

$$T(x_{13}) = P(x_{11}) + P(x_{12}) + \frac{1}{2} P(x_{13})$$

$$T(x_{13}) = \frac{1}{36} + \frac{1}{36} + \frac{1}{2}\left(\frac{1}{36}\right) = \frac{5}{72}$$

2.5. Olasılık Tabanlı Tekniklerin Uygulamaları

2.5.1. Öne Alma (Move-To-Front) Kodlaması

MTF olarak kısaltılan öne alma kodlaması, Burrows-Wheeler algoritmasının da arasında bulunduğu birçok başka algoritma tarafından bir ara işlem olarak kullanılır. Öne alma kodlamasının temel mantığı, mesajdaki karakterleri daha küçük değerlere sahip tamsayı değerlere çevirmektir. Bu tamsayı dizisi daha sonra Huffman ya da Aritmetik kodlama gibi bir çeşit olasılık tabanlı kodlama ile sıkıştırılır. Algoritma,

karşılaştığı karakterin alfabedeki sırasını kodlar ve o karakteri alfabenin ilk karakteri yapar. Örneğin [*a, b, c, d, ...*] alfabesi için *c* karakterinin kodlanması istenirse, *c*'nin alfabedeki sırası olan 3 kodlanır ve yeni alfabe c'nin başa taşınması ile [*c, a, b, d, ...*] şeklinde olur. Bu işlem sıkıştırılacak kaynak mesajının sonuna kadar tekrar edilir.

Eğer aynı karakterler mesaj içinde birbirlerine yakın yerleşmişlerse, kodlanacak tamsayı değerleri küçük olacak, böylece daha düzgün dağılım ve daha iyi sıkıştırma gerçekleşecektir.

2.5.2. Artık Değer (Residual) Kodlaması

Artık değer kodlaması da öne alma kodlaması gibi, birçok başka sıkıştırma tekniği tarafından bir ara işlem olarak kullanılır. Bu kodlamada da, veri üzerinde bir ön işlem gerçekleştirilip daha düzgün bir dağılım oluşturulur, ve sonrasında bu dağılım standart bir olasılık tabanlı kodlayıcı kullanılarak sıkıştırılır. Kodlayıcı, daha önce kodlanmış değerleri temel alarak bir sonraki mesajın (karakterin, pikselin, …) değerini tahmin etmeye çalışır ve tahmin edilen değer ile gerçek değerin farkını (artık değeri) kodlar. Bu fark mesajın değerinden daha düşük ise (öyle olması muhtemeldir), sıkıştırma oranı artacaktır. Kod çözücü (decoder), önceki içeriği zaten çözdüğü için kodlayıcı (coder) ile aynı tahmini yapabilir, ve daha sonra aldığı artık değeri kullanarak tahmini doğrular. Bu kodlama tekniği, artık değerin sadece belirli aralıklarla kullanılması ile kayıplı sıkıştırmada da kullanılabilir.

Artık değer kodlaması, hem renkli hem de gri-tonlamalı görüntülerin sıkıştırılmasında kullanılan JPEG-LS'de kullanılır.

2.5.3 PPM (Prediction by Partial Matching)

PPM algoritması ilk olarak 1984 yılında Cleary ve Witten tarafından yayınlanmış [Cleary ve Witten, 1984], daha sonra Moffat bu algoritmayı daha da geliştirerek PPMC'yi yaratmıştır [Moffat, 1988]. 90'lı yıllarda PPM'in bir çok farklı türevi ortaya çıkmıştır. PPM'in günümüzde en çok kullanılan türevi Dmitry Shkarin tarafından geliştirilmiş olan ve daha çok PPMd olarak bilinen PPMII'dır [Shkarin, 2002].

Kısmi eşleme yoluyla öngörü (tahmin) anlamına gelen PPM'in temel mantığı, bir karakterin ne olabileceğini o karakterden önceki birkaç karakteri (konteksti) kullanarak tahmin etmektir. Tahmin yapılırken sıkıştırılmakta olan girdinin daha önceki kısımlarından elde edilen, karakterlerin birbirlerini takip etme olasılıklarının saklandığı çizelge kullanılır. Bu çizelgedeki olasılık dağılımları Huffman veya aritmetik kodlayıcı ile sıkıştırılır. Olasılık aralığı çok geniş olabileceği için genellikle aritmetik kodlama tercih edilir. PPM'in çalışma biçimi Örnek 2.5 ile ayrıntılı olarak açıklanmıştır.

Örnek 2.5: *abracadabra* kelimesinin PPM kullanılarak sıkıştırıldığını düşünelim. PPM, bu kelimede yer alan karakterlerin ve o karakterlerden önce yer almış olan birli ve ikili karakter gruplarının var olma olasılıklarını Çizelge 2.11'de görülen bir biçimde saklar. Bu çizelgede *ts* kontekstin tekrar sayısını, *o* ise olasılığını ifade eder.

Çizelge 2.11. "abracadabra" karakter katarı işlendikten sonraki PPM modeli

Derece k = 2			**Derece k = 1**			**Derece k = 0**			**Derece k = -1**		
Tahminler	*ts*	*o*	Tahminler	*ts*	*o*	Tahminler	*ts*	*o*	Tahminler	*ts*	*o*
ab → r	2	2/3									
ab → *Esc*	1	1/3	a → b	2	2/7						
ac → a	1	1/2	a → c	1	1/7	→ a	5	$\frac{5}{16}$			
ac → *Esc*	1	1/2	a → d	1	1/7						
ad → a	1	1/2	a → *Esc*	3	3/7						
ad → *Esc*	1	1/2									
br → a	2	2/3	b → r	2	2/3	→ b	2	$\frac{2}{16}$			
br → *Esc*	1	1/3	b → *Esc*	1	1/3						
ca → d	1	1/2	c → a	1	1/2	→ c	1	$\frac{1}{16}$			
ca → *Esc*	1	1/2	c → *Esc*	1	1/2						
da → b	1	1/2	d → a	1	1/2	→ d	1	$\frac{1}{16}$			
da → *Esc*	1	1/2	d → *Esc*	1	1/2						
ra → c	1	1/2	r → a	2	2/3	→ r	2	$\frac{2}{16}$			
ra → *Esc*	1	1/2	r → *Esc*	1	1/3						
						→ *Esc*	5	$\frac{5}{16}$	→ *A*	1	$\frac{1}{\lvert A \rvert}$

Bu kelimeyi takip eden bir *a* karakteri olduğunu farz edelim. PPM öncelikle sakladığı modeller içinde en yüksek dereceye (*k*) sahip olandan tahmin yürütmeye çalışır. Çizelge 2.11'de görüldüğü gibi örneğimizde 4 model saklanmıştır ve 2 karakterli grupları takip eden karakterlerin olasılıklarının saklandığı en soldaki model en yüksek dereceye (2) sahiptir. Dolayısıyla kelimenin son iki karakteri olan *ra* karakter çiftini takip eden bir *a* karakterinin olasılığı aranır. Daha önce *ra* karakter çiftini sadece *c* karakteri takip etmiştir. O yüzden sadece *rac* üçlüsü için bir tahmin yapılabilmektedir. Bu durumda *ra* karakter çiftini takip eden diğer tüm karakterler kaçış (escape) olasılığını kullanmak zorundadır. Bu nedenle tüm olasılık dağılımları ile beraber *Esc* olarak kısaltılan bir de kaçış karakteri bulunmaktadır.

Kaçış karakterinin olasılığını belirlemek için çok farklı yaklaşımlar geliştirilmiştir. A yöntemi (PPMA) tüm kaçış karakterlerinin tekrar sayısını 1 olarak kabul ederken, örneğimizde kullandığımız C yöntemi (PPMC) ise ilgili karakter grubunu takip eden farklı karakter sayısını kullanmaktadır. Örneğin, 1. derecede *a* karakterini *b*, *c* ve *d* olmak üzere 3 farklı karakter takip ettiği için, *a* karakterinden sonraki kaçış karakterinin tekrar sayısı 3 olarak kabul edilmiştir. Hepsinin tekrar sayılarının toplamı 7 olduğu için a → *Esc*'nin olasılığı da 3/7 olarak belirlenmiştir. D yöntemi (PPMD) ise *Esc*'nin olasılığını, kontekstteki farklı karakter sayısını toplam karakter sayısının iki katına bölerek bulur. Bu yönteme göre hesaplansaydı a → *Esc*'nin olasılığı 3/8 olacaktı. Olasılık dağılımının düzgün olması için diğer karakterlerin tekrar sayıları da 2 katının 1 eksiği ile güncellenmelidir. Bu durumda a → b'nin tekrar sayısı 3, olasılığı da 3/8 olacak, a → c ve a → d ise 1/8 olasılığına sahip olacaktır.

2. derecede bulunamadığı için kaçış karakterinin olasılığı olan 1/2 kodlanır ve 1. dereceye inilir. 1. derecede bu defa *a* karakterini takip eden *a* aranır. Burada da bulunamayınca kaçış karakterinin olasılığı olan 3/7 kodlanır ve 0. dereceye inilerek *a* karakterinin kendisi aranır. Burada *a* karakteri vardır ve olasılığı 5/16'dır. O halde kodlanması gereken seri (1/2, 3/7, 5/16) şeklindedir.

2. derecede *ra*'dan sonra *c*'ye rastlanmadığı için kaçış karakteri kullanıldığına göre, 1. dereceye inildiğinde *a*'dan sonra *c* gelmesi olasılığı yoktur. O halde *a*'dan sonra *c* gelmesi olasılığı göz ardı edilerek tekrar sayısı 0 yapılırsa, *Esc*'nin olasılığı da 3/7

yerine 3/6 olarak kodlanabilir. Benzer şekilde kodlanacak karakter *b*, *c* veya *d*'den birisi olmadığı için 0. dereceye inildiğine göre, bu derecedeki *b*, *c* ve *d*'nin tekrar sayıları sıfırlanabilir. O halde 0. derecede a'nın olasılığı 5/12 olarak kabul edilebilir. Bu işleme çıkarma (exclusion) denir. PPMC çıkarma işlemini kullanarak sıkıştırma oranını arttırır. Bu durumda kodlanacak seri (1/2, 3/6, 5/12) şeklinde olacaktır. Bu seri için, yani *abracadabra*'dan sonra gelen *a* için kodlama masrafı aşağıdaki gibi hesaplanabilir:

$$-\log_2\left(\frac{1}{2}\times\frac{3}{6}\times\frac{5}{12}\right)=3.3bit$$

Çizelge 2.11'de en sağda yer alan -1 dereceli model, geleneksel olarak kullanılan *A* alfabesindeki tüm karakterlerin olasılıklarının eşit olduğu başlangıç modelidir. Eğer 0. derecede yer alan karakterler haricinde bir karakter, örneğin *e* karakteri kodlanmak istenseydi, *Esc*'nin olasılığı olan 5/12 kodlanıp -1. dereceye inilmesi gerecekti. Bu karakter 0. derecedeki 5 karakterden biri olmadığı için ASCII Tablosundaki diğer 251 karakterden biridir ve doğal olarak olasılığı da 1/251'dir. Bu durumda *abracadabra*'dan sonra gelen *e* için kodlama masrafı;

$$-\log_2\left(\frac{1}{2}\times\frac{3}{6}\times\frac{5}{12}\times\frac{1}{251}\right)=11.2bit$$ olacaktır.

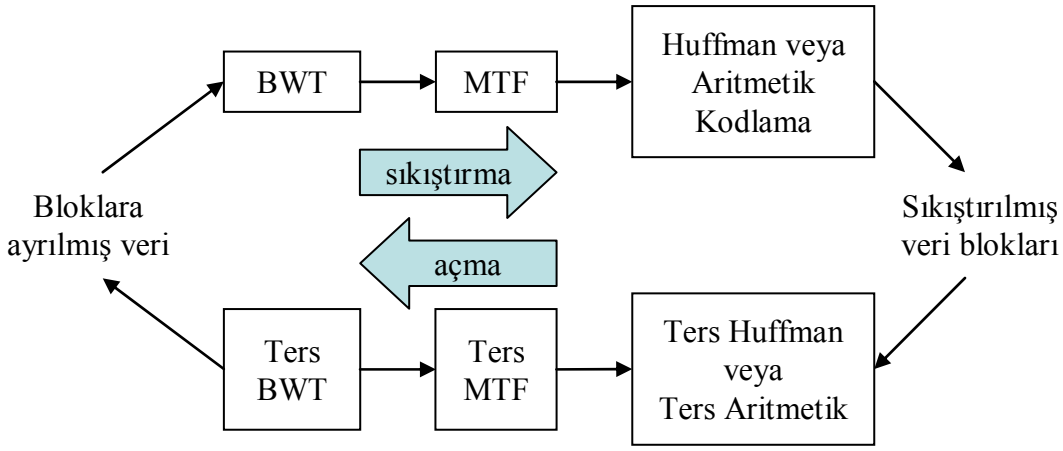
Kontekst büyüklüğü olarak ta adlandırılan *k* değeri 5'ten büyük olursa, kaçış karakterlerinin olasılıkları için yapılan masraf nedeniyle sıkıştırmada artış meydana gelmeyecek, hatta sıkıştırma süresi artacağı için elverişsiz bir durum ortaya çıkacaktır. Büyük kontekst büyüklüğü ile kullanıldığında çok iyi sıkıştırma oranlarına sahip olan PPM tabanlı sıkıştırma yöntemleri, sıkıştırma süresi açısından çok yavaş oldukları ve çok bellek kullandıkları için, çoğunlukla sadece saklama kapasitesini azaltmak amacıyla kullanılmaktadırlar. PPM, ağ transferlerini hızlandırma amacıyla kullanılmak istenirse, yavaşlığı nedeniyle bazı durumlarda transferi hızlandırmak yerine yavaşlatabilir.

2.5.4 BWCA (Burrows-Wheeler Compression Algorithm)

1994 yılında Michael Burrows ve David Wheeler tarafından yayınlanan "A Block-sorting Lossless Data Compression Algorithm" adlı makale ile tanımı verilen

algoritma, sonraları Burrows-Wheeler Sıkıştırma Algoritması (BWCA: Burrows-Wheeler Compression Algorithm) olarak anılmaya başlanmıştır. BWCA, Lempel-Ziv algoritmalarının hızına ve PPM algoritmalarının sıkıştırma oranına sahip olduğu için kısa sürede yaygınlaşmıştır.

BWCA'da yer alan işlemler Şekil 2.4'te gösterilmiştir. Algoritmanın temelini oluşturan, Burrows ve Wheeler'ın "blok sıralama" olarak adlandırdıkları işlem, günümüzde daha çok Burrows-Wheeler Dönüşümü (BWT: Burrows-Wheeler Transform) olarak bilinir. Benzer içeriklere sahip olan sembolleri bir araya getirip gruplayan BWT'den sonra MTF kodlaması gerçekleştirilir. BWT ve MTF sıkıştırma yapmaz sadece sonradan yapılacak olan sıkıştırma işleminin daha başarılı olması için veriyi uygun biçime getirir. Bu iki aşamayı asıl sıkıştırma işleminin gerçekleştiği Huffman veya aritmetik kodlama takip eder.

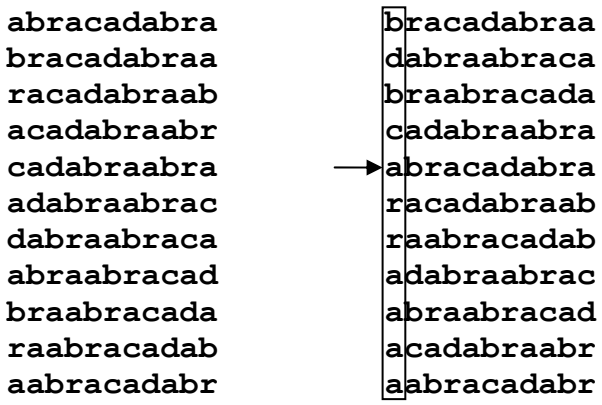


Şekil 2.4. BWCA Sıkıştırması

Sonradan geliştirilen başka BWT-tabanlı algoritmalar, MTF yerine sıkıştırma oranını arttıran daha farklı dönüşümler kullanmışlardır. Bu algoritmaların çoğu, Huffman veya aritmetik kodlama öncesinde RLE kodlamasını da kullanmışlardır. BWT ve ters BWT dönüşümleri Örnek 2.6 ile uygulamalı olarak açıklanmıştır.

Örnek 2.6: "abracadabra" kelimesini BWT ile daha iyi sıkıştırılabilir bir biçime getirelim. İlk işlem, bu kelimeyi bir matrisin ilk satırı gibi düşünüp, bu kelimeye dairesel sola kaydırma uygulayarak (en soldaki karakteri en sağa taşıyarak), oluşan yeni

kelimeyi matrisin bir alt satırına yazma ve bu işlemi en sondaki karakter en başa gelene kadar devam ettirme işlemidir. Böylece Şekil 2.5 (a)'da görüldüğü gibi 11×11 büyüklüğünde bir matris meydana gelecektir. İkinci işlem, bu matrisin satırlarını oluşturan kelimeleri, tersten okunuşlarına göre alfabetik sıralayarak yeni bir matris oluşturmaktır. Oluşturulan bu yeni matris Şekil 2.5 (b)'de görülmektedir. Sıkıştırılan "abracadabra" kelimesinin bu matrisin hangi satırında yer aldığının bilgisi olan 5 değeri ve ilk sütunu oluşturan karakterler (bdbcarraaaa) BWT'nin çıktısı olacaktır. Bu çıktı MTF kodlaması ile daha da sıkıştırılabilir bir biçime getirilir.



Şekil 2.5. BWT dönüşümünün iki aşaması: (a) matris oluşturma (b) tersten sıralama

Açma aşamasında kullanılan ters BWT, BWT'nin ürettiği ve ters MTF dönüşümünden de aynı şekilde çıkan "bdbcarraaaa" ifadesini, BWT'nin ürettiği diğer bir değer olan 5 değerini de kullanarak "abracadabra" ifadesini geri getirme işlemini yapar. Ters BWT'nin temel mantığı aşağıdaki kodda yer alan döngü ile gösterilmiştir. Bu döngüde **j** değeri ilk başta 5, **n** değeri ise girdinin uzunluğu olan 11'dir.

```
for i=1 to n
     Çıktı[i] = Girdi[j]
     j = SıraBul(Çıktı[i])
```

SıraBul fonksiyonu, parametre olarak aldığı karakterin **Girdi** dizisinde yer alan karakterler alfabetik sıraya göre dizildiğinde kaçıncı sırada yer aldığını döndürür. Çizelge 2.12'de **Girdi**, Sıralanmış **Girdi**, **SıraBul** çıktısı ve **Çıktı** değerleri yan

yana gösterilmiştir. Aynı değere sahip harflerin sırasının karışmaması için tüm harfler girdi dizisindeki sıraları ile numaralandırılmıştır. Döngüde **`j`** değerinin her adımda **`SıraBul`** fonksiyonunun sonucuna göre değişmesi ve **`Girdi`** dizisinde bir sonraki konumlanacak yerin belirlenmesi, döngü sırasına göre soldan sağa doğru yerleştirilmiş oklarla gösterilmiştir.

Çizelge 2.12. Ters BWT ile "bdbcarraaaa" ifadesini "abracadabra" ya dönüştürme

Girdi dizisi	Sıralanmış Girdi	SıraBul çıktısı	Çıktı dizisi
b_1	a_1	6	a_1
d_1	a_2	9	b_1
b_2	a_3	7	r_1
c_1	a_4	8	a_4
a_1	a_5	1	c_1
r_1	b_1	10	a_2
r_2	b_2	11	d_1
a_2	c_1	2	a_3
a_3	d_1	3	b_2
a_4	r_1	4	r_2
a_5	r_2	5	a_5

Sıkıştırılması istenen veri bloklara ayrılarak, her blok için ayrı BWT dönüşümü uygulanır. Pratikte blok boyu yukarıdaki örnekte olduğu gibi 11 byte değil, 100KB katları gibi çok daha büyük değerlere sahiptir. Patent koruması altında olmayan ücretsiz bir sıkıştırma aracı olan bzip2, BWT ve Huffman kodlamasını kullanarak dosyaları sıkıştırır. Sıkıştırma oranı geleneksel LZ tabanlı sıkıştırma araçlarından çok daha iyidir. PPM ailesi istatistiksel sıkıştırıcıların performansına oldukça yakındır.

3. SÖZLÜK TABANLI TEKNİKLER

Bir metinde sıkça tekrar eden kelimeler, bir görüntü dosyasında tekrar eden piksel grupları gibi, yinelenen kalıpların belirlenmesi ve bu kalıplardan bir sözlük oluşturularak, her kalıbın sözlükteki sıra numarasının kodlanmasına dayalı tekniklerdir.

Sözlük tabanlı teknikleri 3 temel kategoriye bölmek mümkündür:

- Statik Sözlük Yaklaşımı
- Yarı-Statik Sözlük Yaklaşımı
- Dinamik (Uyarlanır) Sözlük Yaklaşımı

Tek geçişli olan statik sözlük modelinde sıkıştırılacak olan tüm veriler aynı sözlük kullanılarak sıkıştırılır. Çift geçişli olan yarı-statik modelde ise, ilk geçişte sıkıştırılacak veride yer alan sembollerin dağılımları öğrenilir ve bu dağılama en uygun sözlük oluşturulur, ikinci geçişte ise bu sözlük kullanılarak sıkıştırma yapılır. En çok kullanılan yaklaşım olan dinamik sözlük modelinde ise, tek bir geçişte hem sözlük oluşturulur, hem de sıkıştırma yapılır. Dinamik sözlük tekniklerinin çoğu Jacob Ziv ve Abraham Lempel tarafından 1977 ve 1978 yıllarında yazılmış olan iki farklı makale üzerine geliştirilmişlerdir. 1977'deki makaleyi temel alan yaklaşımlara LZ77 ailesi, 1978'deki makaleyi temel alan yaklaşımlara ise LZ78 ailesi denir. LZ78 ailesinin en çok bilinen ve en iyi sıkıştırma oranı sağlayan üyesi 1984 yılında Terry Welch tarafından yayınlanan LZW algoritmasıdır.

Kaynak hakkında önceden önemli oranda bilgi varsa, sözlük oluşturma masrafından kurtulmak için statik sözlük kullanılabilir. Eğer kaynak hakkında bilgi yoksa, dinamik veya yarı-statik yaklaşımlardan birini kullanmak daha etkili olacaktır.

Bu bölümde öncelikle statik sözlük yaklaşımına değinilecek, daha sonra en çok bilinen statik sözlük yaklaşımı olan digram kodlaması ve tarafımızdan geliştirilen bu kodlamanın yarı-statik çalışmasını sağlayan SSDC ve ISSDC algoritmaları [Mesut ve Carus, 2004] anlatılacaktır. Dinamik sözlük yaklaşımından bahsedilirken, en çok kullanılan LZ ailesi algoritmaları anlatılacak, son olarak ta LZ77 ve Huffman

sıkıştırmalarını bir arada kullanan ve WinZip sıkıştırma aracında kullanılan Deflate algoritmasına değinilecektir.

3.1. Statik Sözlük Yaklaşımı

Sıkıştırılacak her verinin aynı sözlük ile sıkıştırılması statik sözlük yaklaşımıdır. Daha çok ASCII standardında kodlanmış metin tipinde verilerin sıkıştırılmasında kullanılır. Örneğin noktadan sonra boşluk (._) veya virgülden sonra boşluk (,_) gibi her tip metinde sıkça geçen karakter grupları, "_ve_", "_veya_" gibi kelimeler, ASCII Tablosunda genellikle kullanılmayan karakterlerin yerine yerleştirilebilir.

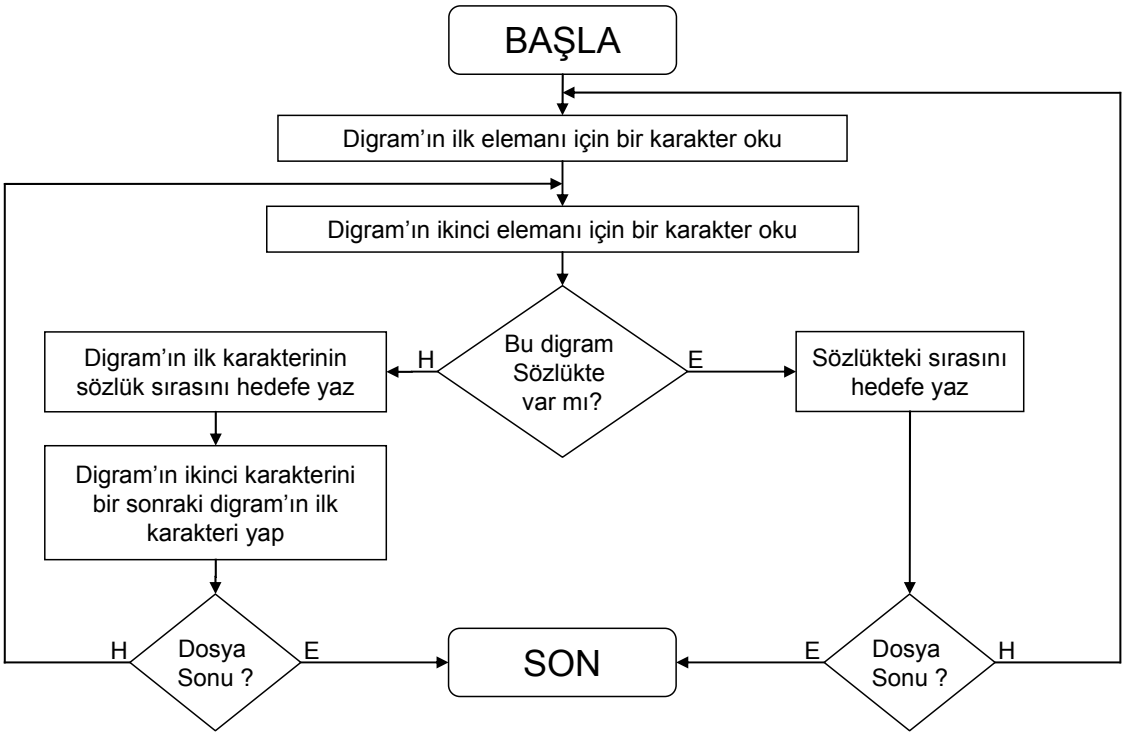
Statik sözlük hem sıkıştırma hem de açma algoritmalarında sabit olarak bulunacağı için, önceden bir veya daha fazla sayıda geçiş yaparak sözlük oluşturma işlemine gerek yoktur. Statik yaklaşımlar sıkıştırılacak veriye göre uyarlanabilir bir yapıda olmadıkları için dinamik yaklaşımlar kadar yüksek oranda sıkıştırma yapamasalar da, onlara göre çok daha hızlı çalışırlar (Sıkıştırma işlemi neredeyse açma işlemi kadar hızlıdır). Dinamik yaklaşımlarda ise sıkıştırma süresi genellikle açma süresinin birkaç katıdır.

3.1.1. Digram Kodlaması

Digram Kodlaması belirli bir kaynak tipine bağımlı olmayan bir statik sözlük tekniğidir. Bu kodlamada, kaynakta bulunabilecek tüm harflerle ve en sık kullanılan ikili karakter grupları (digram) ile oluşturulan bir statik sözlük kullanılır. Örneğin, kaynakta kullanılan dilin alfabesindeki tüm büyük harfler, küçük harfler, rakamlar ve noktalama işaretleri sözlüğün ilk kısmına ve istatistiksel bir analiz sonucunda bulunabilecek olan bu dildeki en sık tekrar edilen ikili karakter grupları sözlüğün ikinci kısmına yerleştirilebilir. Eğer ikili değil de üçlü karakter grupları ile sözlük oluşturulduysa trigram kodlaması, n'li karakter grupları ile oluşturulduysa n-gram kodlaması olarak adlandırılır.

Şekil 3.1'de akış şeması verilen Digram Kodlayıcısı şu şekilde çalışır; Kaynak dosyadan iki karakter okur ve bu karakterlerden bir digram oluşturur. Bu digram'ı sözlükte arar. Bulursa, sözlükteki sırasını hedef dosyaya yazar ve yeni bir digram

oluşturmak için iki karakter daha okur. Bulamazsa, ilk karakterinin sözlükteki sırasını hedef dosyaya yazar ve ikinci karakteri bir sonra aranacak digram'ın ilk karakteri yapar. Dosyadan bir karakter daha okuyarak digram'ı tamamlar. Döngü dosya sonuna kadar bu şekilde devam eder.



Şekil 3.1. Digram kodlayıcısının akış şeması

3.2. Yarı-Statik Sözlük Yaklaşımı

Önceki bölümde değindiğimiz gibi, statik sözlük yaklaşımında her metinde sıkça geçen karakter grupları veya kelimeler ASCII Tablosunda genellikle kullanılmayan karakterlerin yerine yerleştirilmektedir. Fakat bu yaklaşım, kullanılmadığını düşündüğümüz karakter eğer metinde kullanıldıysa hataya yol açacaktır. Bundan kaçınmak için;

- Sözlük büyüklüğü 256 karakter uzunluğundaki ASCII Tablosu ile sınırlı bırakılmayarak, 512 ya da 1024 karaktere kadar genişletilebilir.
- Hangi karakterlerin kullanılıp hangilerinin kullanılmadığı, sıkıştırılacak metin önceden bir defa okunarak tespit edilebilir (yarı-statik yaklaşım).

Bu işlemler bir miktar zaman kaybına neden olsa da, sıkıştırmanın performansına yapacağı olumlu etki nedeniyle, bu zaman kaybı göz ardı edilebilir.

3.2.1. SSDC (Semi-Static Digram Coding)

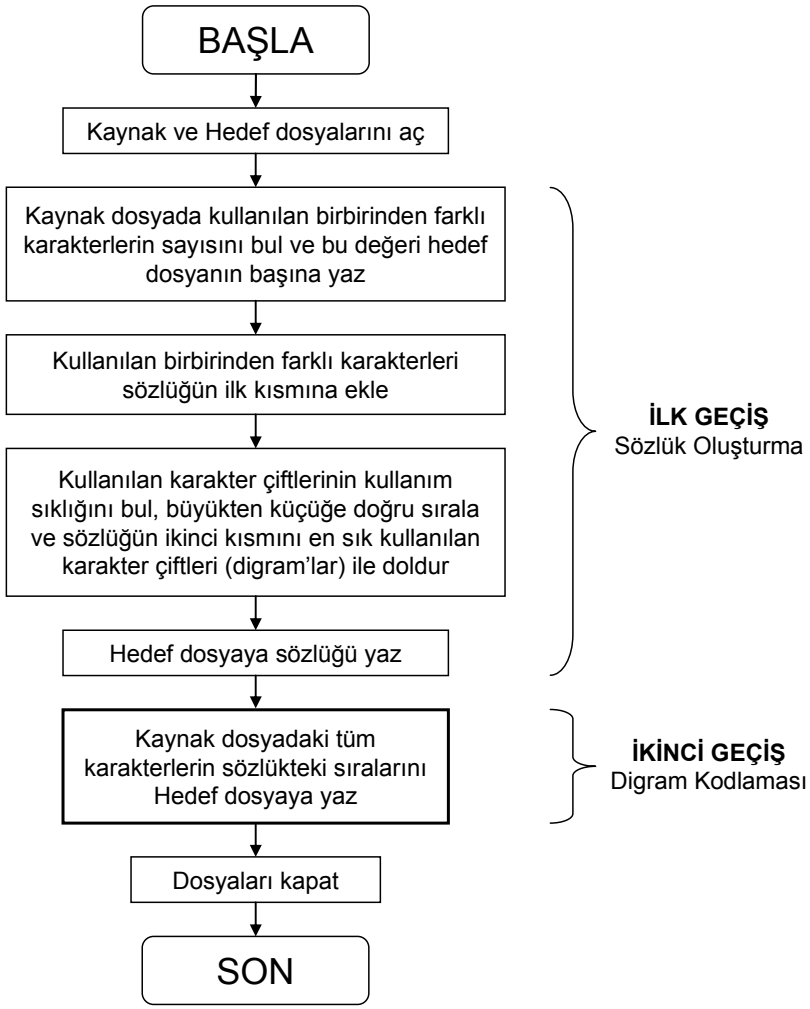
Geliştirdiğimiz ilk algoritma olan SSDC, digram kodlamasını temel alan, çift geçişli yarı-statik bir algoritmadır. Bu algoritma sözlüğü oluşturmak amacıyla bir ön geçiş yaptığı için, sıkıştırma zamanı tek geçişli statik sözlük yaklaşımına göre daha fazla olacaktır. Fakat kaynağa özel bir sözlük oluşturulduğu için, sıkıştırma oranı statik sözlük yaklaşımına göre daha iyi hale gelecektir.

Sıkıştırma Algoritması: SSDC algoritmasının ilk geçişinde, öncelikle kaynak dosyada kullanılan tüm farklı karakterler bulunur ve ASCII Tablosundaki sıralarına göre sözlüğün ilk kısmına eklenirler. Örneğin 'j' karakteri kaynakta hiç kullanılmamışsa, sözlükte yer almasına gerek olmadığı için, onun yerini bir sonraki karakter olan 'k' karakteri alır. ASCII Tablosunda sonraki karakterlerin hepsi de bir aşağı kaydırılır. İlk geçişte yapılan bir diğer işlem de, bütün karakter çiftlerinin (digram'ların) ve bu çiftlerin dosyada bulunma sıklıklarının kaydedilmesidir. Daha sonra, kaydedilen digram'lar tekrar sayılarına göre büyükten küçüğe doğru sıralanırlar. Eğer sözlük n farklı karakter içeriyorsa ve sözlük büyüklüğü d ise, sözlüğe eklenebilecek digram sayısı (sözlüğün ikinci kısmının büyüklüğü) d-n'dir. Bu nedenle, dosyada en sık tekrar eden digram'lar içinden ilk d-n tanesi sözlüğün ikinci kısmına eklenir. Dosyanın başına, sözlüğün kaç elemanının tek karakter ve kaç elemanının çift karakter (digram) olduğunun decoder tarafından bilinebilmesi için n değeri yazılmalıdır. 1 byte'lık bu değerin hemen ardından n adet tek karakterlik elemanlar, ve d-n adet çift karakterlik elemanlar yazılır. Dolayısıyla, dosyadaki sözlük masrafı;

$$1 + n + 2(d - n) = 2d - n + 1$$

olacaktır.

Algoritmanın ikinci geçişinde, Digram Kodlayıcısı ile ilk geçişte oluşturulan sözlük kullanılarak sıkıştırma işlemi gerçekleştirilir. Şekil 3.2'de SSDC sıkıştırma algoritmasının akış şeması verilmiştir.



Şekil 3.2. SSDC sıkıştırma algoritmasının akış şeması

Açma Algoritması: Sıkıştırılan dosyayı açma işlemi tek geçişlidir ve bu yüzden sıkıştırma işlemine göre daha hızlıdır. Statik sözlük yaklaşımından tek farkı, sözlüğün dosyanın içinde gömülü olmasıdır. Önce ilk karakter (*n* değeri) okunur. Daha sonra, bu karakterin sayısal değeri kadar karakter okunarak sözlüğün tek karakterlik elemanlardan oluşan ilk kısmı oluşturulur. En sonunda *d-n* defa iki karakter okunarak sözlüğün digram'ları kapsayan ikinci kısmı oluşturulur. Sıkıştırmanın ilk geçişinde oluşturulan sözlük açma işlemi için bu yolla tekrar elde edildikten sonra açma işlemi başlatılır. Açma işlemi, sıkıştırılmış dosyadan sırayla karakterlerin okunması ve sözlükteki karşılıkları ile değiştirilmesi ile gerçekleştirilir.

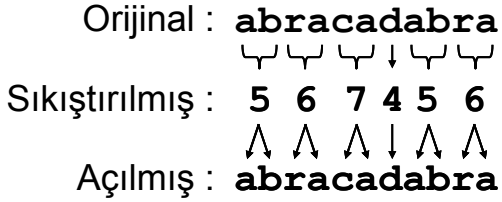
SSDC sıkıştırma ve açma algoritmalarının kodları C dili ile yazılmıştır. Sıkıştırma ve açma algoritmalarının performansı bu kodlar kullanılarak test edilmiş, ve

sonuçlar diğer kayıpsız veri sıkıştırma algoritmalarının performans sonuçları ile birlikte Bölüm 3.4'te verilmiştir.

3.2.2. ISSDC (Iterative Semi-Static Digram Coding)

SSDC tabanlı çalışan ISSDC'de, tekrarlanan bir yaklaşım kullanılarak sıkıştırma oranı arttırılmıştır. Bu çok geçişli algoritma, sözlüğün ikinci kısmı olan digram kısmını tek bir geçişte değil, kullanıcı tarafından da belirlenebilen tekrarlama sayısı kadar geçişte doldurur. Her tekrarlama, kendisinden önce gelen tekrarlamaların digram'lar ile doldurduğu sözlük kısımlarını da sözlüğün ilk kısmı gibi, yani tek karakterlik elemanlar gibi görür ve buna göre digram kodlaması kullanarak sıkıştırma yapar. Böylece, $2^{(\text{tekrarlama sayısı})}$ karakterlik sık tekrar edilen bir karakter grubu bile, hedef dosyada tek bir karakter ile temsil edilebilir. İki algoritma arasındaki farkın anlaşılabilmesi için, her iki yöntem ile de sıkıştırma ve açma adımlarını açıklayan bir örnek aşağıda verilmiştir.

Örnek 3.1: "abracadabra" kelimesinin SSDC algoritması ile sıkıştırılması ve açılması Şekil 3.3'te, ISSDC algoritması ile sıkıştırılması ve açılması da Şekil 3.4'te gösterilmiştir. Tekrar sayısı 1 olan digram'ların sözlüğe eklenmesi gerçekte faydasızdır. Örneğimizde kelimenin birçok defa tekrar ettiği düşünülerek sözlük oluşturulmuştur.

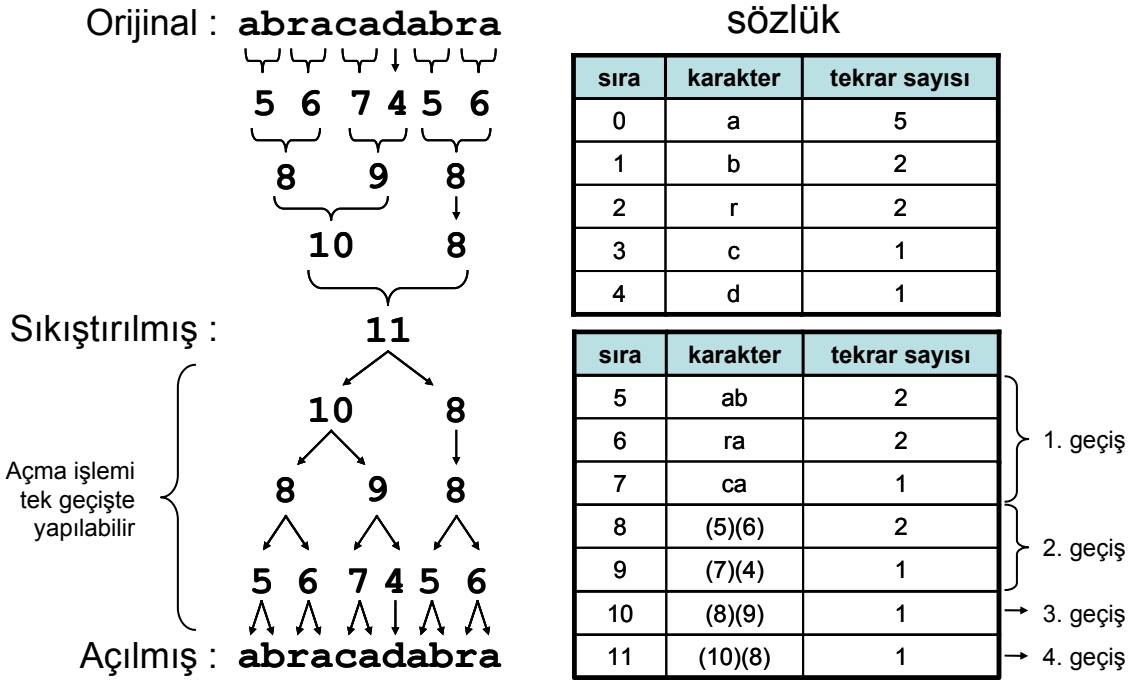


sözlük

sıra	karakter	tekrar sayısı
0	a	5
1	b	2
2	r	2
3	c	1
4	d	1

sıra	karakter	tekrar sayısı
5	ab	2
6	ra	2
7	ca	1

Şekil 3.3. SSDC algoritması ile sıkıştırma ve açma

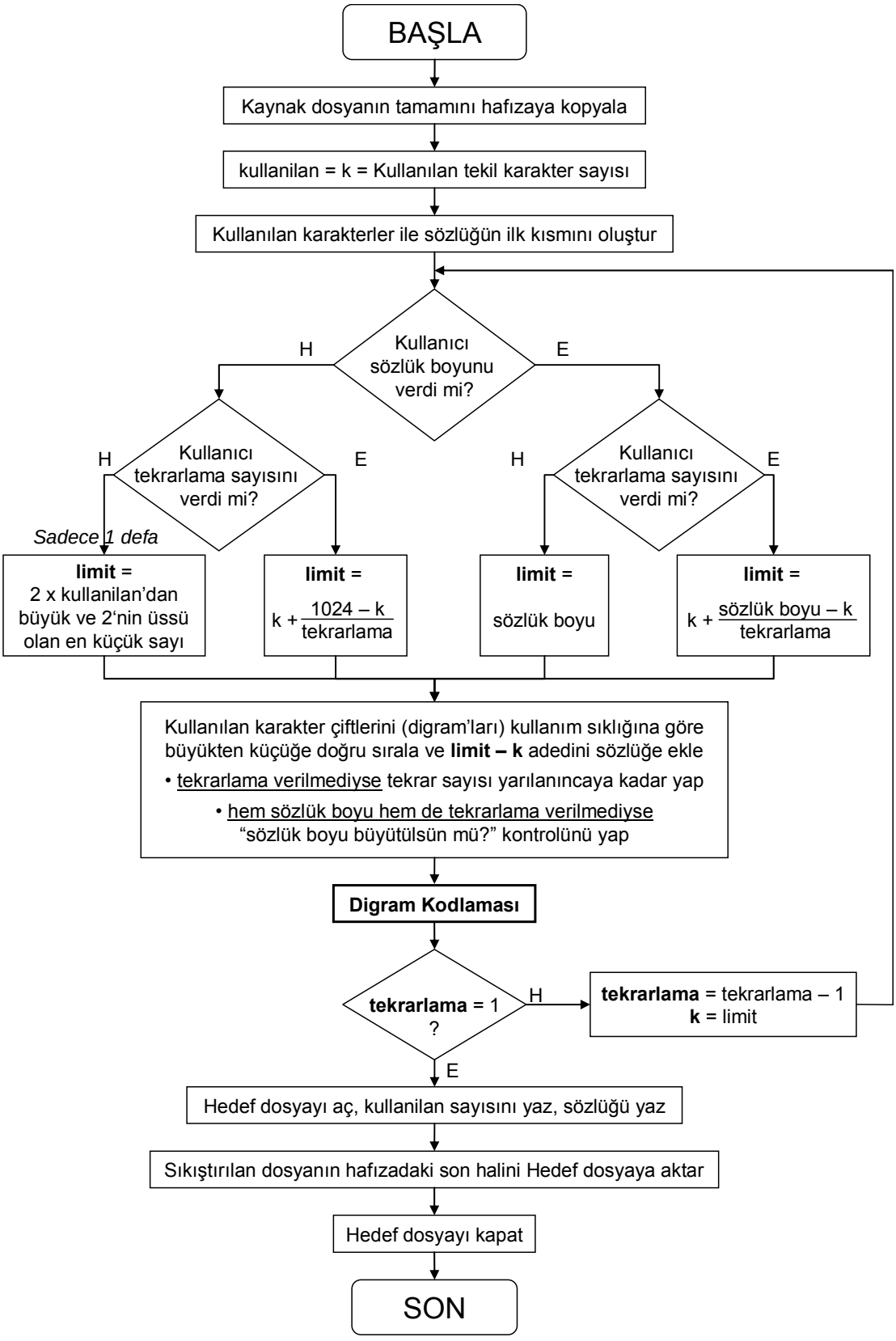


Şekil 3.4. ISSDC algoritması ile sıkıştırma ve açma

11 karakterlik "abracadabra" kelimesi, SSDC ile kodlananınca 6 karaktere kadar sıkıştırılabilirken, ISSDC ile 4 tekrarlamada 1 karaktere kadar sıkıştırılabilmektedir. Bununla beraber, her iki şekil incelendiğinde görülebileceği gibi, ISSDC kullanılınca bu kelime için sözlük masrafı daha fazla olmaktadır.

Sıkıştırma Algoritması: ISSDC sıkıştırma algoritmasının akış şeması Şekil 3.5'te verilmiştir. Şekilde görüldüğü gibi ISSDC, sadece SSDC'nin verilen sayıda tekrar eden bir hali değildir. ISSDC'de aynı zamanda sözlük büyüklüğüne ve tekrarlama sayısına otomatik olarak karar verebilen yapılar da mevcuttur.

ISSDC sıkıştırma algoritması kaynak verinin üzerinde çok geçiş yapacağı için dosya okuma/yazma zamanları sıkıştırma süresini arttıracağından, öncelikle kaynak dosya geçici bellek bölgesine taşınır. Bu taşıma işlemi sırasında kaynak dosyada kullanılan ve kullanılmayan karakterler de bulunur. SSDC'de olduğu gibi kullanılmayan karakterler elenerek, kullanılan karakterlerle sözlüğün ilk kısmı oluşturulur. Geçici bellekteki kaynak verinin sembolleri, yeni oluşturulan sözlükteki karşılıkları ile değiştirilirler.



Şekil 3.5. ISSDC sıkıştırma algoritmasının akış şeması

Eğer tekrarlama sayısı ve sözlük büyüklüğü verildiyse, otomatik olarak karar verilmesi gereken bir durum yoktur. Bu durumda, kaynakta kullanılan karakterler sözlüğe eklendikten sonra, sözlükte kalan boş yer tekrarlama sayısına bölünerek, yapılacak tekrarlama adımında sözlüğün ne kadar büyüklükte bir kısmının doldurulacağı belirlenir. Bu değer, kullanılan karakter sayısına (k) eklenerek bir limit değeri bulunur ve sözlüğün [k, limit] aralığı kaynakta en çok tekrarlanmış olan karakter çiftleri ile doldurulur. Karakter çiftleri, yani digram'lar, kullanım sayılarına göre büyükten küçüğe doğru sıralı şekilde sözlüğe eklenirler.

Her tekrarlama adımında sözlüğün hangi noktasına kadar en sık kullanılan karakter çiftlerinin yerleştirileceği (limit değeri) aşağıdaki formül ile bulunur:

$$\text{limit} = \text{k} + \frac{\text{sözlük boyu} - \text{k}}{\text{tekrarlama}}$$

Limit değerine kadar doldurulan sözlük ile kaynak veri SSDC benzeri bir yöntem ile sıkıştırıldıktan sonra, bir sonraki tekrarlama adımına geçmeden önce, k değeri limite eşitlenir ve tekrarlama değeri bir azaltılır. "k = limit" yapılması ile bir sonraki tekrarlama, bir öncekinde sözlüğe dâhil edilen karakter çiftlerini de kaynakta kullanılan tek karakterlik semboller gibi görecektir. Zaten bir önceki tekrarlamada sıkıştırma işlemi de gerçekleştirildiği için, kaynağın geçici bellekteki görüntüsünde aslında onlar tek bir karaktere indirgenmiş olacaklardır. Bu durumda sıkıştırma işlemindeki her tekrarlama, bir önceki tekrarlamadan bağımsız olarak çalışan bir SSDC sıkıştırması gibi düşünülebilir.

Eğer tekrarlama sayısı verilmiş, fakat sözlük büyüklüğü verilmemişse, sözlük büyüklüğü 1024 olarak kabul edilir ve yukarıda anlatılan işlemler aynı şekilde uygulanır. Ama sözlük büyüklüğü verilmiş, tekrarlama sayısı verilmemişse, en çok kullanılan digram'ın tekrar sayısının yarısı kadar tekrar etmiş digram'lara ulaşılıncaya kadar tekrarlama devam edecek ve sıralanmış digram'lar sözlüğe eklenecektir. Örneğin o tekrarlamada en çok tekrar etmiş digram'ın tekrar sayısı 100 ise, en son olarak 50 tekrar sayısına sahip olan digram'lar sözlüğe eklenecek, 49 tekrar sayısına sahip olanlar eklenmeyecektir.

Eğer hem tekrarlama sayısı hem de sözlük büyüklüğü verilmemiş ise, her iki büyüklüğe de algoritmanın karar vermesi gerekecektir. Bu durumda sözlük büyüklüğü, kullanılan karakter sayısının iki katından büyük olan ve 2'nin üssü olan en küçük tamsayı olarak belirlenecektir. Sözlük büyüklüğü belirlendikten sonra, sözlük büyüklüğü verilmiş fakat tekrarlama sayısı verilmemiş durumda olduğu gibi, en büyük tekrar sayısı yarılanıncaya kadar digram'ları sözlüğe ekleme yaklaşımı kullanılacaktır. Fakat bu defa, sözlük tamamen dolduğunda, sözlüğün boyunu iki katına çıkarmanın yarar sağlayıp sağlamayacağına bakılacaktır. Sözlüğün boyunu iki katına çıkarmak, karakter başına kullanılan bit sayısını 1 bit arttırmak anlamına geldiği için her zaman yarar sağlamayabilir. Örneğin, sözlük büyüklüğü 256'dan 512'ye çıkarmak, karakter başına 8 bit yerine 9 bit kullanılmasını beraberinde getireceği için, kaynağın büyüklüğü %12,5 artacaktır. Eğer sözlük büyüklüğü arttırıldıktan sonra gerçekleştirilecek olan sıkıştırma ile %12,5'lik bu kayıp karşılanamayacaksa, sözlüğü büyütmek yarar değil zarar getirecektir. Sözlüğü büyütüp sıkıştırmayı yapmadan, bu işlemin yararlı mı yoksa zararlı mı olacağını kestirebilmek güçtür. ISSDC, bu kestirimi yapabilmek için, son tekrarlamada en çok tekrar eden digram'ın tekrar sayısını temel alır. Bu sayıyı ikiye bölüp sözlük büyüklüğü ile çarparak bir eşik değeri belirler. Kaynağın büyüklüğünün %12,5'inin (1/8'inin) bu eşik değerinden küçük olması durumunda sözlüğü iki katına çıkarır. Eşik değerinin neden bu şekilde belirlendiğini bir örnek ile açıklayalım;

Son tekrarlamada en çok tekrar eden digram'ın tekrar sayısı 20 ise ve sözlük büyüklüğü 256 ise, sözlük büyüklüğü 512 yapıldığında, 256 digram için sözlük boşluğu yaratılmış olacaktır. Eklenecek 256 digram'ın ortalama tekrar sayısını 20'nin yarısı, yani 10 olarak düşünürsek, sözlük masrafını göz önünde bulundurmadığımız zaman 2560 karakter kazanç sağlanacağı düşünülebilir. O halde kaynağın büyüklüğünün %12,5'inin 2560'tan küçük olması, bit artımı sebebiyle oluşan büyüme miktarını sıkıştırmanın telafi edebileceğini gösterecektir. Ortalama kazancın beklenenin altında (10'dan düşük) çıkması durumunda, sözlük büyütmenin yarar yerine zarar vermesi olasılığı meydana gelecektir. Birçok test yapılarak belirlenmiş olan bu formülün her zaman doğru kestirim yaptığı kanıtlanamasa da, başarı oranının yüksek olduğu söylenebilir.

Şunu da belirtmek gerekir ki, sözlük büyüklüğü 512'den 1024'e çıkartılırsa, kaynağın büyüklüğünün sıkıştırma yapılmadan önce %12,5 değil, %11,1 artacağı, benzer şekilde 128'den 256'ya çıkartılan sözlük büyüklüğünün ise %14,3 oranında bir artış meydana getireceği gerçeği göz önünde bulundurularak, sözlük büyüklüğüne göre de katsayı belirlenebilir. Fakat hepsi için %12,5'in (1/8'in) kullanılması da, aslında kestirimin sonucunu çok fazla etkilememektedir.

Eğer son tekrarlamada en çok tekrar eden digram'ın tekrar sayısı çok küçükse, yukarıdaki ölçütü sağlasa bile, sözlüğün büyütülmesi digram'ların tekrar sayılarının sıfıra kadar düşmesine ve sonsuz döngülere neden olabileceği için, ek bir ölçüt olarak bu değerin 8'den küçük olması durumunda sözlüğün büyütülmesine izin verilmez.

Algoritmanın sıkıştırma oranını arttırmak için sonradan eklenen küçük bir kontrol ile sözlükteki gereksiz elemanların bir kısmının atılması sağlanmıştır. Buna göre, bir digram'ın ilk karakteri daha önce sözlüğe eklenen digram'ların herhangi birinin ikinci karakteri ile aynı ise, ya da ikinci karakteri diğer digram'ların herhangi birinin ilk karakteri ile aynı ise, bu digram sözlüğe eklenmez. Örneğin "ler_" tümcesi "le" ve "r_" şeklinde sıkıştırılıyorsa, ortada yer alan "er" tümcesi sözlüğe eklenmez. Bu işlem aynı tekrarlama içerisinde halledilebildiği için gerçekleştirilmesi kolaydır. Fakat, bir sonraki tekrarlamanın bir önceki tekrarlamanın elemanlarını kapsadığı durumda, önceki tekrarlamanın ilgili elemanları gereksiz hale gelmesi gibi bir durum için daha karmaşık bir geliştirme yapmak gereklidir. Sözlüğün digram tabanlı çalışması durumunda, eski elemanları sözlükten çıkarmak zor olacağı için, sözlüğün saklama biçimi değiştirilerek daha etkin sıkıştırma sağlanabileceği düşünülmüştür. Bu doğrultuda yapılan çalışılmalar neticesinde sıkıştırma oranı bir miktar arttırılabilmiş, fakat fazla sayıda kontrolün yapılması nedeniyle sıkıştırma işlemi oldukça yavaşlamıştır.

Tüm tekrarlamalar bitirildikten sonra kaynak dosyanın sıkıştırılmış hali olan hedef dosyayı oluşturma işlemi gerçekleştirilir. Hedef dosyanın ilk karakteri kullanılan sözlük büyüklüğüdür. ISSDC mevcut hali ile 5 farklı sözlük boyutu kullanır. Bunlar: 64, 128, 256, 512 ve 1024'tür. Eğer 64 veya 128 kullanıldıysa, direkt olarak sözlük büyüklüğü değeri yazılır. Fakat 256, 512 veya 1024 kullanıldıysa, bu değerler 8-bit ile

temsil edilemeyeceği için 256'ya bölünerek çıkan sonuç (1, 2 veya 4) yazılır. Hedef dosyanın ikinci karakteri ise orijinal kaynakta kullanılan birbirinden farklı karakterlerin sayısıdır. Yani ilk tekrarlamadan önceki 'k' sayısıdır. İlk iki karakterden sonra sözlük dosyaya eklenir. Sözlüğün boyu 512 veya 1024 ise, ilk 256 karakter 8'er bit olarak *fputc* fonksiyonu kullanılarak, diğer karakterler ise 9 veya 10 bit olarak yazılmaları gerektiği için özel olarak geliştirilen *dosyaya_yaz* fonksiyonu kullanılarak yazılırlar. Sözlüğün eklenmesinin ardından, geçici bellekte yer alan kaynak dosyanın sıkıştırılmış hali hedef dosyaya aktarılır. Bu aktarım işlemi de sözlük büyüklüğüne göre üç farklı şekilde gerçeklenebilir. Eğer sözlük büyüklüğü 256 ise *fputc*, 512 veya 1024 ise *dosyaya_yaz*, 64 veya 128 ise *dosyaya_yaz2* kullanılır.

ISSDC sıkıştırmasını gerçekleştiren C dili ile yazılmış olan program kodu, tekrarlama sayısını ve sözlük büyüklüğünü parametre olarak alır. Eğer bu parametreler verilmediyse algoritma otomatik olarak karar verme kipinde çalışır. Sıkıştırma algoritmasının işleyişinin daha iyi anlaşılabilmesi için aşağıda bir örnek verilmiştir.

Örnek 3.2: 47 byte uzunluğunda "altan altan altan altan altan altan altan altan" girdisi için gereksiz digram'lar atılmadan ilk tekrarlamada oluşturulan sözlük Çizelge 3.1'deki gibi olacaktır:

Çizelge 3.1. ISSDC ile gereksiz digram'lar atılmadan oluşturulan sözlük

Kod	Temsil ettiği	Tekrar sayısı	Kod	Temsil ettiği	Tekrar sayısı	Kod	Temsil ettiği	Tekrar sayısı
0	a	16	4	_ [boşluk]	8	8	03 (an)	8
1	l	8	5	01 (al)	8	9	34 (n_)	7
2	t	8	6	12 (lt)	8	10	40 (_a)	7
3	n	8	7	20 (ta)	8			

Çizelge 3.1'de görülen sözlük ile sıkıştırma yapılırsa, girdi olarak alınan 47 byte'lık veri "57957957957957957957957 3" halini alarak 24 byte'a düşürülmüş olur. Fakat 5 adet tek karakter ve 6 adet iki karakterlik digram'dan oluşan sözlüğün de saklanması gerekir. Sözlük, içerdiği tüm karakterlerin arka arkaya sıralanması ile "altn allttaann a" şeklinde saklanır. Kod çözücünün (decoder) sözlükte yer alan karakterlerin kaç tanesinin tek karakter olduğunu bilmesi gerekir. Ancak bu sayede bitişik karakterler

topluluğu gibi görünen bu sözlüğü ayrıştırabilir. Sözlüğün başına bir karakter de (5 karakteri) be nedenle eklenir. Kod çözücü 5 karakterini görünce ilk olarak 5 adet tek karakter okuyacak, geri kalanları da ikişer ikişer okuyarak digram'ları oluşturup sözlüğü yaratacaktır. Dolayısıyla sözlük "5altn allttaann a" şeklinde 18 karakterden oluşacaktır.

Gereksiz yere sözlükte yer kaplayan digram'ların ISSDC algoritması tarafından sıkıştırma yapılmadan önce sözlükten atıldığını söylemiştik. Buna göre ilk digram olan *al* sözlüğe yerleştikten sonra *lt* digram'ının ilk karakteri olan *l* harfi, *al* digram'ının ikinci karakterine eşit olduğu için *lt* sözlükten atılır. Tersi bir durum olarak, *ta* digram'ının ikinci karakteri *al* digram'ının ilk karakteri ile aynı olduğu için *ta* digramı da sözlükte yer almayacaktır. *an* digram'ı bu koşullardan birisi ile uyuşmadığı için sözlükte kalır. 9 ve 10 kodlu *n_* ve *_a* digram'ları da aynı nedenle sözlükten atıldıktan sonra 6 digram'dan geriye sadece iki digram kalır. Gereksiz digram'ların atılması ile oluşan yeni sözlük Çizelge 3.2'de gösterilmiştir. Bu sözlük "5altn alan" şeklinde saklanacağı için 18 byte yerine 10 byte yer kaplayacaktır.

Çizelge 3.2. ISSDC ile ilk tekrarlamada oluşturulan sözlük

Kod	Temsil ettiği	Tekrar sayısı
0	a	16
1	l	8
2	t	8
3	n	8

Kod	Temsil ettiği	Tekrar sayısı
4	_ (boşluk)	8
5	01 (al)	8
6	03 (an)	8

Çizelge 3.2'de görülen sözlük ile sıkıştırma yapılırsa, girdi olarak alınan 47 byte'lık veri "52645264526452645264526452645264526" halini alır. 31 byte'a düşen bu veri ile 10 byte uzunluğunda sözlük birleştirilerek elde edilen, kaynak dosyanın sıkıştırılmış hali olan 41 byte büyüklüğünde hedef dosya aşağıda verilmiştir. Altı çizili kısım sözlüktür.

5altn alan52645264526452645264526452645264526

İkinci tekrarlamada sözlüğe eklenen yeni digram'lar Çizelge 3.3'te gösterilmiştir. 7 koduna sahip yeni eklenen digram, ilk tekrarlamada üretilen 5 kodlu *al*

digram'ının 2 kodlu *t* ile birleştirilmesiyle oluşur. Benzer şekilde 8 kodlu digram da 6 kodlu *an* ve 4 kodlu boşluk karakterlerinin birleştirilmesi ile oluşur.

Çizelge 3.3. ISSDC ile ikinci tekrarlamada oluşturulan sözlük

Kod	Temsil ettiği	Tekrar sayısı
0	a	16
1	l	8
2	t	8
3	n	8

Kod	Temsil ettiği	Tekrar sayısı
4	_ (boşluk)	8
5	01 (al)	8
6	03 (an)	8
7	52 (alt)	8

Kod	Temsil ettiği	Tekrar sayısı
8	64 (an_)	7

İkinci sıkıştırma sonrasında sıkıştırılan veri 16 byte'a düşecek ve 14 byte büyüklüğündeki sözlük ile birlikte hedef dosyaya yazılan toplam 30 byte'lık veri aşağıdaki gibi olacaktır:

5altn alan526478787878787876

Üçüncü tekrarlamada sözlüğe eklenen yeni digram'lar Çizelge 3.4'te gösterilmiştir. 9 koduna sahip yeni eklenen digram, bir önceki tekrarlamada üretilen 7 kodlu *alt* digram'ının 8 kodlu *an_* ile birleştirilmesiyle oluşur.

Çizelge 3.4. ISSDC ile üçüncü tekrarlamada oluşturulan sözlük

Kod	Temsil ettiği	Tekrar sayısı
0	a	16
1	l	8
2	t	8
3	n	8

Kod	Temsil ettiği	Tekrar sayısı
4	_ [boşluk]	8
5	01 (al)	8
6	03 (an)	8
7	52 (alt)	8

Kod	Temsil ettiği	Tekrar sayısı
8	64 (an_)	7
9	78 (altan_)	7

Üçüncü sıkıştırma sonrasında sıkıştırılan veri 9 byte'a düşecek ve 16 byte büyüklüğündeki sözlük ile birlikte hedef dosyaya yazılan toplam 25 byte'lık veri aşağıdaki gibi olacaktır:

5altn alan526478999999976

Açma Algoritması: ISSDC'nin açma algoritması da SSDC'nin açma algoritması gibi tek geçişlidir ve neredeyse onun kadar hızlı çalışır. Fakat sıkıştırılmış karakterlerin sözlüğü kullanarak açılması işlemi gerçekleştirilirken ISSDC'de öz yinelemeli bir yapı kullanılır. Yukarıdaki örneği düşünürsek, 9 koduna sahip karakterin "altan_" karakter serisini ifade ettiğini ancak 3 tekrarlama sonunda bulabiliriz. İlk önce 9'u 78'e, sonra 78'i 5264'e ve en son olarak ta 5264'ü "altan_" serisinin karşılığı olan 012034'e açmamız gerekir. Bu işlemi öz yinelemeli olarak gerçekleştiren fonksiyonun kaba kodu aşağıda verilmiştir:

```
Açma (integer Kaynak, dosya Hedef){
    if (source < n){          - Kaynak bir karakterdir -
        Kaynak değerinin sözlük karşılığını Hedef dosyasına yaz
    } else {                  - Kaynak bir digram'dır -
        Açma (Kaynak digram'ının ilk karakteri, Hedef);
        Açma (Kaynak digram'ının ikinci karakteri, Hedef);
    }
}
```

Sözlüğün aktarımı tamamlandıktan sonra, dosyadan karakterler okunurken sözlüğün büyüklüğüne göre 3 farklı fonksiyondan biri kullanılır. Eğer sıkıştırılmış dosya 256 büyüklüğünde sözlük kullanıyorsa *fgetc*, 512 veya 1024 büyüklüğünde sözlük kullanıyorsa *dosyadan_oku*, 64 veya 128 büyüklüğünde sözlük kullanıyorsa *dosyaya_oku2* kullanılır.

SSDC ve ISSDC algoritmalarına aşağıdaki geliştirmeler yapılabilir:

- Daha etkili arama ve sıralama algoritmaları kullanılarak sıkıştırma hızı arttırılabilir.
- Digram yerine trigram veya tetragram kullanılarak sıkıştırma oranı arttırılabilir. Fakat bu durumda çok bellek kullanılması gerekecek ve sıkıştırma hızı yavaşlayacaktır.
- Sözlüğün bir kısmı statik hale getirilebilir. Sıkıştırılacak dosyada kaç adet farklı karakter kullanıldığı bulunduktan ve sözlüğe eklendikten sonra, sözlüğün belli bir yerine kadar dosyanın içeriğine (metin ise yazıldığı dile) uygun olan sık kullanılan ikili ya da çoklu tümceler eklenerek sözlüğün ilk

kısmı oluşturulabilir. Daha sonra sözlüğün boş kalan ikinci kısmı SSDC ya da ISSDC ile doldurulabilir. Statik kısım sıkıştırılacak dosya ile uyumlu değilse sıkıştırma oranı azalacaktır, fakat daha az digram ekleneceği için sözlük oluşturma işlemi daha çabuk bitecektir.

- ISSDC algoritmasını kullanan Windows tabanlı bir sıkıştırma yazılımı geliştirilebilir. Bu yazılımda daha kolay kullanım için tekrarlama sayısı ve sözlük büyüklüğü yerine Minimum, Normal, Maksimum gibi sıkıştırma profilleri tanımlanabilir. Bu profiller ya dosyanın tipine ve büyüklüğüne göre sözlük büyüklüğü ve tekrarlama parametrelerini farklı oranlarda kullanabilir, ya da her iki parametrenin de otomatik olarak belirlenmesi prosedürleri her profile göre farklı tanımlanabilir.

3.3 Dinamik (Uyarlanır) Sözlük Yaklaşımı

Dinamik sözlük yaklaşımı, statik sözlük yaklaşımı ve yarı-statik sözlük yaklaşımının avantajlı yönlerini bir araya getirmiştir. Statik sözlük yaklaşımı gibi tek geçişlidir dolayısıyla hızlıdır ve yarı-statik sözlük yaklaşımı gibi kaynağa özel sözlük üretir dolayısıyla sıkıştırma oranı yüksektir. En çok kullanılan dinamik sözlük yaklaşımları LZ77, LZ78 ve onun türevi olan LZW'dir.

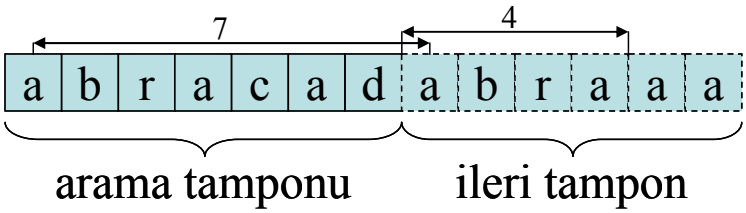
3.3.1. LZ77

Abraham Lempel ve Jakob Ziv tarafından geliştirilen ve 1977 yılında yayınladıkları "A Universal Algorithm for Data Compression" isimli makalelerinde tanımladıkları bu yöntem, o yıllarda tüm dünyada büyük ilgi görmüştür. Algoritmanın eksik yönleri zaman içinde farklı bilim adamları tarafından geliştirilmiştir. Sonraları yeni geliştirilen algoritmaların hepsine LZ77 ya da LZ1 ailesi denilmiştir.

LZ77 ailesi metin tabanlı veri sıkıştırmada büyük aşama kaydedilmesinin yolunu açmış, PKZip, Zip, Lharc (LHA) ve ARJ gibi 80'li ve 90'lı yılların popüler sıkıştırma paketleri değişken uzunluklu kodlayıcı ile desteklenen LZ77 tabanlı algoritmalar kullanmışlardır.

Sıkıştırma Algoritması: LZ77 yaklaşımında sözlük, daha önce kodlanmış serinin bir parçasıdır. Algoritmadaki arama tamponunun büyüklüğü, daha önce kodlanmış serinin ne büyüklükte bir parçasında arama yapılacağını belirler. Arama tamponu büyütüldükçe, sıkıştırma oranı artar, fakat aynı zamanda sıkıştırma zamanı da artar. Sıkıştırma algoritması aşağıdaki örnek ile açıklanacaktır:

Örnek 3.3: Örnek 3.1'de kullandığımız "abracadabra" kelimesini LZ77 algoritması ile sıkıştıralım.



Şekil 3.6. LZ77'de arama tamponu ve ileri tampon

İleri tamponun ilk karakteri olan *a*, arama tamponunda sondan başa doğru aranır. İkinci karşılaştırmada benzerlik bulunur, fakat bu karakterden sonra *b* karakteri değil de *d* karakteri yer aldığı için benzerlik uzunluğu sadece 1'dir. Arama devam ettirilir. İki karakter sonra bir *a* daha bulunur, sonrasında *c* yer aldığı için bunun da benzerlik uzunluğu 1'dir. Aramaya devam edilir. Arama tamponunun başında, yani ileri tamponda aranan karakterden 7 uzaklıkta (offset=7) bir *a* daha bulunur. Bu defa benzerlik uzunluğu 4'tür (abra). İleri tamponda "abra" serisinden sonra yer alan *a* karakteri ile birlikte [7,4,C(a)] şeklinde üçlü olarak kodlanır. İleri tamponun en sonundaki *a* karakteri ise [0,0,C(a)] şeklinde kodlanır.

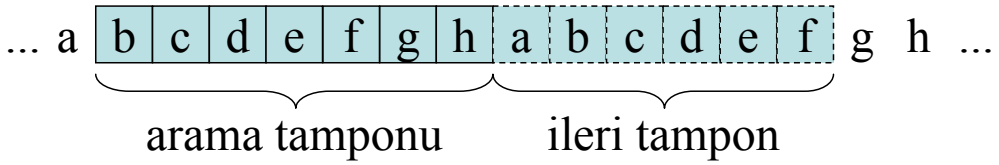
Açma Algoritması: Açma aşaması kodlanan tüm üçlü karakter gruplarının benzer şekilde açılması ile gerçekleştirilir. Mesela, Örnek 3.3'te yer alan üç karakterlik [7,4,C(a)] kodu, 7 karakter geri git, 4 karakter kopyala, sonuna *a* karakterini ekle şeklinde açılır.

Uygulanması kolay olan bu algoritmanın en büyük dezavantajı, eğer arama tamponunda aranan tekrarlar bulunamazsa kullanılan üçlü sistemin 1 byte'lık veriyi [0,0,C(a)] şeklinde temsil etmesi nedeniyle sıkıştırma yerine genişletme yapmasıdır.

LZ77 algoritması temel alınarak geliştirilen, LZSS olarak adlandırılan bir yaklaşım [Storer ve Szymanski, 1982], bu israfı tek bitlik bir bayrak kullanarak ortadan kaldırmıştır. Bu bayrak kendisinden sonra gelen verinin tek bir karakter mi yoksa bir karakter katarını ifade eden veri mi olduğunu belirler. Bu bayrak sayesinde üçüncü elemana da gerek kalmamıştır.

Eğer arama tamponu boyutu yeteri kadar büyük değilse tekrarlar bulunamaz. Örnek 3.3'te arama tamponunun boyutunu 6 olarak kabul edilseydi sıkıştırılacak benzerlik bulunamazdı. Öte yandan arama tamponu çok büyükse, arama zamanı ve dolayısıyla da sıkıştırma zamanı artar. Her ne kadar geliştirilen efektif arama yaklaşımları ile bu zaman bir ölçüde kısaltılabilmişse de, arama tamponu yine de çok büyük seçilmemelidir.

LZ77 yaklaşımının tıkandığı bir nokta vardır. Eğer periyodik bir dizimiz varsa ve periyodu arama tamponundan büyükse, hiç benzeşme bulunamaz ve her karakter için fazladan gönderdiğimiz veriler nedeniyle sıkıştırma yerine genişletme yapmış oluruz. Periyodik olarak tekrarlanan 8 karakter uzunluğundaki "abcdefgh" karakter grubu için benzeşme bulunamayacağı Şekil 3.7'de görülmektedir.



Şekil 3.7. LZ77'in tıkandığı periyodik tekrarlamaya örnek

3.3.2. LZ78

Lempel ve Ziv, LZ77'de yer alan tampon büyüklüğünün sınırlı olmasının yarattığı sıkıntıları ortadan kaldırmak için, tampon kullanmayan çok farklı bir yöntem geliştirerek 1978'in Eylül ayında yayınlanan "Compression of Individual Sequences via Variable-Rate Coding" isimli makalelerinde bu algoritmalarına. LZ77'den çok farklı bir yapıda olması nedeniyle, bu algoritma ve geliştirilmiş biçimleri, LZ78 (veya LZ2) ailesi olarak adlandırılmıştır. LZ77'den farklı olarak, sadece metin tabanlı sıkıştırmada değil, bitmap gibi farklı sayısal veriler üzerinde de başarıyla uygulanabilmiştir.

Sıkıştırma Algoritması: LZ78, hem kodlayıcı (encoder) hem de çözücü (decoder) tarafından aynı sözlüğün oluşturulması prensibine dayanır. Kaynaktan okunan sembol grupları sözlükte bulunan en uzun ön eklerinin indeksi ile birleştirilerek ikililer (pairs - [*i*,*c*]) ile kodlanır. Bu ikilideki *i*, yeni girişin sözlükte bulunan en uzun ön ekinin indeksi, *c* ise, bu ön eki takip eden karakterdir. Bu yöntemde kodlama yapılırken sözlüğün nasıl genişlediği aşağıda verilen örnek ile açıklanmıştır.

Örnek 3.4: Örnek 3.2'deki "altan altan altan altan altan altan altan altan" serisini kodlayalım. Başlangıçta boş olan sözlük sıkıştırma yapılırken doldurulur. Serinin tamamı kodlandığında Çizelge 3.5'te verilen 21 elemanlı bir sözlük oluşacaktır.

Çizelge 3.5. LZ78 ile kodlama ve oluşturulan sözlük

Sözlük sırası	Temsil ettiği	Kodu	Sözlük sırası	Temsil ettiği	Kodu	Sözlük sırası	Temsil ettiği	Kodu
1	a	[0,C(a)]	8	ta	[3,1]	15	alta	[10,1]
2	l	[0,C(l)]	9	n_	[5,6]	16	n_a	[9,1]
3	t	[0,C(t)]	10	alt	[7,3]	17	lta	[13,1]
4	a	[0,1]	11	an	[4,5]	18	n_al	[16,2]
5	n	[0,C(n)]	12	_a	[6,1]	19	tan	[8,5]
6	_	[0,C(_)]	13	lt	[2,3]	20	_al	[12,2]
7	al	[1,2]	14	an_	[11,6]	21	tan	[8,5]

Sıkıştırmanın nasıl yapıldığını ve aynı anda sözlüğün nasıl oluşturulduğunu açıklayalım. İlk karakter olan *a* karakteri okunur ve sözlüğe bakılır. Sözlük ilk etapta boş olduğu için bu karakterin sözlükte bir karşılığı yoktur, bu yüzden 1. sıraya yerleştirilir ve [0,C(a)] olarak kodlanır. Sözlüğün 2. ve 3. sırası da benzer şekilde oluşturulur. Serinin 4. karakteri olan *a* karakteri okunduktan sonra sözlüğe bakılınca ilk sırada olduğu görülür. Bir sonraki karakter olan *n* karakteri için tekrar sözlüğe bakılır, bulunamayınca birleştirme yapılamaz ve *a* karakteri [0,1] olarak, *n* karakteri de [0,C(n)] olarak kodlanır. 6. sıradaki boşluk karakteri de kodlandıktan sonra, hem 7. sıradaki a karakterinin, hem de 8. sıradaki n karakterinin sözlükte karşılığı olduğu bulunur. Bu iki karakter birleştirilerek [1,2] şeklinde kodlanır ve sözlüğün 7. sırasına yerleştirilir. Sözlüğün 8. ve 9. sırası da benzer şekilde oluşturulur. Serinin 13. karakterine gelindiğinde, öncelikle *a* karakteri sözlükte aranır. Bulunduktan sonra 14. karakter olan

l karakteri ile birleştirilerek oluşturulan *al* ikilisi de sözlükte aranır. Bu ikili de 7. sırada bulunduktan sonra 15. karakter olan *t* karakteri ile birleştirilerek *alt* üçlüsü oluşturulur. Bu üçlü sözlükte yer almamaktadır ama *t* karakteri sözlükte vardır. Bunun sonucunda *al* ikilisinin kodu olan 7 ile t karakterinin kodu olan 3 birleştirilerek [7,3] şeklinde kodlanır ve sözlüğün 10. sırasına yerleştirilir. Benzer şekilde kodlama devam ettirilirse sıkıştırma bitene kadar Çizelge 3.5'te görülen sözlük 21 elemana genişlemiş olacaktır.

Açma Algoritması: Açma algoritması, sıkıştırma algoritmasında oluşturulan sözlüğe ilk etapta sahip değildir. Bir yandan açma işlemi yapılırken diğer yandan sözlük oluşturulur. Okunan ikili kodun ilk karakteri 0 ise, bu ikilinin aslında tek karakteri temsil ettiği anlaşılır ve ikinci karakter okunarak kodlanır ve sözlüğe eklenir. İlk karakter sıfırdan farklı ise, bu kod iki ya da daha fazla karakterin birleşmesi ile oluşan bir grubu temsil ediyordur. Bu durumda, sözlüğe eklenmiş olan elemanlar içinde bu kodun iki elemanının da karşılıkları bulunur ve kodlanır.

LZ78 algoritması kalıpları bulma ve ayrı ayrı saklama becerisine sahip olsa da, en önemli dezavantajı Örnek 3.4'te görüldüğü gibi, sözlüğün sınırsız bir şekilde büyümesidir. Pratikte, sözlüğün büyümesi belirli bir noktada durdurulmalı, ya gereksiz girdiler elenmeli, ya da kodlama sabit sözlük şemasına zorlanmalıdır.

3.3.3. LZW

Terry Welch 1984'te Unisys (o zamanki adı Sperry Corporation idi) için çalışırken, LZ78 yaklaşımını yüksek performanslı disk ünitelerine uyarlamış ve ortaya çıkan yeni algoritma LZW olarak kabul görmüştür [Welch, 1984]. LZW hem sıkıştırma hem de açma performansı açısından LZ78 ailesinin en iyisi olmayı başarmıştır. Her tip veri üzerinde iyi sonuçlar veren bir algoritma olduğu için, kendisinden sonra gelen birçok algoritma LZW'yi temel almıştır. 1985 yılından beri Unisys LZW'nin patentini elinde bulundurmaktadır.

Sıkıştırma Algoritması: LZW algoritmasında, LZ78'de kullanılan ikili yapısındaki ikinci elemanın gerekliliği ortadan kalkmıştır. Kodlayıcı, önce kaynaktaki tüm karakterlerden bir sözlük oluşturarak gönderir. Bu karakterler bir ön geçişle bulunabilir. Eğer ASCII Tablosundaki tüm karakterler kullanılacaksa ön geçiş

yapılmasına ve sözlüğün gönderilmesine gerek yoktur. Kodlama aşamasında okunan her karakter sözlükte aranır. Bulunursa bir sonraki karakter de okunur ikisi birleştirilerek aranır. Sözlükte karşılığı bulunamayana kadar bu şekilde devam eder. Sözlükte karşılığı bulunmayan bir girdiye ulaşıldığında ise son karakteri hariç önceki karakterlerinin sözlükteki karşılığı kodlanır. Bu kod ile son karakterin sözlükteki kodu birleştirilerek sözlükte yeni bir girdi oluşturulur. Son karakter, sonraki adımda ilk karakter yapılarak kodlamaya devam edilir.

Örnek 3.5: Önceki örneklerde kullandığımız seriyi LZW ile kodlayalım. Başlangıçta sözlükte tüm ASCII karakterleri bulunacaktır. Çizelge 3.6'da sadeleştirme yapılarak sadece seride kullanılan karakterlere yer verilmiştir. Çizelgenin ilk 5 elemanı bu karakterlerden oluşmaktadır. 256 ve daha sonraki karakterler, LZW tarafından kodlama yapılırken sözlüğe eklenen karakterlerdir.

Çizelge 3.6. LZW ile oluşturulan sözlük

Sözlük sırası	Temsil ettiği	Sözlük sırası	Temsil ettiği	Sözlük sırası	Temsil ettiği	Sözlük sırası	Temsil ettiği
32	_	258	ta	265	alta	272	ltan
97	a	259	an	266	an_	273	n_alt
108	l	260	n_	267	_al	274	tan_
110	n	261	_a	268	lta	275	_alt
116	t	262	alt	269	an_a		
256	al	263	tan	270	altan		
257	lt	264	n_a	271	n_al		

Kodlama aşaması şu şekilde gerçekleşir: Önce ilk iki karakter *a* ve *l* karakterleri okunur. İki karakterin birleşimi olan *al* henüz sözlükte olmadığı için, [97,108] kodu ile sözlüğe 6. eleman olarak eklenir. Çizelge 3.6'da sözlükteki gerçek kod değerlerine değil, sadece temsil ettikleri karakter öbeklerine yer verilmiştir. İlk karakter sözlükteki karşılığı olan 97 ile kodlanarak ikinci karakter ilk karakter yapılır ve kodlamaya devam edilir. Sözlüğün 261'inci karakterine kadar kodlama benzer şekilde olacaktır. Serideki ikinci *altan* kodlanırken ilk *al* daha önce sözlüğe eklenmiş olduğu için bu defa *a* karakterinin kodu olan 97 değil, *al* ikilisinin kodu olan 256 kodlanır. Sözlüğe eklenirken ise bir sonraki karakter olan *t* ile birleştirilerek *alt* karakter öbeği oluşturulur ve 262'inci sözlük girdisi [256,116] kodu ile eklenir. Kaynak bitene kadar bu şekilde kodlanacak ve

sözlük büyümeye devam edecektir. Kaynak verinin LZW ile kodlanmış hali aşağıda verilmiştir:

97,108,116,97,110,32,256,258,260,262,259,261,257,266,
265,264,268,271,263,267,263

Serinin tamamı kodlandığında Çizelge 3.6'da verilen 275 elemanlı sözlük oluşacaktır. Bu sözlük sıkıştırma ve açma aşamalarında bellekte oluşturulur. ISSDC'de olduğu gibi asıl veriden önce gönderilen ve dosya içinde saklanan bir sözlük değildir. Bu nedenle LZW'de aslında sözlük masrafı yoktur.

Açma Algoritması: LZ78'de olduğu gibi LZW'de de açma algoritması, sıkıştırma algoritmasında oluşturulan sözlüğe ilk etapta sahip değildir. Sözlük sıkıştırma algoritmasında yaratıldığı gibi açma algoritmasında da yaratılır. Örnek 3.5'te üretilen çıktıyı düşünelim. İlk altı karakter (97,108,116,97,110,32) *altan_* karakter öbeğinin ASCII kodlarıdır. İlk adımda 97 ve 108 bir araya getirilip sözlüğün 256'ıncı elemanı, yani *al* oluşturulur. Birer karakter kaydırma ile sırasıyla [108,116] ile 257, [116,97] ile 258, [97,110] ile 259, [110,32] ile 260 ve [32,97] ile 261'inci sözlük girdileri oluşturulur. Kod çözücü kodlanan verinin yedinci karakteri olan 256'ya ulaştığında, artık bu kodun karşılığını sözlükten bulabilir. Bir yandan açma işlemi sürerken, diğer yandan sözlük 275'inci girdiye kadar benzer şekilde oluşturulur.

LZ ailesinin en çok kullanılan üyesi LZW'dir. Hızı ve sıkıştırma performansı yüksektir. Fakat, LZ78'de olduğu gibi, LZW algoritmasında da sözlük büyüklüğünün sürekli artması sorun yaratmaktadır. Çözüm olarak birçok farklı yöntem geliştirilmiştir. LZW tabanlı çalışan bir kayıpsız görüntü sıkıştırma algoritması olan GIF, ve Unix'teki "Compress", sözlük büyüklüğü 2^b değerine eriştiğinde (b değeri önceden belirlenir), sözlüğü iki katına (2^{b+1}) çıkartır. Sözlük daha da büyüdüğünde, statik sözlük yaklaşımına döner. Eğer sıkıştırma oranı belli bir seviyenin altında kaldıysa sözlüğü boşaltarak, sözlük oluşturma işlemini yeniden başlatır. British Telecom'un geliştirdiği BTLZ standartında ise en az kullanılan girdi çıkartılarak sözlük küçültülür.

3.3.4. DEFLATE

Phil Katz tarafından 90'lı yılların ortalarında geliştirilmiş kayıpsız veri sıkıştırma formatıdır. Huffman Kodlaması ve LZSS algoritmasının bir bileşimidir. LZW'nin tersine, Deflate herhangi bir patent koruması altında değildir. Bu nedenle geniş bir alanda kullanılmaktadır. WinZip (Windows), gzip (Unix), jar (Java) dosyaları (jar aslında bir zip dosyasıdır) ve PNG kayıpsız görüntü sıkıştırma standardı Deflate algoritmasını kullanmaktadır [Deutsch, 1996].

Sıkıştırılacak olan veri birbirini takip eden bloklar kümesi olarak düşünülür. Her blok LZSS algoritması ve Huffman kodlamasının birlikte kullanılması ile sıkıştırılır. Her blok için oluşturulan Huffman ağacı bir önceki ve bir sonraki bloktan bağımsızdır. Sıkıştırılabilen blokların büyüklüğü değişkendir. Deflate algoritması Huffman ağacının etkili kodlama yapamayacak kadar büyüdüğünü gördüğünde, yeni bir Huffman ağacı oluşturmak için o bloğu sonlandırarak yeni bir blok başlatır. Sıkıştırılamayan blokların boyu ise 65.535 byte ile sınırlıdır.

Her blok, sıkıştırılmış verilerin bulunduğu kısım ve bu verileri temsil eden Huffman kod ağaçlarının saklandığı kısım olmak üzere iki kısımdan oluşur. Kullanılan Huffman ağaçları her bloğun başında sıkıştırılmış veriden önce yer alır, ve bu ağaçlar da Huffman kodlaması kullanılarak sıkıştırılır. LZSS algoritması 32.768 byte uzunluğunda bir arama tamponu kullanır, ve bu sınırlar içerisinde kalmak koşuluyla bir önceki blokta yer alan bir katarı referans olarak kullanabilir. LZSS algoritmasında tekrar eden karakter sayısı uzunluğu (ileri tampon) 258 ile sınırlandırılmıştır. Kodlama etkinliği açısından en az 3 karakter uzunluğunda tekrarlar göz önüne alınır. 3 ile 258 arasındaki 256 farklı karakter sayısı uzunluğu 1 byte ile temsil edilebilir. 32.768 byte büyüklüğünde olan arama tamponu 15 bit ile temsil edilirken, 1 bit ise sıkıştırılmamış veriyi temsil eden bayrak için kullanılacağından [bayrak, mesafe, tekrar uzunluğu] gösterimi toplam 3 byte (1+15+8 = 24 bit) ile yapılır.

Herhangi bir uygun tekrarlamanın bulunamadığı durumlarda kodlanan gerçek değerler karakter sayısı uzunluğu değeri ile aynı büyüklükte (0-255 arasında) olduğu için ikisi aynı Huffman ağacı ile kodlanırken, mesafe bilgisinin boyu daha büyük olduğu için bu bilgi farklı bir Huffman ağacı ile kodlanır.

Tekrar eden karakter katarları hash tabloları sayesinde hızlı bir şekilde bulunur. 3 karakter uzunluğundaki tüm giriş katarları hash tablosuna eklenir. Bir sonraki 3 byte için bir hash indeksi hesaplanır. Eğer bu indeks için hash zinciri boş değilse, eklenmek istenen katar zincirdeki tüm katarlar ile karşılaştırılır ve en uygun eşleşme seçilir. Hash zincirlerinde arama, en son kullanılan katardan başlanarak yapılır. Bu sayede küçük mesafeler önce ele alınmış olur ve Huffman kodlaması daha etkin çalışır.

Deflate ile sıkıştırılmış kodu açan algoritmaya *Inflate* adı verilmiştir. Inflate, Huffman ağaçlarının kodunu çözerken hız kazanmak için iki kademeli bir arama tablosu kullanır. İlk kademe tablosu kısa kodları kullanan semboller ile yaratılır. Sıkıştırılan veride çok fazla yer alan semboller kısa kodlar ile kodlandığı için, kodu çözerken çoğunlukla kısa koda sahip semboller ile karşılaşılacak ve çoğunlukla bu tablo kullanılacaktır. Eğer okunan veri için bu tabloda karşılık bulunamazsa ikinci kademe tablolarına bakılacaktır.

Eğer Huffman tablosunda en uzun kodu kullanan sembol 15 bit büyüklüğünde ise, 2^{15} = 32.768 elemanlı bir arama tablosu oluşturmak çok zaman alacaktır. Bunun yerine 8 bit ve daha kısa semboller için 256 elemanlı bir ilk kademe tablosu, ve daha uzun kodların ilk 8 bitleri haricinde geri kalan bitlerini saklamak üzere birkaç ikinci kademe tablosu yaratmak daha az zaman alacaktır. Arama tablosu yaratmanın amacı ise, her bit okununca Huffman tablosuna bakarak iki misli zaman kaybetmemektir. Arama tablosu kullanılınca, 8-bit birden okunup, hemen karşılığı tablodan bulunabilir.

3.4. Kayıpsız Sıkıştırma Algoritmalarının Karşılaştırılması

Bu bölümde, önceki bölümlerde bahsedilen kayıpsız sıkıştırma algoritmalarının sıkıştırma oranları ve sıkıştırma zamanları ölçülmüş, sonuçların değerlendirilmesi verilmiştir. Sıkıştırma algoritmalarının performansını değerlendirmek için en çok kullanılan külliyat olan ve toplam 3.141.622 byte büyüklüğündeki 14 dosyadan oluşan *Calgary Külliyatı* (*Calgary Corpus*) [Witten ve Bell] kullanılmıştır.

Testte, Intel Pentium4 1.7GHz işlemcisi ve 256MB RDRAM belleği olan, işletim sistemi olarak Windows XP Professional yüklenmiş olan bir bilgisayar kullanılmıştır. Testte kullanılan tüm algoritmaların C kodları Microsoft Visual C++ 6.0

derleyicisi ile derlenmiştir. Testte kullanılan kodlar, kodu yazan kişiler ve kodların yazıldığı tarihler Çizelge 3.7'de verilmiştir.

Çizelge 3.7. Testte kullanılan kodlar

Yöntem	Kod Dosyası	Kodu Yazan	Tarih
LZ77	prog1.c	Rich Geldreich, Jr.	1993
LZW	codlzw.c	David Bourgin	1995
Huffman	codhuff.c	David Bourgin	1995
Aritmetik	ari.cpp	Mark Nelson	1996
SSDC	ssdc.cpp	Altan Mesut	2004
ISSDC	issdc.cpp	Altan Mesut	2005

Çizelge 3.8'de gerçekleştirdiğimiz karşılaştırmanın sonuçları verilmiştir. Bu çizelgede sıkıştırma verimi karakter başına bit kullanımı (bit/kr) olarak ifade edilmiştir. SSDC ve ISSDC algoritmalarında farklı parametreler kullanıldığında oluşan farklı sonuçları görmek için, birçok sıkıştırma sonucu verilmiştir. Bu parametrelerden *s* parametresi sözlük büyüklüğünü, *i* parametresi ise tekrarlama sayısını ifade eder. Eğer parametrenin değeri sıfırsa, ISSDC o parametre için otomatik hesaplama yapar.

Çizelge 3.8'de görülebileceği gibi SSDC ve ISSDC'de sözlük büyüklüğü arttıkça sıkıştırma verimi de artmakta, fakat sıkıştırma süresi uzamaktadır. Tekrarlama sayısının artması ise belirli bir noktaya kadar sıkıştırma verimini arttırmakta, sıkıştırma süresini pek fazla etkilememektedir. 512 elemanlı sözlük için 15, 1024 elemanlı sözlük için ise 30 tekrarlama en iyi sıkıştırmayı vermektedir. 512 elemanlı sözlükte 20 tekrarlama 15'e göre daha kötü sonuç vermiştir.

Tüm sözlük-tabanlı sıkıştırma algoritmalarında, açma işlemi sıkıştırma işleminden daha hızlı gerçekleşir. ISSDC'de açma işlemi sıkıştırmadaki tekrarlama sayısından bağımsız olduğu için (her zaman bir geçişli), açma süreleri birbirine eşit çıkmıştır. Sözlük büyüklüğü 256 olarak sıkıştırıldığında, açarken *dosyadan_oku* fonksiyonu kullanılmadığı için, bu durumda iki misli daha hızlı açma yapılabilmektedir.

Çizelge 3.8. Calgary Külliyatı'nın sıkıştırılma sonuçları

	Sıkıştırılmış Boyut (byte)	Verim (bit/kr)	Sıkıştırma Zamanı	Açma Zamanı
Sıkıştırılmamış	3.141.622	8,00		
Huffman	1.764.418	4,49	0.64sn	0.45sn
Aritmetik	1.713.128	4,36	1.09sn	1.19sn
LZW	1.521.341	3,87	0.59sn	0.34sn
LZ77	1.347.216	3,43	0.88sn	0.13sn
SSDC, d=256	2.003.492	5,10	1.00sn	0.14sn
SSDC, d=512	1.936.050	4,93	1.58sn	0.28sn
ISSDC s=256, i=0	1.682.659	4,28	2.92sn	0.14sn
ISSDC, s=512, i=5	1.407.663	3,58	5.00sn	0.30sn
ISSDC, s=512, i=10	1.388.119	3,53	5.17sn	0.30sn
ISSDC, s=512, i=15	1.387.250	3,53	5.47sn	0.30sn
ISSDC, s=512, i=20	1.396.380	3,56	5.78sn	0.30sn
ISSDC s=512, i=0	1.399.181	3,56	5.39sn	0.30sn
ISSDC s=1024, i=5	1.314.748	3,35	24.48sn	0.30sn
ISSDC s=1024, i=10	1.278.164	3,25	25.53sn	0.30sn
ISSDC s=1024, i=15	1.268.431	3,23	24.77sn	0.30sn
ISSDC s=1024, i=20	1.265.297	3,22	24.36sn	0.30sn
ISSDC s=1024, i=25	1.264.712	3,22	24.58sn	0.30sn
ISSDC s=1024, i=30	1.264.166	3,22	24.72sn	0.30sn
ISSDC s=1024, i=0	1.279.234	3,26	26.34sn	0.30sn
ISSDC s=0, i=0	1.278.014	3,25	25.06sn	0.30sn

ISSDC, beklenildiği gibi SSDC'den daha iyi bir sıkıştırma performansı göstermiş, fakat çok geçiş yaptığı için sıkıştırma süresi daha fazla olmuştur. Sıkıştırma süresini arttıran bir diğer etmen de sözlük büyüklüğüdür. Sözlük büyüklüğü 1024 olarak seçildiğinde, 512 büyüklüğündeki sözlüğe göre %10'a kadar daha iyi sıkıştırma oranı elde edilebilmiş, fakat sıkıştırma süresi 4.5 kat artmıştır.

SSDC, iki karakteri tek karaktere indirebildiği için en fazla %50 oranında bir sıkıştırma sağlayabilir. Örneğin Calgary Külliyatı'ndaki *pic* dosyası arka planı büyük oranda tek bir renk olan bir görüntü dosyasıdır. ISSDC tarafından %90 gibi büyük bir oranda sıkıştırılabilen bu dosya, SSDC tarafından sadece %43 oranında sıkıştırılabilmiştir. Eğer digram yerine trigram veya tetragram kullanılırsa, SSDC'nin performansı da yüksek olabilir.

Calgary Külliyatı'ndaki *geo*, *obj1* ve *obj2* dosyalarında 256 farklı ASCII karakterinin hepsi kullanılmaktadır. Bu dosyalar 256 sözlük boyutu ile sıkıştırıldığında digram'lar için sözlükte boş yer bulunamadığından dolayı sıkıştırma yerine şişme meydana gelmiştir.

PPM, BWCA veya Deflate gibi birden fazla sıkıştırma yaklaşımını bir arada kullanan bir yöntem ile sıkıştırma yapılırsa, sıkıştırma oranı büyük oranda artacaktır. Bu nedenle günümüzün popüler sıkıştırma araçları bu tip yöntemleri tercih etmektedirler. WinZip'in önceki sürümleri sadece Deflate yöntemi ile sıkıştırma yaparken, 10.0 sürümü, PPMd ve bzip2 (BWT tabanlı) yöntemlerini kullanarak ta sıkıştırma yapabilmektedir. Çizelge 3.9'da WinZip 10.0 kullanılarak Calgary Külliyatı'nın farklı yöntemler ile yapılan sıkıştırma sonuçları verilmiştir.

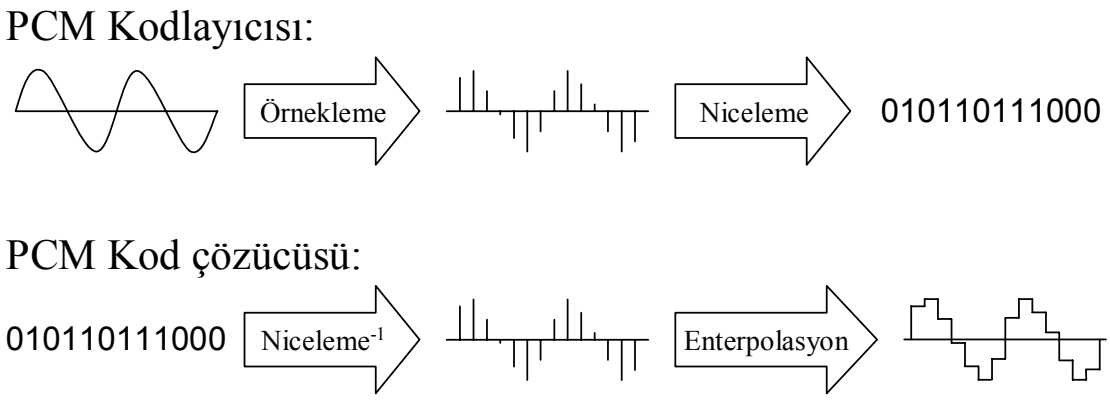
Çizelge 3.9. WinZip 10.0 ile Calgary Külliyatı'nın sıkıştırma sonuçları

Sıkıştırma Yöntemi	**Sıkıştırılmış Boyut (byte)**	**Verim (bit/kr)**
PPMd	745.089	1,90
Bzip2	829.559	2,11
Enhanced Deflate	988.926	2,52
Deflate	1.037.541	2,64

4. SES SIKIŞTIRMA

Ses sıkıştırmasından bahsetmeden önce sesin kodlama biçimlerine değinmek gerekir. Ses kodlama yöntemleri, ses dalgası kodlaması (waveform coding) ve konuşma kodlaması (speech coding) olarak iki grupta incelenir. Ses dalgası kodlamasında ses dalgaları belirli aralıklar ile örneklenerek sayısal hale getirilip kodlanırken, konuşma kodlamasında ise konuşmanın analizine dayalı bir model oluşturularak, bu model üzerinden parametreler üretilmesi ile kodlama yapılır. Konuşma kodlaması, ses dalgası kodlamasına göre daha yüksek oranda sıkıştırma sağlar. Ama konuşma kodlaması ile müzik gibi farklı yapıya sahip ses verileri sıkıştırıldığında çok düşük kalitede sonuç elde edilir. Ses dalgası kodlaması ise her türde sesi kaliteli bir şekilde sıkıştırabilir. Tezin bu bölümünde, *audio* türü seslerin sıkıştırılmasında tek alternatif olan ses dalgası kodlamasına dayalı sıkıştırma yöntemleri ele alınacaktır.

Analog ses dalgasının sayısal biçime dönüştürülmesi için örnekleme (sampling) ve niceleme (quantize) işlemlerinden geçirilmesi gereklidir. Bu işlemler genellikle Darbe Kod Modülasyonu (Pulse Code Modulation - PCM) yöntemi ile yapılır. PCM kodlayıcısının ve kod çözücüsünün ses üzerinde yaptıkları işlemler Şekil 4.1'de gösterilmiştir.



Şekil 4.1. PCM kodlama ve kod çözme

PCM ile ses sinyali belirli zaman aralıklarında örneklenir ve her örnek için dalganın yüksekliği kaydedilir. Nyquist teoremine göre, bir sinyalin en yüksek frekans

bileşeni f_0 ise, o sinyali en az $2 \times f_0$ ile örneklemek gerekir. Aksi halde örneklenmiş sayısal sesten, orijinal analog sesi geri getirmek imkânsız olur. Yapılan testler insan kulağının duyma eşik değerinin 20 kHz seviyesinde olduğunu göstermiş, bu nedenle Audio CD için frekans değeri belirlenirken bunun iki katından büyük olan 44.1 kHz değeri seçilmiştir. Her 1/44100 saniyede bir alınan örneklerin 16 bit büyüklüğünde bir sayı ile saklanması (niceleme işlemi) kaliteli bir ses elde etmek için yeterli görülmüştür. İnsan sesinin frekansı ise 4 kHz değerini aşmadığı için telefon haberleşmesinde 8 kHz ile örnekleme yapmak yeterlidir. Eğer niceleme işlemi her örnek için 16 bit yerine 8 bit ile gerçekleştirilirse, saklama kapasitesi yarıya inecek, fakat ses kalitesi de düşecektir. Örnekleme ve niceleme işlemlerinde bu gibi azaltmalar sesin kayıplı sıkıştırılması gibi düşünülebilir. Fakat gerçekte ses sıkıştırmada kayıplı ve kayıpsız ayrımı yapılırken, analog ses örneği PCM ile kodlandıktan sonra elde edilen sayısal ses örneği üzerinde yapılan işlemlerin kayıplı olup olmaması göz önüne alınır.

4.1. Kayıpsız Ses Sıkıştırma Yöntemleri

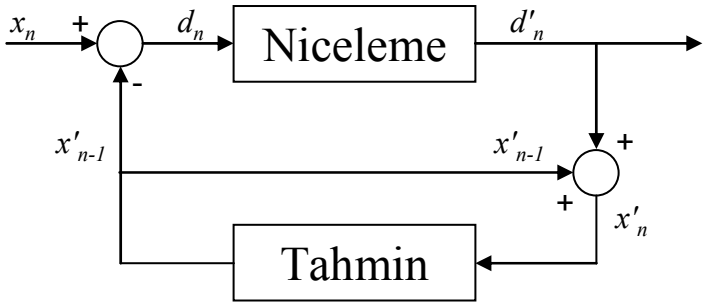
Kayıpsız ses sıkıştırma yöntemleri, PCM ile kodlanmış olan sayısal sesin kayıpsız olarak sıkıştırılmasını sağlayan yöntemlerdir. Dolayısı ile bu yöntemlerle sıkıştırılan orijinal PCM kodu aynı şekilde tekrar elde edilebilir. Kayıpsız ses sıkıştırma genellikle profesyonel amaçlı uygulamalar için gereklidir.

Lempel-Ziv, Huffman veya Aritmetik Kodlama gibi yöntemleri direkt olarak PCM ses verisine uygulamak, değer aralıklarının ve korelasyonun yüksek olması nedeniyle fazla etkili olmaz. Bu yüzden WinZip veya gzip gibi veri sıkıştırma araçları sayısal ses verisi üzerinde başarılı olamazlar. Eğer DPCM'de kullanılan *Doğrusal Tahmin* gibi bir yöntem kullanılırsa, korelasyon azaltılarak daha başarılı bir sıkıştırma gerçekleştirilebilir.

4.1.1. DPCM (Differential Pulse Code Modulation)

DPCM Bell Laboratuarları'nda İkinci Dünya Savaşı'nın bitiminden birkaç sene sonra geliştirilmiştir. Konuşma kodlama sistemi olarak çok popüler olan DPCM, halen sayısal telefon iletişiminde geniş çaplı olarak kullanılmaktadır.

Basit fark kodlama yöntemleri her örneğinin bir önceki örneğe olan farkını kodlarken, DPCM ise, kodlanacak örnek değeri önceki örneklerin yardımı ile tahmin ederek, örneğin gerçek değeri ile tahmin edilen değer arasındaki farkı kodlar. Bu açıdan bakıldığında, DPCM'de aslında artık değer kodlaması (2.5.2) yapıldığı görülmektedir. Öngörü yapmanın ana fikri fark değerlerini daha da küçültmektir. Örneğin, eğer önceki 3 sembol 2, 5 ve 8 ise, her seferinde 3 birimlik artış olduğu ve bir sonraki değerin 11 olabileceği tahmin edilebilir. Eğer bir sonraki değer 12 çıkarsa, 8 ile 12 arasındaki fark olan 4 değerini kodlamak yerine, 11 ile 12 arasındaki daha küçük olan 1 değerini kodlamak daha etkili sıkıştırma sağlayacaktır. DPCM'in kullandığı da yukarıdaki örnektekine benzer bir *Doğrusal Öngörü* (*Linear Prediction*) yaklaşımıdır. DPCM kodlayıcısının şeması Şekil 4.2'de verilmiştir.



Şekil 4.2. DPCM kodlayıcı şeması

4.1.2. ADPCM (Adaptive Differential Pulse Code Modulation)

DPCM'de yer alan niceleme ve öngörme aşamalarının, kodlanan ses verisinin karakteristik özelliklerine göre uyarlanabilir yapılması ile *Uyarlanır DPCM* (Adaptive DPCM - ADPCM) ortaya çıkmıştır. Uyarlama işlemi kodlayıcının girişinde uygulanırsa *ileri yönde*, çıkışında uygulanırsa *geri yönde* olarak tanımlanır [Sayood, 1996].

Öngörü işleminden önce, ses verisi blok ya da çerçeve olarak isimlendirilen yapılara bölünür. Konuşma kodlaması yapılırken blok uzunluğu genellikle 16 veya 20ms olarak belirlenir. Konuşma genellikle 8 kHz ile kodlandığı için bir blok 128 ya da 160 örnekten oluşur. İleri yönde uyarlanır öngörüde bir çerçeve işlenmeden önce, o çerçeve için en uygun öngörü ve niceleme katsayıları belirlenir ve bu değerler öngörü işleminin sonuç değerleri ile birlikte yan bilgi olarak aktarılır. Geri yönde uyarlanır

öngörüde ise, en uygun öngörü ve niceleme bir önceki çerçeveye dayalı olarak belirlendiği için, kod çözücü de aynı öngörüyü yapabileceğinden, sadece çerçeve bilgisinin gönderilmesi yeterlidir. Geri yönde uyarlanır öngörüde yan bilgi gönderilmediği için veri oranı daha düşüktür, fakat bir önceki çerçeve düşünülerek tahmin yapıldığı için tahmin hataları daha büyüktür.

Eğer örnekleme bit oranı gereğinden daha düşük belirlenirse, oluşacak olan örnekleme gürültüsü nedeniyle DPCM ve ADPCM kayıplı olabilir.

ITU-T G.726: 1990 yılında tamamlanan G.726, ITU-T'nin 15. çalışma grubu tarafından hazırlanmıştır. 40, 32, 24 ve 16 kbit/s sıkıştırma oranlı, ADPCM (Adaptive Differential Pulse Code Modulation) tabanlı bir sistemdir. Daha önce geliştirilen ve 1988'de tamamlanan ADPCM tabanlı G.721 ve G.723'ün yerini almıştır.

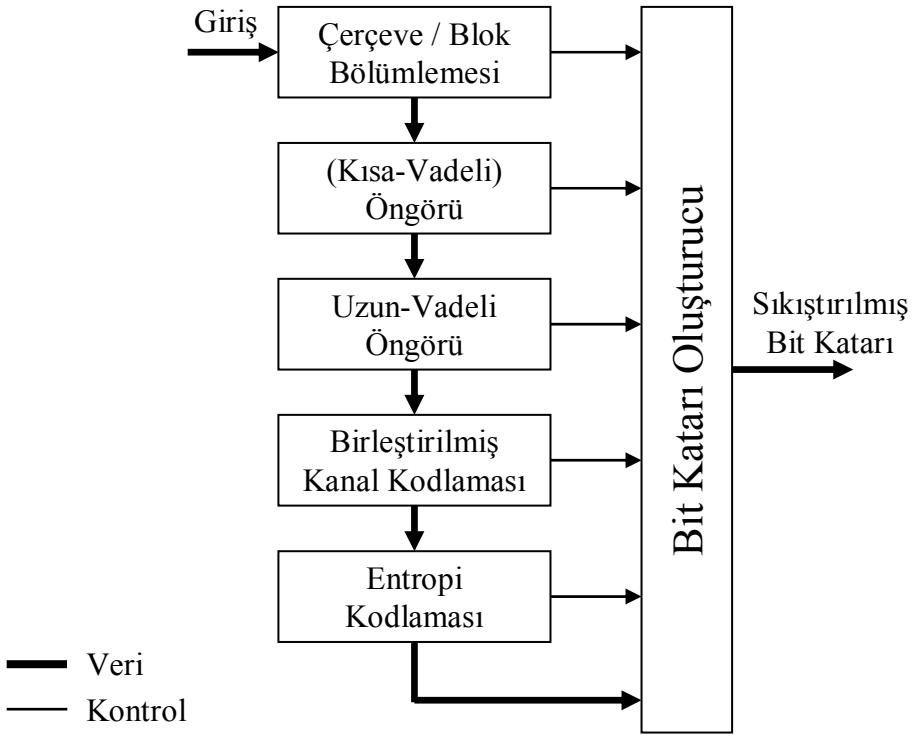
4.1.3. MPEG-4 ALS (Audio Lossless Coding)

Berlin Teknik Üniversitesi'nde geliştirilen bir kayıpsız ses kodlama sistemini temel alarak geliştirilen MPEG-4 ALS standardı, en son geliştirilen kayıpsız ses sıkıştırma yöntemlerinden biridir [ISO/IEC, 2005]. Etkili ve hızlı kayıpsız ses sıkıştırma yapan MPEG-4 ALS standardında diğer kayıpsız sıkıştırma yapan yöntemlerde olmayan birçok özellik mevcuttur:

- 32-bit PCM çözünürlüğünü ve 192 kHz frekansı destekler. (CD-DA'da 16-bit ve 44.1 kHz)
- 65536 kanala kadar çoklu kanal desteği vardır. (CD-DA'da 2)
- IEEE754 biçimindeki 32-bit kayan noktalı ses verisini destekler. (CD-DA'da tamsayı)
- wav, aiff, au, bwf ve raw gibi birçok sıkıştırılmamış ses formatını sıkıştırabilir.
- Kodlanmış verinin her bölgesine hızlı erişim sağlar.
- MPEG-4 Video sıkıştırması ile kullanılabilir.
- Kodlayıcı parametrelerinin sahip olduğu esneklik birçok uygulama için elverişlidir.

Bu önemli özelliklerinin yanında, uluslararası bir standart olması nedeniyle donanım üreticilerinin ve yazılım geliştiricilerin desteklediği bir yöntem olacaktır.

ALS kodlayıcısının temel yapısı Şekil 4.3'te gösterilmiştir [Liebchen, 2005]. Ses verisi önce çerçevelere ayrılır. Daha sonra, çerçevelerin içindeki tüm kanallar ses örneklerinden oluşan bloklara bölünür. Her blok için ileri yönde uyarlanır öngörü yapılarak, öngörünün hatası hesaplanır. Temel (kısa-vadeli) öngörü, uzun-vadeli öngörü ile birleştirilebilir. Birleştirilmiş Kanal Kodlaması aşamasında ise kanallar arasındaki benzerliklerden faydalanılarak sıkıştırma oranı arttırılır. En son aşamada ise geriye kalan tahmin farkı Huffman benzeri bir entropi kodlaması ile sıkıştırılır. Oluşturulan bit katarının çerçeveleri arasında rasgele erişim mümkündür.



Şekil 4.3. ALS kodlama şeması

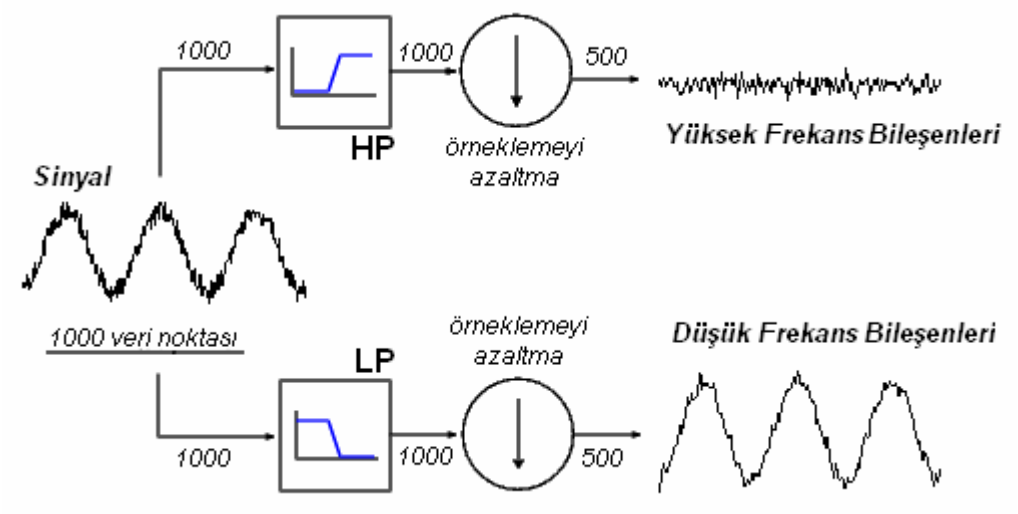
Doğrusal tahmin kodlaması analiz filtreleri ile yapılır. ALS ters kodlamasında önce ters Huffman kodlaması yapılır, sonra da sentez filtreleri kullanılarak sayısal ses sinyali eski haline getirilir. Kullanılan analiz sentez filtreleri küçük te olsa bir miktar kayba neden olabilmektedir.

4.2. Kayıplı Ses Sıkıştırma Yöntemleri

Kayıplı ses sıkıştırma yöntemleri, insan kulağının algılayamayacağı sesleri kodlamanın dışında bırakan ve daha sonra birçok ara işlem ile sıkıştırma oranını daha da arttıran yöntemler için kullanılır. Bu yöntemlerle kodlanmış olan sayısal ses, verilen kayıplar nedeniyle eski haline (ilk andaki PCM koduna) döndürülemez. Fakat uygun yöntemle ve uygun oranlarda sıkıştırma yapılırsa, kayıplı yöntemlerle sıkıştırılmış olan ses örneği, kayıpsız halini aratmayacak kalitede olabilir.

Bu bölümde günümüzde en yaygın olarak kullanılan kayıpsız ses sıkıştırma yöntemi olan MP3 ve onun yerini alması beklenen AAC yöntemleri anlatılacak, diğer yöntemler hakkında da özet bilgi verilecektir. Fakat önce kayıplı ses sıkıştırma yöntemlerinde ilk aşamalardan biri olan filtreleme işleminden bahsetmek gerekir.

Filtreleme: Belirli frekans bileşenlerini izole eden sisteme filtre denir. Alçak geçiren (LP: low-pass) filtreler belirli bir frekansın altında kalan bileşenlerin geçmesine izin verirken, yüksek geçiren (HP: high-pass) filtreler ise belirli bir frekansın üstünde kalan bileşenlerin geçmesine izin verir. Birçok sinyal için düşük frekanslı içerik sinyalin kimliğini verdiği için daha önemlidir. Yüksek frekanslı içerik ise sadece ayrıntıyı verir. İnsan sesi kaydedildikten sonra yüksek frekanslı bileşenleri atılırsa, ses farklılaşacak ama hala ne söylendiği anlaşılabilecektir. Ama düşük frekanslı bileşenler atılırsa, sinyalin asıl kimliği bozulacak ve anlamsız sesler duyulacaktır. Şekil 4.2'de gösterilen, bir alçak geçiren ve bir de yüksek geçiren filtreden oluşan yapı ile düşük ve yüksek frekanslı bileşenleri ayrıştırmak mümkündür. Çok fazlı filtre bankaları kullanılarak sinyaller daha fazla sayıda (genelde 2'nin katları olan sayılarda) alt bantlara da ayrıştırılabilir. Her alt bant sinyalden alınan örnek sayısını arttıracaktır. Örneğin Şekil 4.4'te sinyalden alınan 1000 adet örnek iki filtreden ayrı ayrı geçerek 2000 adete çıkacaktır. Filtrelerin ardından kullanılan örnekleme azaltma (down-sampling) devreleri ile artan örnek sayısını tekrar azaltmak mümkündür.



Şekil 4.4. İki kanallı alt bant kodlayıcısı

4.2.1. Yaygın Olarak Kullanılan Yöntem: MPEG-1 Layer III (MP3)

1987'de Almanya'daki "Fraunhofer Institut Integrierte Schaltungen" enstitüsünde EUREKA isminde bir ses sıkıştırma algoritması geliştirme projesi başlatıldı. 1989'da Fraunhofer Almanya'da algoritması için patent aldı. 1992'de bu algoritma ISO/IEC (International Organisation for Standardisation / International Electrotechnical Commission) tarafından MPEG-1 standardının audio kısmını oluşturması için kabul edildi. Bir yıl sonra MPEG-1 standardı yayınlandı [ISO/IEC, 1993]. 1995'te ABD'de MP3'ün patenti için başvuran Fraunhofer, 1996'da ABD patentini de aldı. 90'lı yılların sonlarına doğru Internet'te müzik paylaşımı için en çok kullanılan standart haline gelen MP3, günümüzde taşınabilir müzik çalarların neredeyse tamamının desteklediği en çok kullanılan ses sıkıştırma yöntemidir.

MPEG-1 Audio Standardı Layer I, Layer II ve Layer III olmak üzere üç farklı karmaşıklık ve sıkıştırma düzeyine sahiptir. En karmaşık olan MPEG-1 Layer III, genellikle MP3 ismi ile bilinir. Bu kodlamaların bant genişlikleri ve sıkıştırma oranları Çizelge 4.1'de verilmiştir. Verilen değerler varsayılan değerlerdir. MP3 standardı sadece 128 kbit/s ile değil, 8 ile 320 kbit/s arasında birçok farklı bant genişliği ile kodlayabilir.

Çizelge 4.1. MPEG-1 Audio Standardı düzeyleri

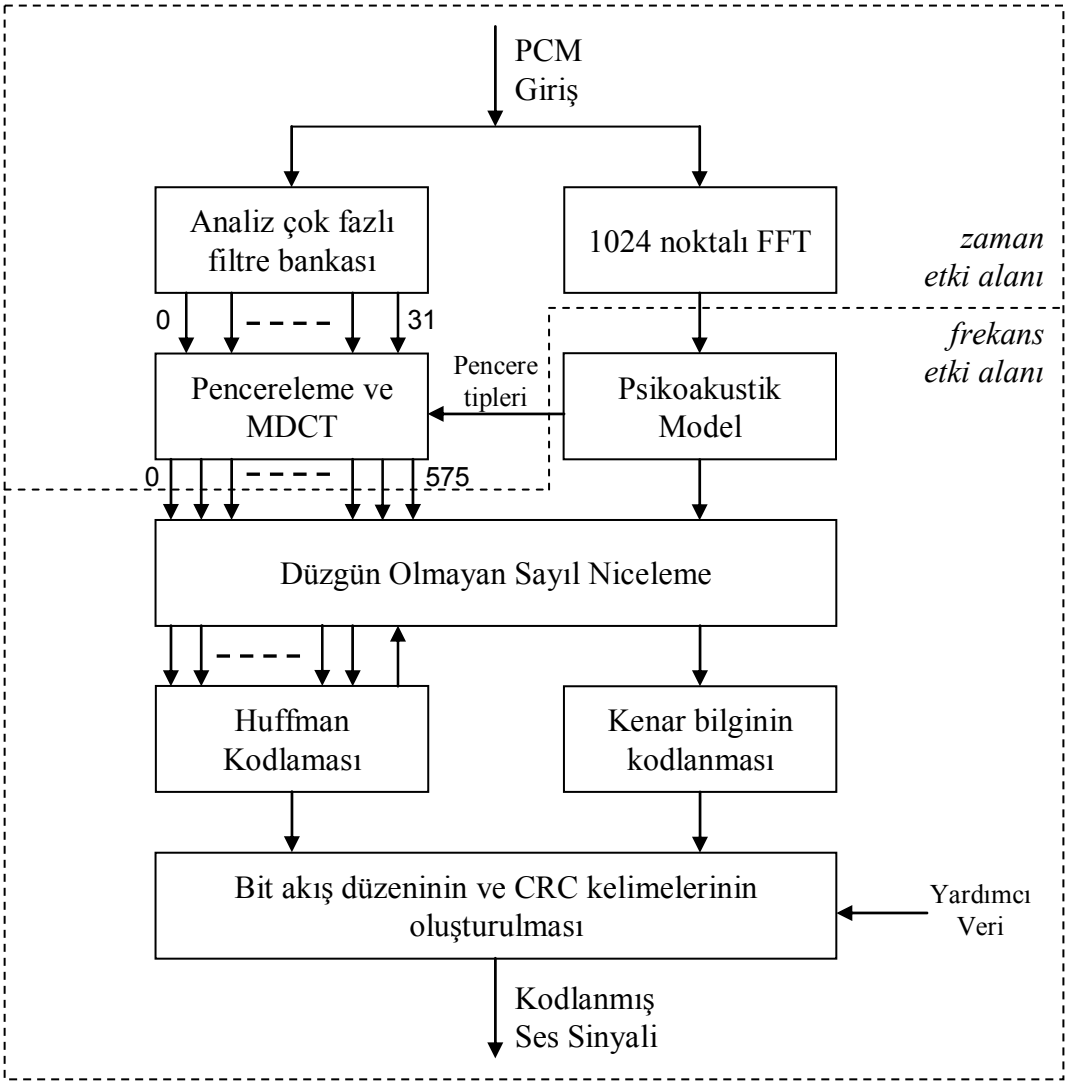
Kodlama	Bant genişliği	Oran
PCM CD Kalitesi	1.4 Mbps	1:1
Layer I	384 kbit/s	1:4
Layer II	192 kbit/s	1:8
Layer III (MP3)	128 kbit/s	1:12

MP3 standardı sabit bit oranı (Constant Bit Rate - CBR) kullanarak sıkıştırma yapabildiği gibi, değişken bit oranı (Variable Bit Rate - VBR) kullanarak ta sıkıştırma yapabilir. Bu sayede bir ses örneğinin karmaşık olan kısımları yüksek bit oranı ile, durağan olan kısımları daha düşük bit oranı ile sıkıştırılarak ses kalitesi arttırılabilir. Fakat VBR türü sıkıştırmanın dezavantajları da vardır. Bunlardan ilki, müzik çalarların zamanı yanlış göstermelerine veya hiç gösterememelerine neden olan zamanlama sorunu yaşatması, ikincisi ise en önemli kullanım alanlarından olan radyo veya televizyon yayınları için elverişli olmamasıdır.

MP3 standardı, 32 kHz, 44.1 kHz ve 48 kHz olmak üzere 3 farklı örnekleme frekansını ve tek kanal (mono), çift kanal, stereo ve birleşik stereo olmak üzere 4 farklı kanal tipini destekler. Çift kanal tipinin stereo'dan farkı, aynı yayın içinde iki farklı dilin iletilmesi gibi durumlarda kullanılabilmesidir. Birleşik stereo ise, sol ve sağ kanallar arasındaki benzerlikten faydalanarak daha etkin kodlama yapan bir kanal tipidir. *M/S (middle/side) Stereo* ve *Intensity Stereo* olarak adlandırılan iki farklı teknik ile gerçekleştirilir. *MS Stereo* tekniğinde sol (L) ve sağ (R) kanalları yerine middle (L+R) ve side (L-R) kanalları kullanılır. İki kanal birbirine yüksek oranda benzer ise side kanalı sürekli sıfıra yakın seyredeceği için sıkıştırmanın etkinliği artacaktır. *Intensity Stereo* tekniği ise, insan kulağının stereo etkinsi fark edemeyeceği kadar düşük veya yüksek frekanslı bölümlerin mono gibi kodlanması prensibine dayanır.

KODLAMA

Tüm MP3 dosyaları çerçeve (frame) adı verilen küçük parçalardan oluşmaktadır. Her çerçeve 1152 ses örneğinden oluşur ve 44.1 kHz örnekleme için 26 ms'lik zaman dilimine denk gelir. Şekil 4.5'te gösterilmiş olan MP3 standardının kodlama şemasında yer alan *Analiz çok fazlı filtre bankası* kısmında, her bir çerçeveyi oluşturan 1152 örnek 32 eşit uzunlukta frekans alt bandına bölünerek her alt banda 36 örnek yerleştirilir.



Şekil 4.5. MP3 kodlayıcısının blok şeması

Eğer PCM sinyalinin örnekleme frekansı 44.1 kHz ise, Nyquist frekansı 22.05 kHz olacaktır. Bu da her alt bandın yaklaşık olarak 22050 / 32 = 689 Hz genişliğinde olacağını gösterir. Her örnek kendi frekans aralığına karşılık gelen alt banda filtrelenir. PQF (polyphase quadrature filter) olarak bilinen bu filtre bankasının çıktısını temsil eden formül aşağıda verilmiştir:

$$St[i] = \sum_{k=0}^{63} \sum_{j=0}^{7} M[i][k] \times (C[k+64j] \times x[k+64j])$$

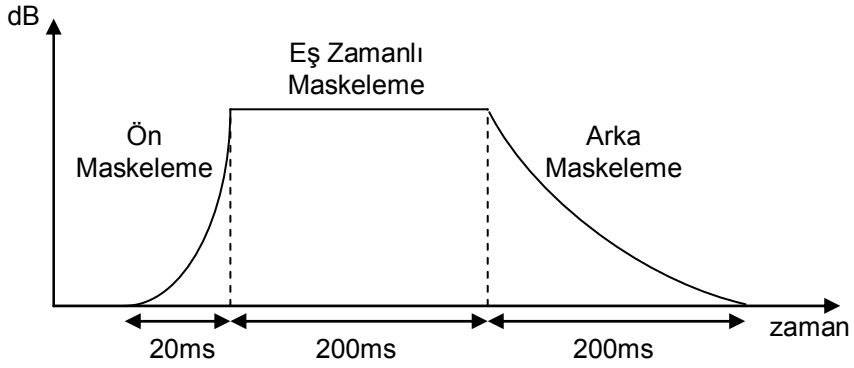
Bu formülde;

- i 0 ile 31 arasında değişen alt bant indeksi,
- $St[i]$ t 32 ses örnek aralığının bir tamsayı katı olmak üzere, i alt bandının t zamanındaki filtre çıktısı,
- $C[n]$ standartta tanımlanmış olan analiz penceresinin 512 katsayısından biri,
- $x[n]$ 512 büyüklüğündeki tampondan okunan bir ses girdi örneği,
- $M[i][k]$ ise analiz matrisi katsayılarıdır. $M[i][k] = \cos\left[\frac{(2i+1)(k-16)\pi}{64}\right]$

Sinyal *Analiz çok fazlı filtre bankası* ile işlenirken, aynı zaman diliminde sinyale *Hızlı Fourier Dönüşümü* (FFT) de uygulanır. FFT ile zaman etki alanından frekans etki alanına geçirilen sinyal daha sonra *Psiko-akustik model*'e gönderilir. Bu model, insan kulağının hangi sesleri algılayabileceğini hangilerini algılayamayacağını belirleyebilen bir yapıdır. İnsan kulağı duyma eşik değerlerinin dışında kalan sesleri veya kuvvetli bir ses tarafından maskelenen zayıf sesleri duyamaz. Bu nedenle bu sesler kodlamaya dâhil edilmezler.

Duyma eşik değerleri: İnsan kulağı 20 Hz ile 20 kHz arasındaki sesleri duyabilir. En iyi fark edilebilen sesler 2 kHz ile 4 kHz aralığındaki seslerdir. Sesin frekansı bu aralıktan uzaklaştıkça duyabilmek için sesin seviyesini arttırmak gerekir.

Maskeleme etkisi: Güçlü bir sesin var olduğu durumlarda, zayıf sesler güçlü ses tarafından maskeleneceği için duyulmazlar. Örneğin bir kasetçalarda dinlenen şarkılar arasındaki tıslama sesi, şarkılar çalarken de vardır ama duyulmaz. *Eş zamanlı maskeleme* olarak bilinen bu durumda, baskın olan kuvvetli sese maskeleyen sinyal, duyulamayacak olan zayıf sese de maskelenen sinyal denir. Kuvvetli olan sinyal, başlamadan 20 ms önce zayıf sinyal üzerinde maskeleme etkisi göstermeye başlar ve sinyal bittikten 200 ms sonrasına kadar bu etki azalarak devam eder. Kuvvetli sinyalin kendisinden önce gelen maskeleme etkisine *ön maskeleme*, kendisinden sonra devam eden maskeleme etkisine ise *arka maskeleme* denir. Şekil 4.6'da maskeleme etkileri gösterilmiştir.



Şekil 4.6. Ön, eş zamanlı ve arka maskeleme

Psiko-akustik model ile elde edilen bilgiler, MDCT'nin (Modified Discrete Cosine Transform) hangi pencere tipini seçmesi gerektiğine ve *düzgün olmayan* (*nonuniform*) *niceleme* işleminin frekans çizgilerini ne şekilde nicelemesi gerektiğine karar vermede yardımcı olur. Normal (uzun), kısa, başlangıç ve bitiş olmak üzere 4 farklı pencere tipi vardır. Aşağıda pencere tiplerinin fonksiyonları verilmiş, Şekil 4.7'de ise pencere tipleri grafiksel olarak gösterilmiştir [Jacaba, 2001].

a) Normal (uzun) pencere (Tip 0)

$$z_i = x_i \sin\left(\frac{\pi}{36}\left(i+\frac{1}{2}\right)\right) \qquad ,\mathrm{i} = 0 \sim 35$$

b) Başlangıç penceresi (Tip 1)

$$z_i = \begin{cases} x_i \sin\left(\dfrac{\pi}{36}\left(i+\dfrac{1}{2}\right)\right) & ,\mathrm{i} = 0 \sim 17 \\ x_i & ,\mathrm{i} = 18 \sim 23 \\ x_i \sin\left(\dfrac{\pi}{12}\left(i-18+\dfrac{1}{2}\right)\right) & ,\mathrm{i} = 24 \sim 29 \\ 0 & ,\mathrm{i} = 30 \sim 35 \end{cases}$$

c) Kısa pencere (Tip 2)

36 örnek birbirleri ile kesişim halinde olan 3 bloğa bölünür:

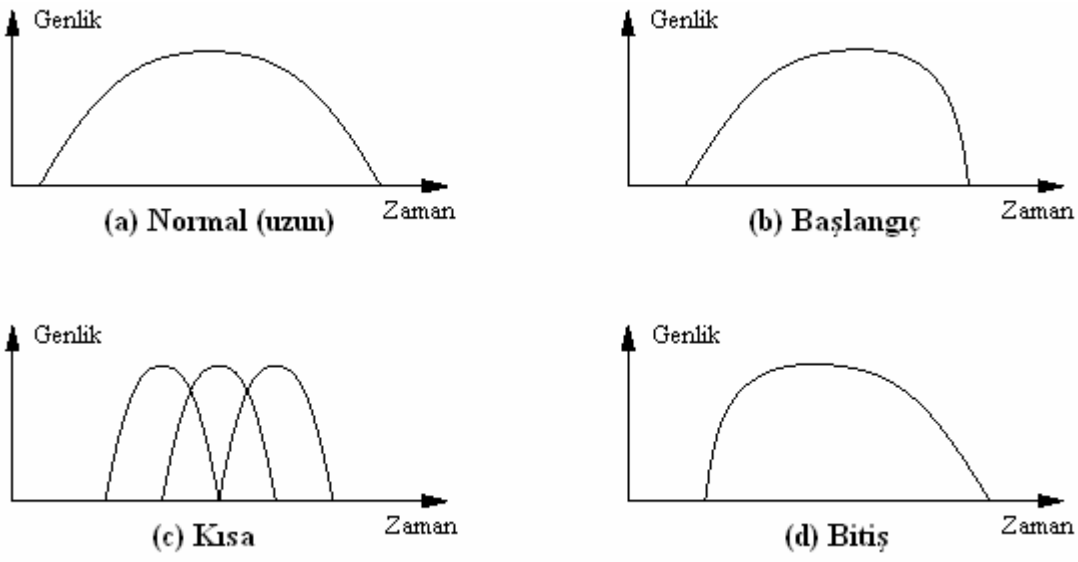
$$y_i^{(k)} = x_{i+6(k+1)} \qquad ,\mathrm{i} = 0 \sim 11, \mathrm{k} = 0 \sim 2$$

Her küçük blok ayrı ayrı pencerelenir:

$$z_i^{(k)} = y_i^{(k)} \sin\left(\frac{\pi}{12}\left(i + \frac{1}{2}\right)\right) \qquad ,\mathrm{i} = 0 \sim 11, \mathrm{k} = 0 \sim 2$$

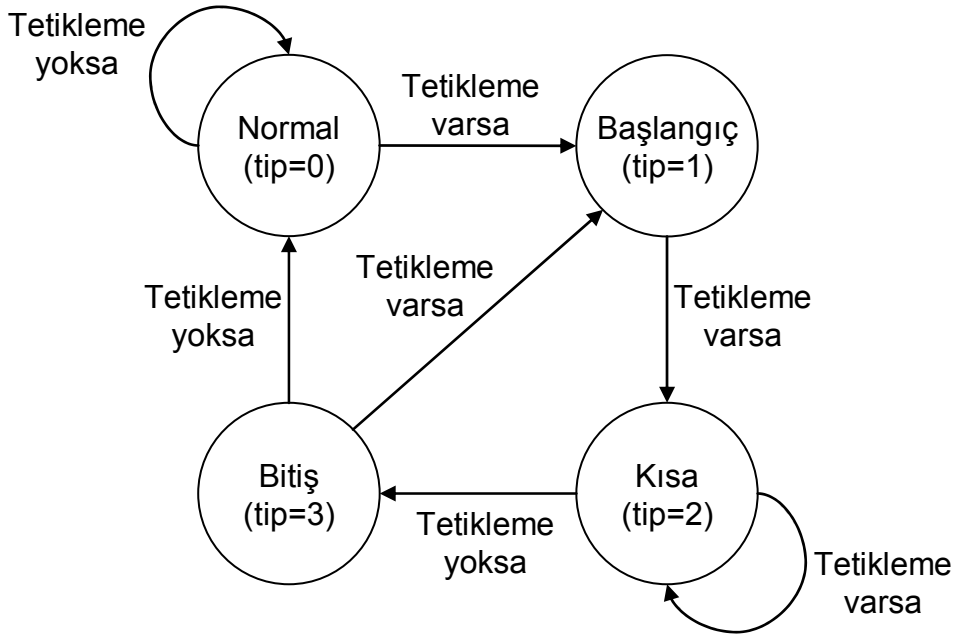
d) Bitiş penceresi (Tip 3)

$$z_i = \begin{cases} 0 & ,\mathrm{i} = 0 \sim 5 \\ x_i \sin\left(\frac{\pi}{12}\left(i - 18 + \frac{1}{2}\right)\right) & ,\mathrm{i} = 6 \sim 11 \\ x_i & ,\mathrm{i} = 12 \sim 17 \\ x_i \sin\left(\frac{\pi}{36}\left(i + \frac{1}{2}\right)\right) & ,\mathrm{i} = 18 \sim 35 \end{cases}$$



Şekil 4.7. Pencere Tipleri

Eğer alt bant örnekleri sürekli bir çizgide hareket ediyorsa (durağan ise) normal pencere tipi (Tip 0) seçilerek uzamsal çözünürlük genişletilir. Eğer örnekler süreksiz (değişken) ise kısa pencere tipi (Tip 2) seçilerek zaman çözünürlüğünü arttırmak için frekans etki alanındaki alt bant çıktıları bölünür. Bu iki pencere tipi arasında keskin geçişler olmaması için iki tane geçiş pencere tipi tanımlanmıştır. Başlangıç pencere tipi (Tip 1) normal'den kısa'ya, bitiş pencere tipi (Tip 3) ise kısa'dan normal'e geçmek için kullanılır. Şekil 4.8'de pencere tipleri arasındaki geçiş yapabilme ilişkileri gösterilmiştir.



Şekil 4.8. Pencere değiştirme karar mekanizması

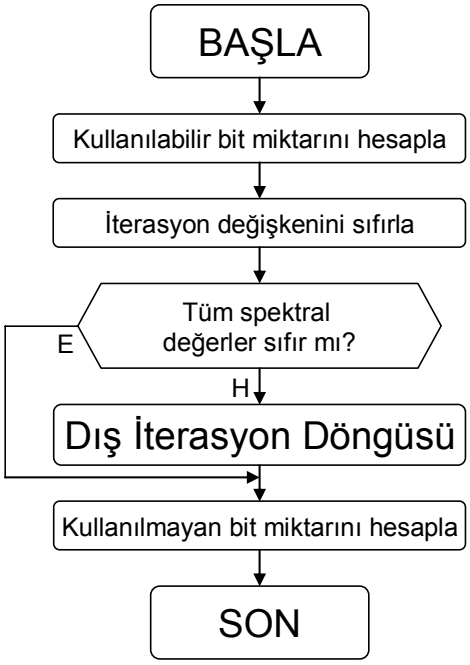
Filtreleme işleminden sonra, her alt banda ait örnekler halen zaman etki alanındadır. Bu örnekleri frekans etki alanına geçirmek için her alt banda MDCT uygulanır. Aşağıda MDCT dönüşümünün formülü verilmiştir:

$$X_i = \sum_{k=0}^{n-1} z_k \cos\left(\frac{\pi}{2n} \left(2k + 1 + \frac{n}{2} \right) (2i + 1) \right)$$

Bu formülde *n* değeri kısa bloklar için 12, uzun bloklar için 36'dır ve *i* değeri de 0 ile (*n*/2)-1 aralığında değişmektedir. Bu durumda kısa blok sayısının 6, uzun blok sayısının ise 18 olacağı görülmektedir. Kısa blok kipinde üç kısa blok bir uzun bloğun

yerini tutar. Böylece, ses örneklerinin bir çerçevesi için MDCT örneklerinin sayısı, blok büyüklüğü ne olursa olsun değişmeyecektir. Ses örneklerinin verilen bir çerçevesi için, MDCT ya hep aynı blok büyüklüğünde (uzun ya da kısa), ya da karışık-blok kipinde olabilir. Hep aynı blok büyüklüğünde iken 32 alt bandın her biri ya 6 ya da 18 bloğa bölünerek 192 ya da 576 frekans çizgisinden oluşan birer tanecik (granule) oluştururlar. Karışık blok kipinde ise iki düşük frekans alt bandı için normal pencere, diğer 30 yüksek alt bant için kısa pencere kullanır. Bu kip, yüksek frekanslar için zaman çözünürlüğünü feda etmeden, düşük frekanslar için daha iyi frekans çözünürlüğü sağlar.

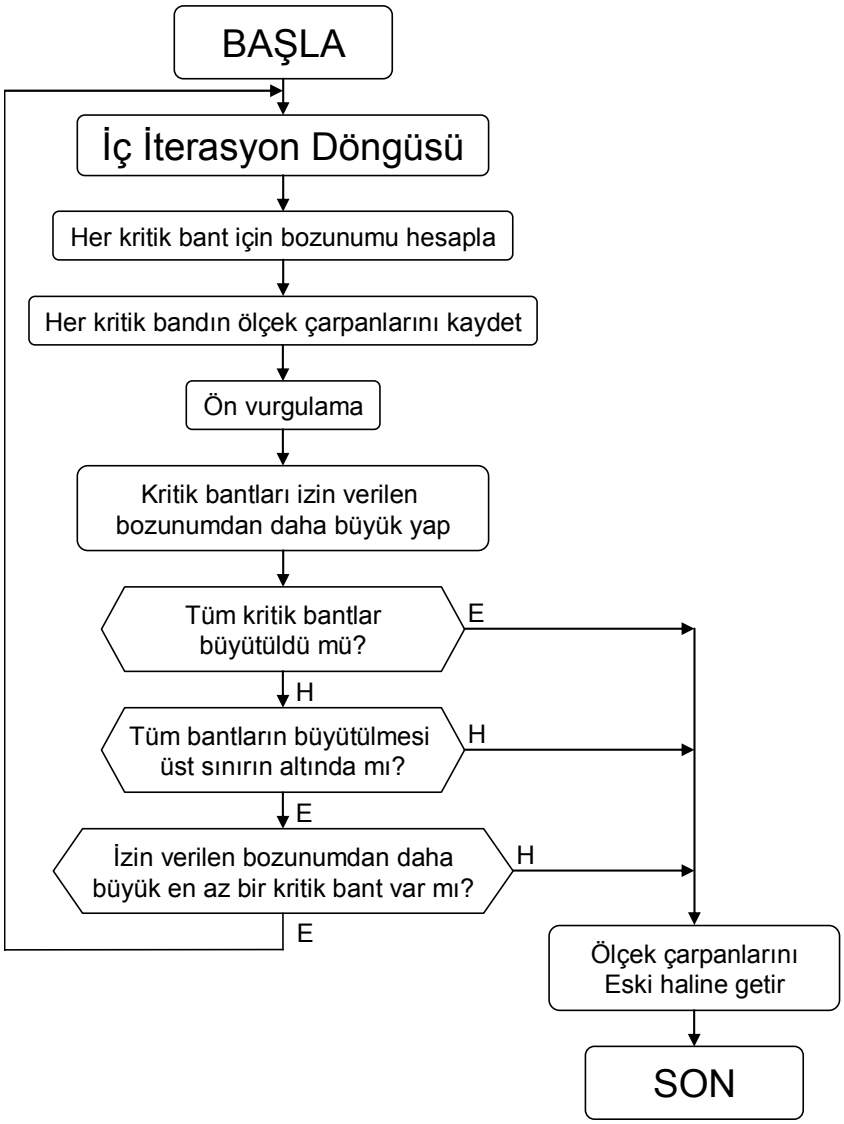
En çok zaman alan kısım olan düzgün olmayan niceleme işlemi, 576 değer için tekrarlanan iç içe iki döngü ile gerçekleştirilir. Dıştaki döngü *bozunum kontrol döngüsü* (*distortion control loop*), içteki ise *oran kontrol döngüsü* (*rate control loop*) olarak adlandırılır. Dış döngüyü çağıran *döngü çerçeve programı*'nın akış şeması Şekil 4.9'da verilmiştir.



Şekil 4.9. Döngü Çerçeve Programı

Bozunum kontrol döngüsü, oran kontrol döngüsü içinde frekans etki alanının çizgilerinin nicelenmesi ile üretilen niceleme gürültüsünü kontrol eder. Amaç, her ölçek çarpanı bandı için niceleme gürültüsünü maskeleme eşik değerinin altında tutmaktır.

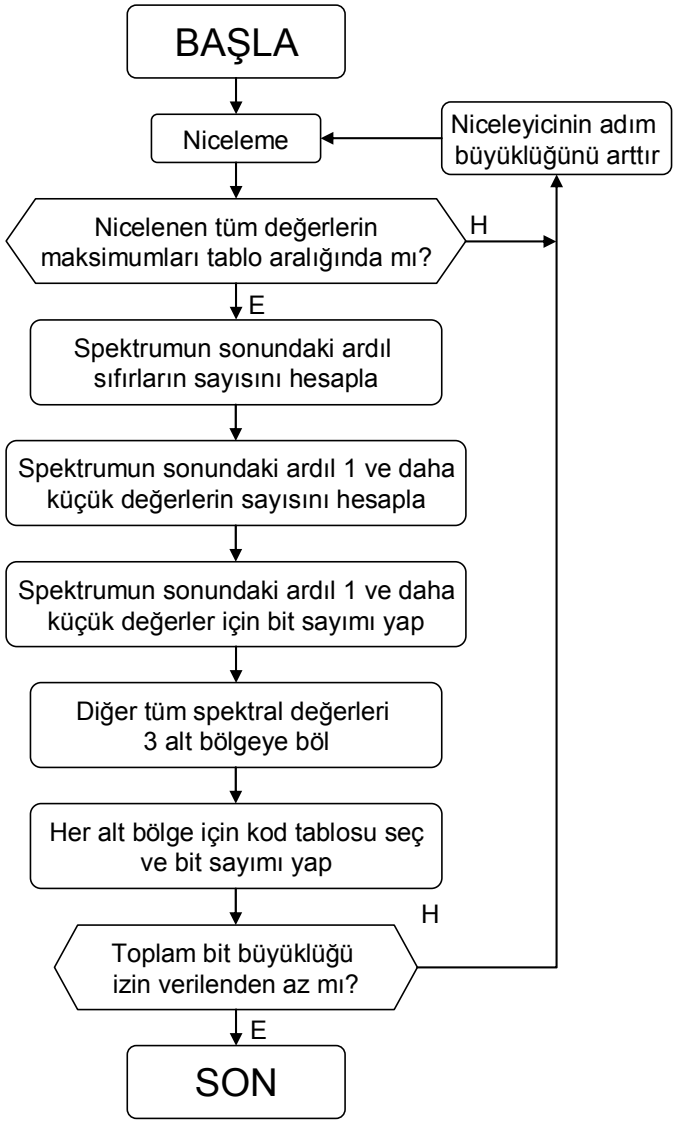
Eğer niceleme gürültüsünün maskeleme eşik değerini aştığı tespit edilirse, o bant için niceleme gürültüsünü azaltacak şekilde ölçek çarpanı ayarlanır. Oran kontrol döngüsü çağrılmadan önce, ölçek çarpanları ve niceleme adım büyüklükleri kaydedilir. Bozunum kontrol döngüsünün akış şeması Şekil 4.10'da verilmiştir.



Şekil 4.10. Bozunum Kontrol Döngüsü (Dış Döngü)

Oran kontrol döngüsü frekans etki alanı örneklerinin nicelenmesi işlemini gerçekleştirir ve bir sonraki aşamada kullanılacak olan niceleme adım büyüklüğüne karar verir. Nicelenen değerler Huffman Tabloları kullanılarak sıkıştırıldıktan sonra, eğer kod uzunluğu çerçeve için belirlenen sınırlardan halen büyükse, daha küçük

nicelenmiş değerlere ulaşabilmek için niceleme adım büyüklüğü arttırılır ve uygun büyüklüğe ulaşılana dek bu işlem tekrarlanır. Büyük değerlerin bölünerek bölgelere ayrılması, her bölge için uygun Huffman Tablosunun seçimi ve bölgeler arasındaki sınırların hesaplanması da bu iç döngüde yapılır. Oran kontrol döngüsünün akış şeması Şekil 4.11'de verilmiştir.



Şekil 4.11. Oran Kontrol Döngüsü (İç Döngü)

Kodlayıcı tarafından üretilen tüm parametreler ses sinyalinin tekrar elde edilmesinde kullanılmak üzere toplanırlar. Bunlar çerçevenin kenar bilgisi kısmını oluştururlar.

Kodlama işleminin son aşaması uygun dosya çerçeve yapısının oluşturulmasıdır. Çerçeve başlığı, CRC, kenar bilgisi, Huffman ile kodlanmış frekans çizgileri ve diğer yardımcı veriler biraraya getirilerek çerçeveler oluşturulur. Bu çerçevelerin her biri 1152 adet kodlamış PCM örneği içerir.

4.2.2. Yeni Yöntem: MPEG AAC (Advanced Audio Coding)

MPEG AAC, MP3 ses kodlayıcısının sıkıştırma performansı geliştirilmiş ve özellikleri arttırılmış yeni bir sürümüdür. MPEG-2 hareketli görüntü sıkıştırma sistemi 1994'te ilk yayınlandığında, ses kodlama kısmında MPEG-1 Audio kodlama sistemi ile büyük ölçüde benzer olan MPEG-2 Audio (MPEG-2 Part 3) sistemini kullanmaktaydı [ISO/IEC, 1994]. AAC ise ikinci bir ses sıkıştırma yöntemi olarak 1997 yılında MPEG-2 standardına eklenmiştir [ISO/IEC, 1997]. MPEG-2 NBC (Non-Backward Compatible) olarak ta bilinen bu yöntem MPEG-1 Part 3 ve MPEG-2 Part 3 ses kodlama yöntemleri ile geriye doğru uyumlu değildir. Yani bir AAC kod çözücüsü bir MP3 dosyasını açamaz. AAC, daha sonra biraz daha geliştirilerek MPEG-4 hareketli görüntü sıkıştırma sisteminde de kullanılmıştır (MPEG-4 Part 3) [ISO/IEC, 1999]. MPEG-4 AAC, MPEG-2 AAC'ye göre geriye doğru uyumludur.

AAC, MP3'te olan kodlama tekniklerini daha etkili bir şekilde kullanır:

- Filtre bankası MP3'te olduğu gibi karışık değil saf MDCT'dir.
- Daha iyi frekans çözünürlüğü sağlamak amacıyla uzun pencereler MP3'tekilerden yaklaşık iki kat daha büyük hale getirilmiştir.
- Daha az ön-yankı ve daha iyi pencere değiştirme idaresi sağlamak için kısa pencereler MP3'tekilerden daha küçük hale getirilmiştir.
- M/S stereo çerçevenin tamamı yerine alt bant temelli yapılabilmektedir.
- Yoğunluk stereo sadece bitişik alt bant gruplarında kullanılmak yerine alt bant temelli yapılabilmektedir.

AAC, MP3'te olmayan TNS ve frekans etki alanı öngörüsü araçlarını da içermektedir. TNS (Temporal Noise Shaping) filtrelenmiş katsayıların iletimi sayesinde niceleme gürültüsünün yerini kontrol etmek amacıyla tasarlanmış bir araçtır. Frekans

etki alanı öngörüsü ise durağan sinyallerin sıkıştırılabilirliğini arttırmak amacıyla tasarlanmış bir araçtır.

MPEG-4 AAC, MPEG-2 AAC'de olmayan PNS ve LTP araçlarını içermektedir. PNS (Perceptual Noise Substitution) Sinyalin gürültüye benzer kısımları kodlanmayıp, bu gürültü yerine kod çözücü tarafında üretilen bir gürültü kullanılır. LTP (Long Term Prediction) ise, frekans etki alanı öngörüsüne göre daha az işlem gücü ile çalışan bir tahmin aracıdır.

MPEG-2 AAC kodlama sistemi, bellek ve işlem gücü gereksinimleri ile kodlama kalitesi arasındaki ödün verme ilişkisinin sağlanabilmesi için 3 farklı profil önermiştir:

- Ana (Main) Profil: Tüm veri oranlarında en iyi ses kalitesinin elde edildiği, buna karşılık işlem gücü ve bellek gereksinimi en çok olan profildir.
- Düşük-Karmaşıklık (LC: Low-complexity) Profili: Çok iyi ses kalitesi sağladığı gibi, fazla bellek ve işlem gücü de gerektirmediği için birçok uygulama için uygun bir profildir.
- Ölçeklenebilir örnekleme oranı (SSR: Scalable sampling rate) Profili: Dört eşit parçaya bölünmüş olan bant genişliği ile frekans ölçeklenebilirliği sağlar.

MPEG-4 AAC kodlama sisteminde zamanla başka profiller de eklenmiştir:

- Uzun Vadeli Öngörü (LTP: Long Term Prediction) Profili: Düşük işlem karmaşıklığına sahip bir ileri yönde öngörü aracı kullanarak kazancı arttırır. 1999 yılında eklenmiştir.
- Yüksek Verimlilik (HE: High Efficiency) Profili: 2003 yılında eklenen HE profili, LC Profili ile SBR (Spectral Band Replication) nesnesinin birleşimi ile daha etkin sıkıştırma sağlar. 2004 yılında geliştirilen ikinci sürümü (v2) ise parametrik stereo özelliği sayesinde düşük bit oranlarında bile iyi ses kalitesi sağlar.

Çizelge 4.2'de her profilin kod çözücülerinin karmaşıklık seviyesi verilmiştir. Bu çizelgede işlemci karmaşıklık birimi (PCU: Processor Complexity Units) olarak

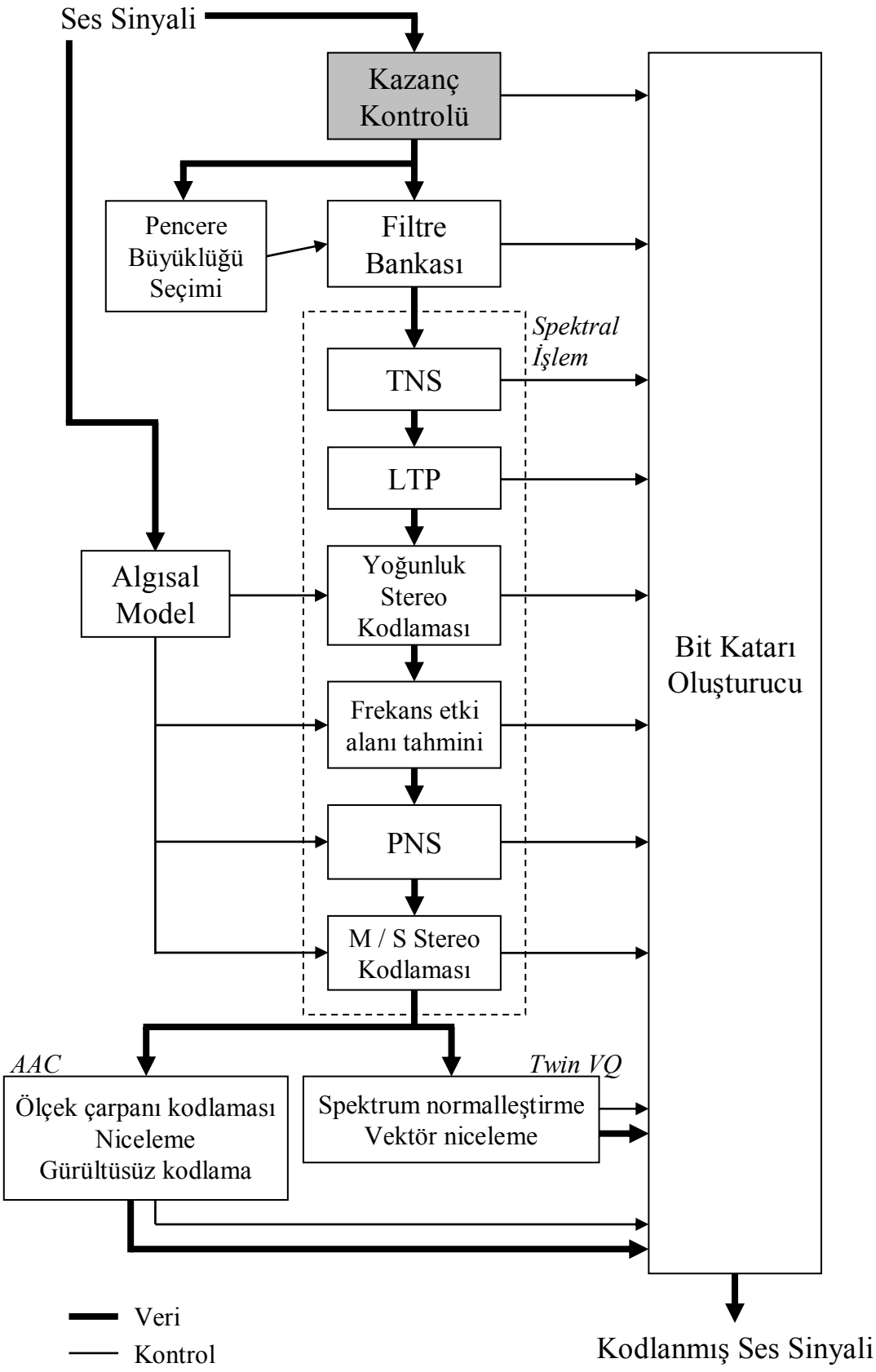
MOPS (Millions of Operations Per Second) kullanılmış, RAM karmaşıklık birimi (RCU: RAM Complexity Units) olarak ise kaç bin kelimelik bellek ihtiyacı olduğunu gösteren kWord kullanılmıştır [MPEG4-3, 2001].

Çizelge 4.2. AAC kod çözücülerinin karmaşıklığı

Profil	Frekans	PCU (MOPS)	RCU (kWord)
Main	48 kHz	5	5
LC	48 kHz	3	3
SSR	48 kHz	4	3
LTP	48 kHz	4	4

Şekil 4.12'de AAC kodlayıcısının blok şeması verilmiştir. İlk aşama olarak yer alan *Kazanç Kontrolü* birimi aslında sadece SSR profilinde kullanılır. Bu birimin gerçekleştirdiği ön işlem bir PQF, kazanç sezicileri ve kazanç değiştiricilerinden oluşur. PQF her ses kanalındaki giriş sinyallerini 4 eşit frekans bandına böler. Kazanç sezicisi tarafından gerekli görülürse, her fitre bankasının çıktısı ön-yankı etkilerini azaltmak için kazanç değiştirmesine tabi tutulur. Kazanç kontrolü 4 bandın her birine bağımsız olarak uygulanabilir. 4 frekans bandının her birine MDCT uygulandığında toplam MDCT blok büyüklüğünün çeyreği kadar spektral katsayı elde edilir (512/2048 veya 256/1024).

Blok şemasında ikinci sırada yer alan standart filtre bankası, SSR profili haricinde kullanıldığında, blok büyüklüğünü 256 ve 2048 arasında dinamik olarak değiştirebilen bir MDCT'dir. MP3'e göre uzun bloğun büyüklüğü 2 misli daha fazla olduğu için düz sinyallerde daha yüksek bir kodlama etkinliği sağlar. Kısa bloğun büyüklüğünün de MP3'tekine göre daha küçük olması sayesinde, kısa süreli sinyallerin kodlanması daha etkin hale gelmiştir. Tüm bloklar bir önceki ve bir sonraki blok tarafından %50 oranında kapsanır. Frekans seçiciliğini arttırmak için gelen ses örnekleri MP3'te olduğu gibi dönüşümden önce çerçevelenir. MPEG-4 AAC, dinamik olarak değişebilen iki farklı pencere tipi sağlar. Bunlar: Sinüs-biçimli pencere ve Kaiser-Bessel türevi penceredir.



Şekil 4.12. AAC kodlayıcısının blok şeması

Farklı blok uzunluklarında serilere sahip olan iki ses kanalı arasındaki eşzamanlılığın sağlanması için, kısa serinin başlangıcı ve sonu için özel olarak tasarlanmış iki geçiş penceresi ve her biri %50 oranında birbiri üzerinde geçişli olan 8 küçük dönüşüm uygulanır. Bu işlem, ardışık bloklar arasındaki mesafeyi 2048 giriş örneğinde sabit tutar.

Temporal Noise Shaping (TNS): TNS, her filtre bankası içindeki niceleme gürültüsünün zamansal yapısı üzerinde bazı kontroller yapabilmek için, spektral değerlerin nicelenmesi ve kodlanması amacıyla kullanılır [Herre ve Johnston, 1996]. TNS, konuşma sinyalleri gibi zaman içinde çok büyük farklılıklar gösteren sinyaller için çok etkili bir araçtır. Bu gibi durumlarda, kodlama verimliliğini azaltan kısa pencerelere çok sık geçiş yapma işlemi yerine, niceleme gürültüsünün zamansal yapısını kontrol etmek için TNS kullanılabilir. TNS'in temel mantığı, frekans etki alanında (örneğin frekans ekseni üzerinde) ileri yönde tahmin yürütmektir. Orijinal spektrumu bir filtreden geçirerek filtrelenmiş sinyali niceler. Nicelenmiş filtre katsayıları aynı zamanda bit katarına aktarılır. Bu katsayılar, kod çözme aşamasında kodlanırken yapılan filtrelemeyi geri almak için kullanılır. TNS, geleneksek iki adımlı filtre bankası yerine sürekli sinyale ayarlanabilen filtre bankası kullanan, dönüşümün arka işlem adımı olarak düşünülebilir. TNS yaklaşımının AAC'deki uyarlaması en fazla üç ayrı filtrenin giriş sinyalinin farklı spektral bölgelerine uygulanmasına izin verir.

Long Term Prediction (LTP): MPEG-2 AAC'de yer almayan LTP, MPEG-4 AAC'de var olan ardıl kodlama çerçeveleri arasındaki bir sinyalin artık değerlerini azaltmak için çok etkili bir araçtır [Ojanperä ve Väänänen, 1999]. LTP, akort düdüğü gibi sabit armonik tonlu sinyaller için çok etkili olmakla beraber, armonik tonlu olmayan sinyaller için de kazanç sağlayabilir. Uygulama karmaşıklığı, MPEG-2 AAC'deki frekans etki alanı öngörü işlemine göre çok daha düşüktür. LTP bir ileri yönde uyarlanır öngörü olduğu için (öngörü katsayıları yan bilgi olarak iletilir) iletilen spektral katsayılardaki bit hatalarına veya kod çözücüdeki yuvarlama hatalarına karşı daha az hassastır.

Çoklu kanal kullanıldığı durumlarda yoğunluk stereo kodlaması ve M/S stereo kodlaması işlemleri sıkıştırmanın etkinliğini arttırır. LTP aşamasından sonra bunlardan

ilki olan yoğunluk stereo kodlaması uygulanır. Bu işlem, 2 kHz'nin üzerindeki yüksek frekanslarda insan duyma sisteminin sesin yerini belirlerken sinyallerin kendilerinden çok kritik-bant-filtrelenmiş biçimlerini kullandığı ispatlanmış olan, bazı algısal bulgular temel alınarak uygulanır. Sol ve sağ sinyalleri tek bir sinyal gibi kodlar ve yanında bir de yön bilgisi gönderir. Kayıplı bir kodlama yöntemi olduğu için genellikle düşük bit oranlarında kullanılır. Yüksek bit oranlarında sadece M/S stereo kodlaması kullanılır.

Sadece AAC ana profilinde kullanılan frekans etki alanı öngörüsü (tahmini), düz sinyallerin bulunduğu kısımlarda önceki örneklere göre tahmin yürüterek fazlalık olan bilgiyi azaltır. Düz sinyallere genellikle uzun bloklarda rastlanacağı için kısa bloklarda kullanılmaz. Öngörü işlemi, her frekans çizgisi için bağımsız olarak hesaplanan ikinci dereceden geri yönde uyarlanır kafes (lattice) yapısındadır. Gerçek değerler yerine öngörülmüş değerlerin kullanılmasına, bu bantta ulaşılan öngörü kazancına bağlı olarak ölçek çarpanı bandı tabanında karar verilebilir. Öngörü işleminin kararlılığını arttırmak için, tahsis edilen bir bit katarı elemanı vasıtasıyla kullanılan, kodlayıcı ve kod çözücü arasında senkronize edilmiş devirli reset mekanizması yer almaktadır. Frekans etki alanı öngörüsü için gerekli olan yüksek işlem gücü ihtiyacı ve ondalıklı sayı kullanma gereksinimi, sabit noktalı platformlarda bu aracın kullanılmasını güçleştirir. Aynı zamanda, öngörü işleminin geri yönde uyarlanır yapısı bu bit katarlarını aktarım hatalarına karşı daha hassas hale getirir.

Perceptual Noise Substitution (PNS): PNS, gürültü benzeri sinyal bileşenlerinin daha küçük yapılarla temsil edilmesi ile sıkıştırma etkinliğini arttıran bir araçtır [Herre ve Johnston, 1998]. Öncelikle kodlayıcıda giriş sinyalinin gürültüye benzer bileşenleri ölçek çarpan bandı tabanında tespit edilir. Gürültü benzeri sinyal bileşenlerini içeren ölçek çarpan bantlarına ait olan spektral katsayılar gurubu, niceleme ve kodlamaya tabi tutulmaz. Bu bantlar için bir gürültü temsil etme bayrağı ve temsil edilen spektral katsayıların toplam gücü iletilir. İletilen bu veriler sayesinde kod çözme aşamasında temsil edilmesi istenen gürültü oluşturulacaktır. MPEG-2 AAC'de yer almayan PNS, MPEG-4 AAC'nin düşük bit oranlarındaki etkinliğini arttırır.

Çoklu kanal kullanıldığı durumlarda uygulanabilecek bir başka stereo birleştirme işlemi de M/S Stereo kodlamasıdır. Bu işlem, sol ve sağ kanal sinyallerine

bir matris uygulayarak, iki sinyalin toplamını ve farkını hesaplar. Sinyalin sol ve sağ kanallardan aynı miktarda geldiği durumlarda M/S Stereo kodlaması önemli ölçüde yer kazancı sağlar. Kod çözücüde ters matris uygulanarak kanallar orijinal haline getirilir.

İzgel değerlerin nicelenmesi tüm dönüşüm kodlayıcılarında olduğu gibi AAC'de de en önemli bit oranı azaltma kaynağıdır. Niceleme sayesinde algısal model tarafından belirlenen sınırlar çerçevesinde spektral değerlere bit tahsis edilmesi işlemi gerçekleştirilir. Niceleme işleminde niceleme fonksiyonu haricinde, ölçek çarpanı kodlamasında elde edilen gürültü biçimlendirme işlemi de önemli rol oynar. AAC'de kullanılan niceleyici MPEG 1/2 Layer III'te kullanılan niceleyiciye benzer ve onun gibi lineer olmayan bir karakteristiğe sahiptir. Niceleme adım büyüklüğü bit katarında yer alan özel bir elaman ile belirlenir. Adım büyüklüğü 1.5dB aralıklar ile ayarlanabilir.

Lineer olmayan niceleyicinin doğasında bir gürültü biçimlendirme işlemi vardır, ama bu işlem genellikle arzu edilen ses kalitesi için yeterli değildir. Kodlanan sinyalin kalitesini arttırmak için gürültü aynı zamanda ölçek çarpanları ile de biçimlendirilir. Ölçek çarpanları, sinyali *ölçek çarpanı bandı* denilen belirli spektral bölgelere genişletmek için kullanılırlar. Bu sayede, bu bantlardaki SNR (signal-to-noise ratio) arttırılmış olur. Yüksek spektral değerler genellikle sonradan daha fazla bite ihtiyaç duyacakları için, frekans üzerindeki bit tahsisi de dolaylı yoldan değiştirilmiş olur. Niceleyicide olduğu gibi ölçek çarpanlarının adım büyüklüğü de 1.5dB'dir.

Orijinal spektral değerlerin kod çözücüde düzgün olarak tekrar elde edilebilmesi için, ölçek çarpanları da bit katarı içinde aktarılmalıdır. Ölçek çarpanları bir ölçek çarpanı bandından diğerine çok fazla değişmedikleri için, önce fark kodlaması ardından Huffman kodlaması kullanılarak bit katarı içinde sıkıştırılmış biçimde saklanırlar.

AAC'de yer alan gürültüsüz kodlama çekirdeğinin görevi, spektral veri kodlaması içindeki artık değerlerin azaltılması işini en iyi şekilde yapmaktır. İzgel veri, maksimum nicelenmiş değere bağlı olarak uygun Huffman kod tablosu ile kodlanır. Eğer ilgili ölçek çarpan bandındaki tüm katsayılar 0 ise, ne katsayılar ne de ölçek çarpanı iletilmez. Seçilen tablonun iletilmesi gerekliliği önemli ölçüde ek bilgi aktarımına neden olur. Ek bilgi aktarımının minimumda tutulması ile her ölçek çarpan

bandı için en uygun tablonun seçilmesi arasındaki ilişki, spektral veriye uygulanan etkili bir gruplama algoritmasıyla en uygun şekilde sağlanır.

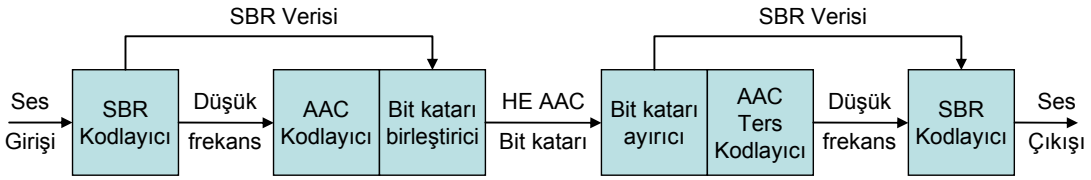
MPEG-4 AAC'de yer alan Twin-VQ tabanlı kodlama araçları, düşük bit oranlarında müzikal sinyallerin etkili şekilde kodlanmasına yardım eder [Iwakami, 1996]. Temel mantık, MPEG-4 AAC'de kullanılan spektral verilerin ve ölçek çarpanlarının geleneksel kodlamasını, normalleştirilmiş spektruma vektör niceleme uygulanması yöntemi ile değiştirmektir. İşlem zincirinin geri kalan kısmı Şekil 4.8'de görüldüğü gibi aynıdır. Twin VQ yüksek kodlama verimliliğine sahip olsa da, ses kalitesinde küçük bir miktar kayba neden olduğu için, genellikle bit oranı kanal başına 16 kbit/s değerinin altında olan sinyallerde kullanılır.

MPEG-4 HE-AAC (High Efficiency AAC)

AAC LC profili ile SBR (Spectral Band Replication) bant genişliği arttırma aracının birleştirilmesi ile oluşan HE profili, 48 kbit/s bit oranında bile iyi stereo ses kalitesi sunar. 2003 Mart ayında yapılan MPEG toplantısında son hali verilmiştir. Çoklu kanal işlemlerinde de kullanılabilen HE-AAC, aacPlus adıyla da bilinir. 2004 yılında parametrik stereo özelliğinin de eklenmesi ile oluşturulan ikinci sürümü (HE-AAC v2) sayesinde sıkıştırma etkinliği daha da arttırılmıştır [Meltzer, 2006].

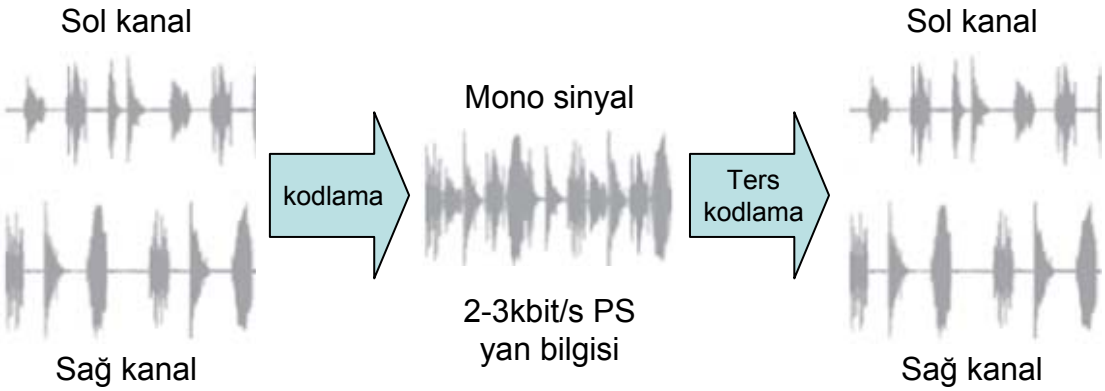
SBR, yüksek frekansları daha düşük miktarda veri ile temsil ederek, düşük ve yüksek frekanslar arasındaki korelasyonu azaltır. SBR ile temsil edilen yüksek frekans verileri, daha sonra AAC kodlayıcısı tarafından sıkıştırılmış olan düşük frekans verileri ile birleştirilir. Örneğin HE-AAC ile 48 kbit/s bit oranında stereo kodlama yaptığımızda, bunun 42 kbit/s büyüklüğündeki kısmı AAC ile kodlanırken, yüksek frekansların yer aldığı 6 kbit/s büyüklüğündeki kısım ise SBR ile kodlanır. Bu sinyaller Şekil 4.13'te görüldüğü gibi bit katarının oluşturulması aşamasında birleştirilir.

HE-AAC, AAC'ye göre geriye doğru tam uyumlu, ileri doğru ise kısmen uyumludur. Bu da demektir ki; AAC ile kodlanmış sesleri HE-AAC tam olarak çözebilirken, HE-AAC ile kodlanmış seslerin sadece AAC kodlayıcısı ile kodlanmış olan düşük frekanslı bölümleri AAC ile çözülebilir.



Şekil 4.13. MPEG-4 HE-AAC yapısı ile SBR ve AAC'nin birleştirilmesi

SBR yüksek frekanslı bölümleri parametrik hale getirip sıkıştırma oranını arttırırken, parametrik stereo (PS) ise panorama, ambiyans ve zaman/faz farkları gibi özellikleri içeren stereo ses sinyallerini parametrik hale getirir. Örneğin sol ve sağ ses kanallarında sesler farklı zamanlarda oluşuyorsa, PS bu iki kanalı Şekil 4.14'te görüldüğü gibi üst üste bindirip mono sinyal haline getirebilir ve hangi aralığın hangi kanala ait olduğunu da az miktarda yan bilgi ile kodlayabilir.



Şekil 4.14. Parametrik stereo kodlaması

4.2.3. MPEG Ses Kodlayıcılarının Özelliklerinin Karşılaştırılması

AAC, MP2 ve MP3 formatlarına göre daha çok kanal desteği ve daha yüksek örnekleme frekansı sağlar. Çizelge 4.3'te bu yöntemlerin destekledikleri kanal sayısı, bit oranları ve örnekleme frekansları ve yayınlandıkları tarihler verilmiştir.

Ses kalitesi algıya dayalı bir durum olduğu için, ses sıkıştırma yöntemlerinin verimliliği genellikle dinleme testleri yapılarak belirlenir. Resmi MPEG dinleme testleri, AAC ile 96 kbit/s bit oranında kodlanmış sesin, MP3 ile 128 kbit/s ve MP2 ile 192 kbit/s bit oranında kodlanmış sesin kalitesinden daha iyi olduğunu göstermiştir.

Çizelge 4.3. ISO/MPEG Ses Kodlayıcılarının Özellikleri

Standart	Ses Örnekleme Oranı (kHz)	Sıkıştırılmış bit oranı (kbit/s)	Kanal Desteği	Yayınlandığı Tarih
MPEG-1 Layer I	32, 44.1, 48	32 – 448	1 – 2	1992
MPEG-1 Layer II	32, 44.1, 48	32 – 384	1 – 2	1992
MPEG-1 Layer III	32, 44.1, 48	32 – 320	1 – 2	1993
MPEG-2 Layer I	32, 44.1, 48	32 – 448 (iki kanal için)	1 – 5.1	1994
	16, 22.05, 24	32 – 256 (iki kanal için)		
MPEG-2 Layer II	32, 44.1, 48	32 – 384 (iki kanal için)	1 – 5.1	1994
	16, 22.05, 24	8 – 160 (iki kanal için)		
MPEG-2 Layer III	32, 44.1, 48	32 – 384 (iki kanal için)	1 – 5.1	1994
	16, 22.05, 24	8 – 160 (iki kanal için)		
MPEG-2 AAC	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48, 64, 88.2, 96	23-bit işaretsiz tamsayı ile gösterilir	1 – 48	1997
MPEG-4 AAC	8, 11.025, 12, 16, 22.05, 24, 32, 44.1, 48, 64, 88.2, 96	23-bit işaretsiz tamsayı ile gösterilir	1 – 48	1999

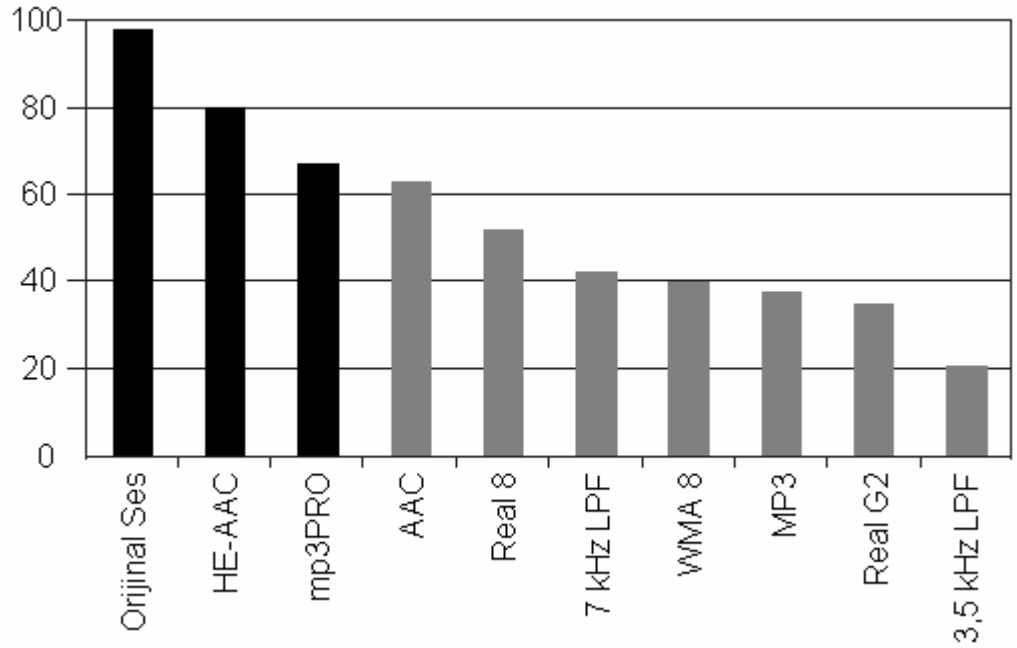
4.2.4. WMA (Windows Media Audio)

Microsoft tarafından geliştirilmiş bir standart olan WMA'nın yapısı hakkında çok fazla kaynak bulunmamaktadır. WMA 9 sürümü temel kodlayıcı haricinde 3 farklı kodlayıcı daha içermektedir; Pro, Lossless ve Voice. AAC ile rekabet edebilecek düzeyde geliştirilen Pro kodlayıcı, temel kodlayıcıya göre çok daha fazla özelliğe sahiptir. Örneğin temel kodlayıcı en fazla 48 kHz frekansı, 16 bit örneklemeyi, 192 kbit/s veri oranını ve 2 ses kanalını desteklerken, Pro kodlayıcı 96 kHz frekansı, 24 bit örneklemeyi, 768 kbit/s veri oranını ve 5.1 ve 7.1 kanal sayısını (surround sound) destekleyebilmektedir. Kayıpsız (Lossless) kodlayıcı orijinal sesi 1/2- 1/3 oranında sıkıştırabilmektedir. Konuşma kodlaması için tasarlanmış olan düşük bit oranlı Voice kodlayıcı ise, karma kip desteği sayesinde istenirse müzik türü sesleri de temel kodlayıcı kullanarak sıkıştırma özelliğine sahiptir. Karma kipte, sesin hangi kısımlarının

konuşma, hangi kısımlarının müzik kodlayıcısı ile kodlanacağı otomatik olarak belirlenir.

4.2.5. Kayıplı Ses Sıkıştırma Yöntemlerinin Karşılaştırılması

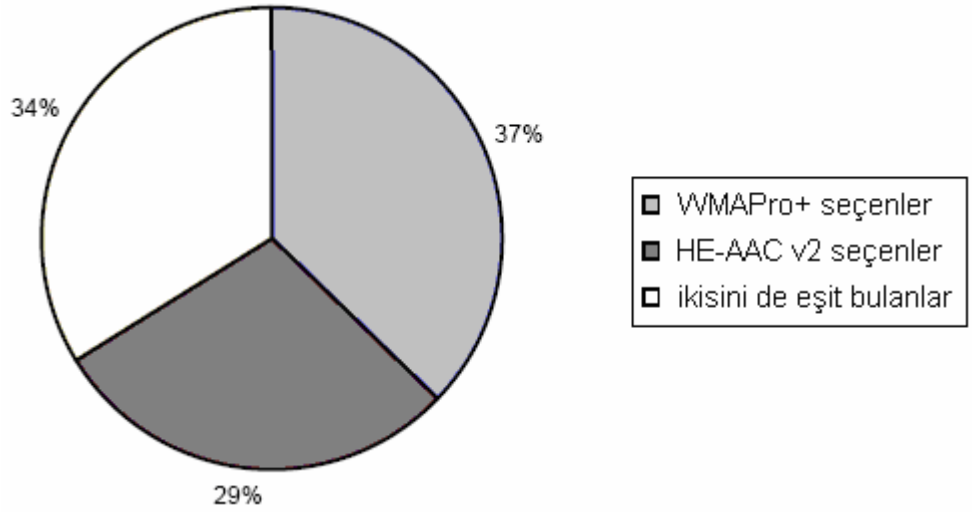
EBU (European Broadcasting Union), MPEG (Moving Pictures Expert Group) ve 3GPP (3rd Generation Partnership Project) gibi birçok kuruluş tarafından kapsamlı dinleme testleri yapılmaktadır. Şekil 4.15'te, 2003 yılında EBU tarafından yapılan kayıplı ses kodlayıcıların 48 kbit/s veri oranında stereo kodlama performansları ile ilgili dinleme testinin sonuçları verilmiştir. Bu dinleme testi dinleyen kişilerin hangi yöntem ile sıkıştırıldığını bilmeden puan verdikleri MUSHRA (MUltiple Stimulus test with Hidden Reference and Anchors) yöntemine uygun olarak yapılmıştır.



Şekil 4.15. EBU öznel dinleme testi sonuçları (48 kbit/s, stereo)

Bu testte HE-AAC yönteminden sonra en iyi ikinci sonucu elde eden mp3PRO yöntemi de, MP3 ile SBR'nin birleştirilmesiyle oluşturulmuş bir yöntemdir. 2003 yılında henüz WMA'nın 9 sürümü tamamlanmamış olduğu için bu testte 8 sürümü yer almıştır. Testte yer alan 3,5 kHz LPF ve 7 kHz LPF alçak geçiren filtrelerdir.

Microsoft firması WMA'nın 9 sürümünü tamamladıktan sonra, WMA 9 Pro kodlayıcısı ile HE-AAC kodlayıcısının performansını karşılaştırmak amacıyla NTSL (National Software Testing Labs) kuruluşuna bir test yaptırmıştır. 2005 yılının Aralık ayında yayınlanan bu dinleme testinin sonuçları Şekil 4.16'da verilmiştir. 300 farklı dinleyici kullanılarak 64 kbit/s veri oranında yapılan testin ortalaması alındığında, dinleyicilerin %37'si WMAPro'yu, %29'u ise HE-AAC'yi daha iyi bulmuşlardır. Fakat testte kullanılan 12 adet ses dosyası da Microsoft tarafından seçildiği için bu testin tarafsız bir test olduğunu söylemek güçtür.



Şekil 4.16. NTSL dinleme testi sonuçları (64 kbit/s, stereo)

5. GÖRÜNTÜ SIKIŞTIRMA

Görüntü sıkıştırma, kayıpsız ve kayıplı olmak üzere iki şekilde yapılabilir. Kayıpsız sıkıştırma genellikle pikseller arası korelasyonun düşük olduğu şekil gibi görüntülerde, kayıplı sıkıştırma ise fotoğraf görüntüleri gibi karmaşık ve yüksek korelasyona sahip görüntülerde kullanılır.

5.1. Kayıpsız Görüntü Sıkıştırma Yöntemleri

Kayıpsız görüntü sıkıştırma, sadece olasılık tabanlı ya da sözlük tabanlı bir yöntem ile yapılabileceği gibi, ikisini bir arada kullanan bir yöntem ile de yapılabilir. Örneğin çok popüler olan PNG görüntü sıkıştırma yöntemi, LZ77 ve Huffman sıkıştırma algoritmalarını bir arada kullanan, DEFLATE sıkıştırma yöntemini kullanır. JPEG-LS, JBIG gibi özellikle kayıpsız görüntü sıkıştırmak için tasarlanmış yöntemler de vardır.

5.1.1. RLE (Run Length Encoding)

Her tür veri için kullanılabilir bir algoritma olsa da, aynı sembolün ardışık olarak çok defa tekrar etmesi durumunda iyi bir sıkıştırma oranı sağladığı için, genellikle görüntü sıkıştırmada kullanılır. BMP, PCX ve TIFF görüntü dosya formatları, RLE ile sıkıştırma yapabilir.

RLE'nin temel mantığı, bir değerin ardışık olarak çok sayıda tekrar etmesi durumunda, o değeri bir defa kodlayıp, ardından tekrar sayısını kodlamaktır [Capon, 1959]. Örneğin 1-bit renk derinliğine sahip bir görüntünün (2 renge sahip: sadece siyah ve beyaz) bir satırındaki 640 pikselin renkleri sırayla; önce 200 adet beyaz, sonra 240 adet siyah ve sonra yine 200 adet beyaz şeklinde ise, o satır RLE ile (200, 240, 200) şeklinde kodlanabilir. Burada açıcının (decoder) ilk pikselin ne renk olduğunu bilmesi gerekir. Ya ilk pikselin rengi kodlamanın başında belirtilir, ya da ilk pikselin her zaman beyaz olduğu farz edilir. Bu durumda ilk 200 piksel siyah olsaydı önce 0 sonra 200 kodlanacaktı.

Fax makinelerinde kullanılan CCITT T4 (Group 3) standardı, RLE ve Huffman kodlamasını bir arada kullanan bir standarttır. Bu standartta ilk piksel her zaman beyaz kabul edilir. Tekrar sayıları çok fazla farklı sayıda olasılığa sahip olabilecekleri için, 64'ten büyük olan tekrar sayıları önce 64'e bölünür, sonra bölüm ve kalan farklı Huffman Tabloları kullanılarak kodlanır. Bu kodlama *modified Huffman* (MH) ya da *Group 3 One-Dimensional* (G31D) olarak bilinir.

CCITT T4 standardı, IBM tarafından 1979 yılında geliştirilen MMR (Modified Modified READ) sıkıştırmasını da kullanabilir. *Group 3 Two-Dimensional* (G32D) olarak ta bilinen bu yöntemle bir önceki satır da dikkate alınarak daha yüksek sıkıştırma oranlarına ulaşılmaktadır. Fakat MMR'nin sıkıştırma performansı da, daha sonra geliştirilen JBIG standardının gerisinde kalmıştır.

5.1.2. JPEG-LS

ISO/IEC tarafından geliştirilen, görüntüleri kayıpsız veya az kayıplı sıkıştırabilen JPEG-LS, 1998 yılının sonlarında tamamlanmıştır. Standardın temelini "Hewlett-Packard Laboratories" tarafından geliştirilen LOCO-I (LOw COmplexity LOssless COmpression for Images) algoritması oluşturmaktadır [Weinberger, 1998].

Hem renkli hem de gri-tonlamalı görüntülerin sıkıştırılmasında kullanılan JPEG-LS, Bölüm 2.5.2'de anlatılan artık değer kodlaması tabanlıdır. Burada sadece gri-tonlamalı görüntülerin sıkıştırılmasında nasıl kullanıldığından bahsedilecektir. Renkli görüntülerde ise kırmızı, yeşil ve mavi renk değerleri için aynı mantığın ayrı ayrı kullanılması ile sıkıştırma yapılabilir.

Kodlayıcı görüntünün sol üst pikselinden başlayarak sağ alt pikseline kadar satır tarama düzeninde işlem yapar. Bundan dolayı, bir pikselin değeri tahmin edilirken kodlayıcı ve çözücüde o pikselin üstündeki ve solundaki piksellerin değerleri mevcuttur. JPEG-LS algoritması bu değerlerden sadece 4 tanesini kullanır; soldaki (W), üst soldaki (NW), üstteki (N) ve üst sağdaki (NE). Tahmin iki aşamada gerçekleşir. İlk aşamada her piksel değeri için aşağıdaki gibi tahmin yapılır:

$$G = \begin{cases} \mathbf{min}(W, N) & \max(N, W) \le NW \\ \mathbf{max}(W, N) & \min(N, W) < NW \\ W + N - NW & \text{diğer durumlarda} \end{cases}$$

Bu denklem aslında görüntüdeki kenarları tarif ederken komşu piksellerin ortalama değerleri ile ilgilenme mantığına dayanır. İlk iki madde dikey ve yatay kenarları yakalar. Örneğin, eğer $N > W$ ise ve $N \le NW$ ise, bu yatay kenarı işaret eder ve W tahmin olarak kullanılır. Son madde ise yatay kenarları yakalar.

Elde edilen G tahmininden sonra ikinci bir geçiş ile, yapılan bu tahminin çevresindeki piksel çiftlerinin değerlerinin farklarına göre düzeltmesi yapılır. Üç piksel çifti kullanılır, bunlar; (*NW*, *W*), (*NW*, *N*) ve (*N*, *NE*). İki bitişik pikselin değerlerinin farkına göre, her biri 9 grubun birine sınıflandırılır. Bu da toplamda 729 ($=9^3$) farklı durumu ortaya çıkarır. Simetri nedeniyle bunlardan sadece 365 tanesi gereklidir. Her farklı durum, tahmini düzeltmek için kullanılan kendine ait düzeltme değerini ve o durum için daha önceki tahminlerin kaliteleri ile ilgili bilgiyi saklar. Algoritma ikinci geçişten sonra sonuç tahmini üretir ve gerçek değer ile arasındaki farkı (artık değeri) kodlar.

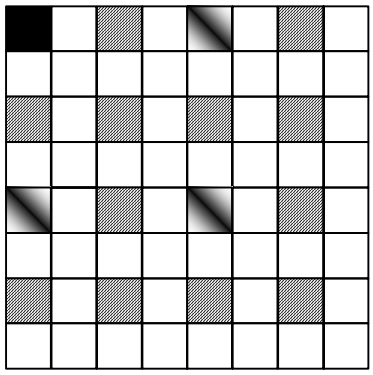
5.1.3. JBIG Standardı

JBIG (Joint Bi-level Image Experts Group), 1988'de ISO/IEC JTC1/SC29/WG1 grubu ile CCITT SGVIII grubunun birleşmesi ile oluşturulmuştur. 1-bit derinlikli (bi-level) görüntü verisini kayıpsız olarak sıkıştırmak için kullanılabilecek bir yöntem bulmak amacıyla kurulan grup, 1993'te JBIG standardını tamamlamış ve yayınlamıştır [ISO/IEC, 1993]. Bu tarihten sonra JBIG standardı MR ve MMR'nin yerini alarak, faks iletiminde yaygın olarak kullanılan bir standart haline gelmiştir.

JBIG, aritmetik kodlama temelli bir standarttır. JBIG standardının temelinde yatan ve IBM ile Mitsubishi şirketlerinin patentini elinde bulundurduğu QM kodlayıcısı, Q kodlayıcısı [Pennebaker vd., 1988] olarak adlandırılan uyarlanabilir ikili aritmetik kodlayıcının değiştirilmiş bir halidir. QM kodlayıcısında işlem etkinliğini arttırmak için aritmetik kodlamada gerekli olan çarpma ve bölme işlemlerinden uzak durulmuştur.

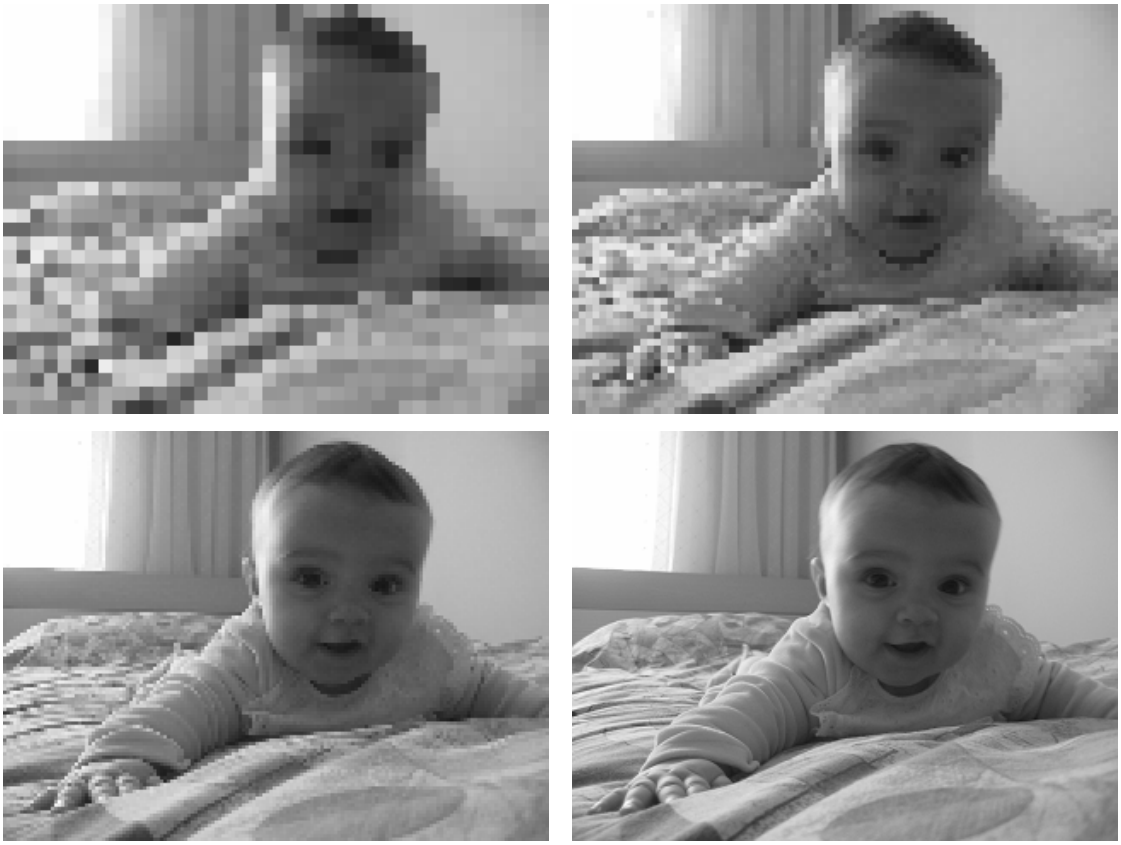
JBIG de JPEG-LS gibi kodlanacak pikselin değerini bulmak için çevresindeki piksellerin değerlerini kullanır. Fakat, JPEG-LS'den farklı olarak, JBIG şarta bağlı olasılıkları direkt olarak kullanır. JBIG aynı zamanda kademeli aktarıma da (progressive transmission) izin verir. Ama istenirse sol üst pikselden başlayarak sağ alt piksele kadar satır satır tüm piksellerin aktarılması biçimindeki sıralı aktarım da uygulanabilir.

Kademeli aktarımda görüntünün önce düşük çözünürlüklü bir gösterimi iletilir. Bu iş için genellikle kullanılan yöntem; görüntüyü bloklara ayırmak ve her bir bloğu temsil etmek için o bloktaki bir pikselin değerini göndermektir. Örneğin 800x600 piksel büyüklüğünde bir görüntüyü 8x8 piksel boyutunda bloklara ayırıp, her bir bloğu temsil etmek için o bloğun sol üst köşesindeki pikselin değerini gönderebiliriz. Sonuçta alıcı tarafa 100x75 piksel boyutunda, orijinal görüntünün hem en hem de boydan 1/8'i büyüklüğünde düşük çözünürlüklü bir kopyası iletilmiş olur. Alıcı taraf her bir pikseli 63 kere tekrar ederek 8x8 boyutuna genişletebilir. Bu sayede ilk etapta görüntünün sadece 64'te biri gönderilmiştir. Görüntünün düşük çözünürlüklü bu gösteriminden alıcı istediği görüntü olup olmadığına karar verebilir. İstediği görüntü değilse devamının gelmesini beklemeyebilir. İkinci kademede 4x4 ve üçüncü kademede 2x2 büyüklüğünde bloklar kullanılabilir. Her bir kademede bir önceki kademenin blokları dörde bölünmüş olur. Bu dört yeni bloktan sol üst bloğun sol üst pikselinin değeri bir önceki kademede gönderilmiş olduğu için, sadece diğer üç bloğun sol üst piksellerinin değerlerini göndermek yeterlidir. Bir örnek ile açıklayacak olursak; Şekil 5.1'de ■ ile gösterilen piksel ilk kademede, ▚ ile gösterilen 3 piksel ikinci kademede, ▨ ile gösterilen 12 piksel üçüncü kademede ve diğer 48 piksel ise dördüncü ve sonuncu kademede gönderilirler.



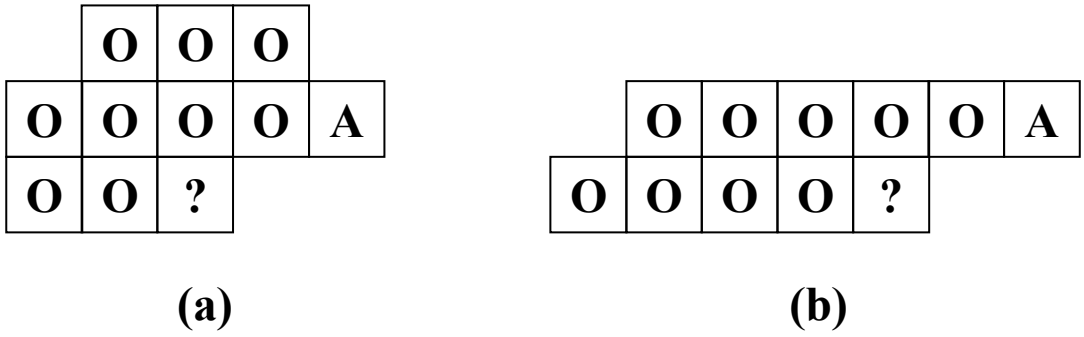
Şekil 5.1. Kademeli aktarımda değerleri gönderilen pikseller

Şekil 5.2'de dört kademelik bir aktarımın örneği yer almaktadır. 320x240 boyutunda 76.800 piksellik bir görüntünün, ilk kademede 40x30 boyutunda düşük çözünürlüklü bir görünümü (sol üst resim) için 1.200 piksel, ikinci kademede 80x60 boyutunda 4.800 piksellik daha yüksek çözünürlüklü bir görünümü (sağ üst resim) için ilave olarak 3.600 piksel daha, üçüncü kademede 160x120 boyutunda 19.200 piksellik daha da yüksek çözünürlüklü bir görünümü (sol alt resim) için 14.400 piksel daha, ve en sonunda da görüntünün tamamı (sağ alt resim) için 57.600 piksel daha aktarılmıştır.

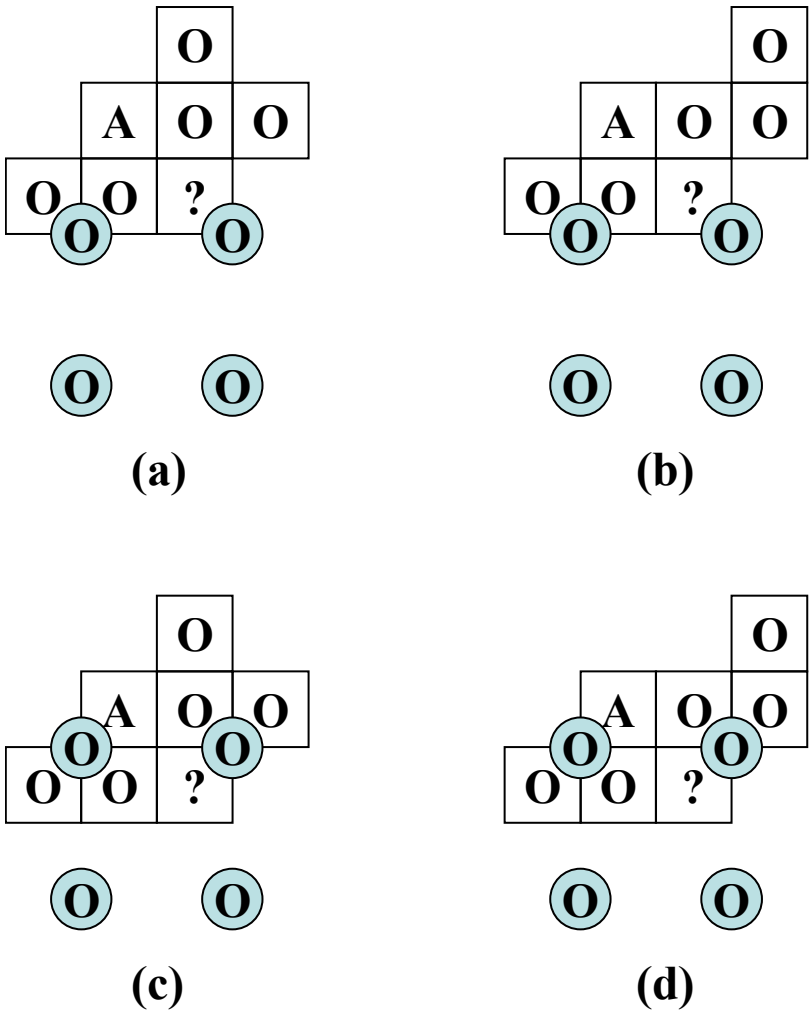


Şekil 5.2. Dört kademeli bir aktarım örneği

Sıralı aktarımda ve kademeli aktarımda uygulanan sıkıştırma prensibi aynıdır. JPEG-LS'de kullanılan 4 komşu piksel yerine, JBIG'de hepsi komşu olmayan 10 piksel kullanılır. Sıralı aktarımda ve kademeli aktarımın ilk kademesinde Şekil 5.3'te gösterilen iki farklı şablondan biri kullanılır. Kademeli aktarımın diğer kademelerinde ise Şekil 5.4'te gösterilen 4 farklı şablondan biri kullanılır.



Şekil 5.3. JBIG şablonları: (a) üç satırlı şablon, ve (b) iki satırlı şablon



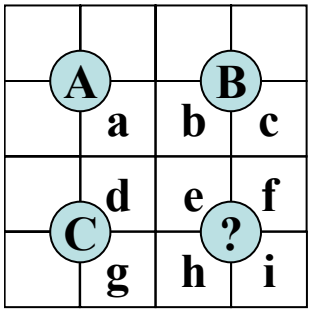
Şekil 5.4. Kademeli aktarım için JBIG şablonları

Şekil 5.3 ve Şekil 5.4'te '?' ile gösterilen piksel kodlanacak olan pikseldir. 'O' ve 'A' ile gösterilen pikseller ise daha önceden kodlanmış olan ve hem kodlayıcıda hem

de çözücüde bulunan piksellerdir. ‘A’ seyyar bir pikseldir ve bulunacağı yer kodlanan içeriğe göre değişebilir. Örneğin 30 piksel mesafe aralığı ile tekrarlanan dikey çizgilerin bulunduğu bir görüntünün sıkıştırılmasında ‘A’ pikseli kodlanacak olan piksele 30 birim uzaklıkta bile olabilecektir. Kodlayıcı iki şablondan hangisinin kullanılacağına ve daha iyi bir sıkıştırma için A’nın nerede olması gerektiğine karar verecektir. Sıkıştırılan mesaj dizisinin başında bu bilgi yer alacaktır. Her piksel 1-bit (2 olasılıklı) bilgi sakladığı için 1024 ($=2^{10}$) adet farklı desen olması mümkündür. Algoritma siyah ve beyaz piksellerin bulunma olasılıklarını dinamik olarak değiştirerek, bu olasılıkları QM kodlayıcısında kullanır.

Kademeli aktarımda da 10 piksel kullanılır. Fakat bu defa, bunlardan 6 tanesi kodlanan kademeye, diğer 4 tanesi ise daha önce kodlanmış olan bir önceki kademeye aittir. Şekil 5.4’te bir önceki kademeye ait olan piksel değerleri yuvarlak içerisinde gösterilmiştir. Her yuvarlak, ortalarında durduğu 4 kareyi daha düşük çözünürlüğe sahip bir önceki kademede temsil etmiştir. JBIG’de kullanılan kademeli aktarımda, bloğun değerini bulmak için, sol üst pikselini ya da bloktaki tüm piksellerin değerlerinin ortalamasını almak yerine, Şekil 5.5’te görülen yapı üzerine aşağıdaki formül uygulanır.

$$\textit{Blok Değeri} = 4e + 2(b + d + f + h) + (a + c + g + i) - 3(B + C) - A$$



Şekil 5.5. Düşük çözünürlüklü pikselin değerini belirlemek için kullanılan pikseller

Bloğu temsil edecek pikselin değeri, eğer formülün sonucu 4.5’ten büyükse 1, küçükse 0 olacaktır. Eğer böyle bir formül uygulamak yerine 4 pikselin ortalaması alınsaydı, 2 siyah ve 2 beyaz piksel olması durumunda ortaya çıkacak 0.5 ortalamasının, siyahı mı yoksa beyazı mı temsil ettiğine karar vermede sorun yaşanacaktı. Bu gibi durumlarda bilgi kaybı oluşabileceği için böyle bir formüle gereksinim duyulmuştur.

Kademeli aktarımda 4 şablondan hangisinin kullanıldığının da 2 bit büyüklüğünde bir ön bilgi olarak eklenmesi gerekeceği için, pikseller için gereken 10 bit ile beraber toplam 12 bit'e ihtiyaç duyulacaktır. Bu da 4096 farklı durumun gerçekleşebileceğine işaret eder.

JBIG-2

ISO ve ITU'nun ortak geliştirdiği bir standarttır. ITU tarafında T.88 olarak, ISO tarafında ise 14492 kod numarası ile projelendirilen JBIG-2 standardı, 2000 yılında tamamlanmıştır [ITU, 2000 ve ISO, 2001]. Önceki JBIG standardı gibi, çoğunlukla faks iletiminde kullanılan 1-bit renk derinliğine sahip görüntüler için geliştirilmiş olan JBIG-2, kayıpsız sıkıştırma yapabildiği gibi kayıplı sıkıştırma da yapabilir.

JBIG-2 kodlayıcısı, bir sayfadaki metin türünde verinin her karakterini *sembol* olarak, resim türü verileri *desen* olarak, diğer verileri de *genel veri* olarak görür ve üç grubu da farklı kodlar. Genel veriler QM kodlayıcısına benzer olan ve yine onun gibi içeriğe bağımlı olan MQ kodlayıcısı ile kodlanır. Semboller her zaman bir karakteri temsil etmek zorunda değildir. Karaktere benzeyen başka ufak şekiller de sembol olarak değerlendirilebilir. Sembollerin bir sözlüğü oluşturularak aritmetik kodlama ile sıkıştırılır. Kayıplı sıkıştırma yapılırken birbirine benzeyen semboller arasındaki farklılık göz ardı edilerek, ikisi aynı sembol gibi düşünülür. Kayıpsız sıkıştırmada ise benzer olan sembollerden biri, diğer sembolün yaratılması için referans olarak kullanılır. Desen olarak düşünülen resimler ise yarım ton (halftone) desenlerinden oluşan bir sözlük kullanılarak sıkıştırılır.

5.2. Kayıpsız Sıkıştırma Kullanan Görüntü Dosya Formatları

5.2.1. GIF (Graphics Interchange Format)

1987'de CompuServe tarafından yaratılan GIF, LZW tabanlı çalışan algoritmalardan biridir. En fazla 8-bit renk derinliğine ($2^8 = 256$ renk) sahip görüntülere izin verir. Bu sebepten dolayı fotoğraf görüntülerinin sıkıştırılmasında yetersiz kalsa da, birkaç rengin çoğunlukta olduğu grafiksel gösterimler ve basit şekiller gibi görüntülerin

kayıpsız olarak sıkıştırılmasında halen kullanılmaktadır. Günümüzde daha çok son sürümü olan GIF 89a kullanılmaktadır [CompuServe, 1990].

GIF formatının temel özellikleri şunlardır:

- Kademeli (progressive) aktarımı destekler.
- Şeffaflık: görüntünün bazı bölgeleri şeffaf olarak işaretlenebilir. Bu sayede dikdörtgensel olmayan görüntü etkisi yaratılır.
- Yardımcı bilgi: Metinsel bilgiler ve başka veriler görüntü dosyasının içinde saklanabilir.
- Donanımdan ve ortamdan tam bağımsızlık sağlar.

5.2.2. TIFF (Tagged Image File Format)

Aldus Corporation tarafından geliştirilmiştir. Aldus'un daha sonra Adobe Systems ile birleşmesi nedeniyle şu anda telif haklarına Adobe Systems şirketi sahiptir. 1988'de yayınlanan 5.0 sürümü ile RGB bitmap verilerini LZW algoritması kullanarak sıkıştırabilme yeteneğine kavuşmuştur. 1992'de yayınlanan 6.0 sürümü [Aldus Developers Desk, 1992] ile de JPEG sıkıştırması eklenmiştir.

TIFF-LZW'nin de GIF gibi patent sorunlarına yol açması sonucunda bazı girişimciler tarafından TIFF altında ZIP sıkıştırması geliştirilmiştir. Bu yapı da PNG gibi Deflate algoritması ile sıkıştırma yapar.

TIFF formatının GIF'ten temel farkı; hem 256 sıralı-renk görüntülerle, hem de 24-bit RGB (8-bit Red + 8-bit Green + 8-bit Blue) ile çalışabilmesidir. 24-bit renk derinliğini desteklemesi sayesinde fotoğrafların kayıpsız olarak sıkıştırılmasında kullanılır. Fax iletiminde kullanılan "TIFF Compression Type 2" ise CCITT Grup 3 (G31D) tabanlıdır.

5.2.3. PNG (Portable Network Graphics)

GIF ve altyapısındaki LZW algoritmalarının, patent anlaşmaları ile koruma altında olmaları, ücretsiz olarak kullanılabilecek bir kayıpsız görüntü sıkıştırma standardı ihtiyacını gündeme getirmiştir. 1995 yılında Thomas Boutell, Scott Elliott,

Mark Adler, Tom Lane, ve birçok başka girişimcinin başlattığı PNG projesi 1996'da sonuçlanmıştır. 1999'da yayınlanan 1.2 sürümünden sonra ile birlikte kullanım oranı hızla artmıştır [Randers-Pehrson, 1999].

48-bit gerçek renk, 16-bit gri tonlama desteği, ve yüksek oranda sıkıştırma becerisi sayesinde GIF ve TIFF standartlarından daha iyi olduğunu ispatlamıştır. Altyapısında DEFLATE veri sıkıştırma yöntemi yer almaktadır.

PNG formatının temel özellikleri şunlardır:

- Basit ve taşınabilir yapı: Geliştiriciler PNG'yi kolayca uygulayabilirler.
- Patent koruması altında değildir.
- Hem sıralı-renk hem de gerçek-renk görüntülerde diğer algoritmalar kadar etkili (hatta çoğu durumda daha etkili) sıkıştırma oranı sağlar.
- Esneklik: Geliştirmelere ve eklentilere izin veren bir yapısı vardır.

PNG'nin GIF ve TIFF'e göre üstün olduğu noktalar şunlardır:

- Piksel başına 48-bit renk derinliğine kadar renkli ve 16-bit renk derinliğine kadar gri-tonlamalı görüntülerle çalışabilme yeteneği vardır.
- Tam alfa kanalı: Genel şeffaflık maskeleri vardır.
- Görüntünün gamma bilgisi: Görüntülerin doğru bir parlaklık/karşıtlık (contrast) ile otomatik olarak gösterilmesini sağlar.
- Dosya hatalarında güvenilir tespit yapabilme yeteneğine sahiptir.
- Kademeli aktarımda daha hızlı ön gösterim yapabilme yeteneğine sahiptir.

5.3. Kayıpsız Görüntü Sıkıştırma Yöntemlerinin Karşılaştırılması

Bu çalışmada, görüntüleri kayıpsız olarak sıkıştırmak için günümüzde en sık kullanılan yöntemleri, birbirinden farklı özelliklere sahip görüntü dosyalarından oluşan bir veri seti üzerinde test edip, hangi yöntemin hangi tip görüntü için daha kullanışlı olduğunu belirlemeyi amaçladık. Ayrıca, kendi geliştirdiğimiz ISSDC algoritmamızı da bu teste dâhil ederek, popüler görüntü sıkıştırma teknikleri ile kıyaslandığında ne oranda başarılı olduğunu gözlemledik.

Karşılaştırmaya dâhil edilen yöntemler aşağıda verilmiştir:

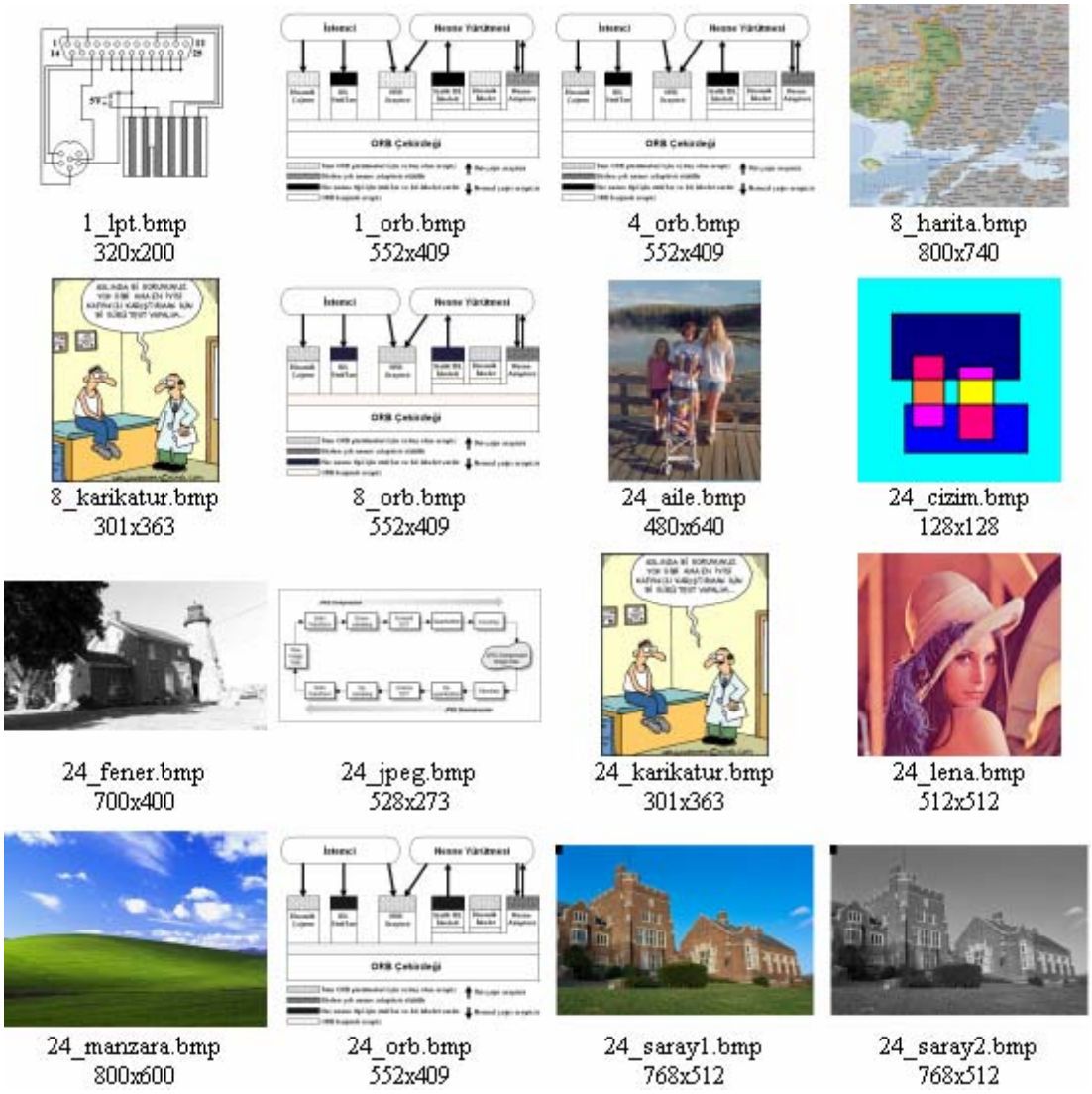
- LZW tabanlı sıkıştırma kullanan GIF görüntü sıkıştırma formatı
- RLE, LZW, LZW+Predict, ZIP, JPEG sıkıştırma yöntemlerini kullanabilen TIFF görüntü sıkıştırma formatı
- Deflate tabanlı PNG görüntü sıkıştırma formatı
- Digram Kodlaması tabanlı ISSDC
- LOCO-I tabanlı JPEG-LS (JPEG-Lossless)
- Kayıpsız JPEG2000 sıkıştırması
- WinZIP ve WinRAR sıkıştırma paketleri

Karşılaştırmada sıkıştırılacak olan dosyalar Windows Bitmap (BMP) formatında seçilmiştir. Bunun sebebi hem bu formatın çok yaygın olarak kullanılması, hem de 1, 4, 8 ve 24 bit renk derinliklerini desteklemesidir.

Karşılaştırmada kullanılan görüntü dosyaları Şekil 5.6'da gösterilmiştir. Her görüntünün altında o görüntünün ait olduğu dosyanın ismi ve (genişlik) x (yükseklik) piksel değerleri yer almaktadır. Dosyaların isimlerinin başında yer alan numaralar, o dosyada 1 pikseli ifade etmek için kaç bit kullandığını (renk derinliğini) gösterir. Kullanılan 16 adet dosyadan iki tanesi 1-bit (1_lpt, 1_orb), bir tanesi 4-bit (4_orb), üç tanesi 8-bit (8_orb, 8_harita, 8_karikatur) ve geri kalan on dosya ise 24-bit renk derinliğindedir. Fakat 24-bit renk derinliğine sahip olan dosyaların hepsi fotoğraf değildir. 24_cizim, 24_orb, 24_karikatur ve 24_jpeg dosyaları, düz bir fon üzerine şekil ihtiva eden, az sayıda renk içeren dosyalardır. Örneğin 24_cizim dosyası, kullanabileceği 16.7 milyon rengin sadece 8 tanesini kullanmaktadır. 24_fener ve 24_saray2 dosyaları ise gri-tonlamalı fotoğraf dosyalarıdır.

Sonuçlar kısmında yer alan çizelgelerde verilen renk sayıları, o dosyanın renk derinliğini yada içerebileceği maksimum renk sayısını değil, o dosyada kullanılan ve birbirinden farklı olan tüm renklerin sayısını, yani tekil renk sayısını gösterir. Bu renk sayılarının bulunmasında ve PNG, TIFF ve GIF formatlarında sıkıştırma yapılmasında XnView v1.80.1 programından yararlanılmıştır. Kayıpsız JPEG2000 sıkıştırmasında

hem XnView hem de Jasper programları kullanılmış ve birbirine yakın sonuçlar ürettikleri görülmüştür. JPEG-LS/LOCO-I sıkıştırmasında ise Hewlett-Packard tarafından geliştirilen LOCO Encoder "locoe.exe" kullanılmıştır. ZIP ve RAR sıkıştırmaları için WinRAR 3.30, ISSDC için ise tarafımızdan geliştirilen sıkıştırma programı kullanılmıştır.



Şekil 5.6. Karşılaştırmada Kullanılan Görüntü Dosyaları

5.3.1. Sonuçlar

Karşılaştırmada sıkıştırma zamanları göz önüne alınmamış, sadece sıkıştırma oranlarına bakılmıştır. Test yapılırken ISSDC, JPEG2000 ve WinRAR yöntemlerinin diğerlerine göre daha yavaş sıkıştırma yaptıkları gözlenmiştir.

Karşılaştırma için 16 farklı görüntü dosyası, renk sayıları ve renk derinliklerine göre 5 farklı gruba ayrılmıştır. Belirli özelliklere sahip görüntü dosyalarının, belirli yöntemlerle daha iyi sıkıştırıldığını daha rahat gösterebilmek için böyle bir gruplamaya ihtiyaç duyulmuştur.

İlk grupta 1-bit ve 4-bit renk derinliğine sahip olan 3 adet dosya yer almaktadır. Bu 3 dosyanın karşılaştırmaya dâhil edilen tüm yöntemler ile sıkıştırılması sonucunda oluşan sıkıştırılmış boyutlar, byte cinsinden Çizelge 5.1'de verilmiştir. Bu çizelgede ve diğer çizelgelerde, sıkıştırma yerine şişirme yapan yöntemler altı çizili olarak gösterilmiştir. Her 3 dosya için ve bu dosyaların toplamı için, en iyi 3 sıkıştırma oranı koyu griden açık griye doğru renklendirilerek ifade edilmiştir.

Çizelge 5.1. 1-bit ve 4-bit renk derinliğindeki dosyaların sıkıştırma sonuçları

Dosya Adı		1_lpt	1_orb	4_orb	*TOPLAM*
Tekil Renk Sayısı		2	2	4	
Dosya Boyutu (byte)		8.062	29.510	113.002	*150.574*
Sıkıştırılmış Boyutlar (byte)	**RAR**	**717**	**3.302**	**4.788**	***8.807***
	ZIP	**773**	**3.585**	**5.407**	***9.765***
	PNG	**815**	**3.636**	**5.614**	***10.065***
	TIFF-ZIP	972	3.928	7.225	*12.125*
	ISSDC	1.125	4.873	6.587	*12.585*
	TIFF-LZW	1.916	6.340	11.085	*19.341*
	TIFF-LZWP	1.916	6.340	11.085	*19.341*
	GIF	3.016	7.676	9.890	*20.582*
	TIFF-RLE	6.982	15.386	28.614	*50.982*
	JPEG2000	13.235	51.830	54.294	*119.359*
	TIFF-JPEG	8.242	28.495	113.354	*150.091*

Çizelge 5.1'de görüldüğü gibi, tüm dosyalarda ve toplamda en iyi sonucu RAR, ikinci en iyi sonucu ZIP ve üçüncü en iyi sonucu da PNG yöntemleri sağlamıştır. ISSDC, 1-bit renk derinliğinde TIFF-ZIP'in gerisinde kalmış, 4-bit renk derinliğinde ise daha başarılı olmuştur. Bu renk derinliklerinde TIFF-LZW ve TIFF-LZWP'nin aynı sonuçları verdiği görülmüştür. 8-bit tabanlı çalışan GIF yöntemi ve 24-bit tabanlı çalışan JPEG yöntemleri bu renk derinliklerinde başarısız sonuçlar üretmişlerdir.

İkinci grupta 8-bit renk derinliğine sahip dosyalar yer almaktadır. Bu gruptaki dosyalar için yapılan sıkıştırma sonuçları Çizelge 5.2'de verilmiştir.

Çizelge 5.2. 8-bit renk derinliğindeki dosyaların sonuçları

Dosya Adı		8_orb	8_karikatur	8_harita	*TOPLAM*
Tekil Renk Sayısı		10	38	49	
Dosya Boyutu (byte)		226.846	111.430	593.078	*931.354*
Sıkıştırılmış Boyutlar (byte)	**RAR**	**5.783**	**14.716**	**61.250**	***81.749***
	ZIP	**7.202**	**17.631**	**72.404**	***97.237***
	PNG	**7.431**	**18.250**	**71.710**	***97.391***
	TIFF-ZIP	10.910	19.726	84.337	*114.973*
	ISSDC	8.236	19.954	91.761	*119.951*
	GIF	12.564	20.252	88.448	*121.264*
	TIFF-LZW	17.322	21.173	102.028	*140.523*
	TIFF-LZWP	19.364	25.402	117.023	*161.789*
	TIFF-RLE	39.790	41.445	194.039	*275.274*
	JPEG2000	57.589	99.044	704.757	*861.390*
	TIFF-JPEG	227.798	111.053	594.382	*933.233*

Çizelge 5.1'de olduğu gibi Çizelge 5.2'de de ilk üç sıra yine RAR, ZIP, PNG şeklindedir. Aralarındaki fark PNG'nin 8_harita dosyasında ikinci en iyi sıkıştırma oranına sahip olmasıdır. Renk sayısı ve görüntünün karmaşıklığı arttıkça PNG'nin ZIP'e göre daha başarılı olduğu diğer çizelgelerde de görülecektir. İki çizelge arasındaki bir diğer fark ise LZW-tabanlı sıkıştırma yapan GIF yönteminin, LZW ve LZWP ile desteklenen TIFF'in önüne geçmiş olmasıdır.

JPEG yöntemleri temelde sadece 24-bit renk derinliğini desteklemektedirler. Bu yöntemler, daha düşük renk derinliklerine sahip görüntüleri sıkıştırırken, önce bu görüntüleri 24-bit renk derinliğine göre tekrar kodlayıp daha sonra sıkıştırma işlemini uyguladıkları için, düşük renk derinliklerinde çoğu zaman sıkıştırma yerine şişirme yaptıkları Çizelge 5.1 ve Çizelge 5.2'de görülmektedir. JPEG-LS yöntemi ile sıkıştırma yapmak için kullandığımız LOCO Encoder, 24-bit haricindeki renk derinliklerinde sıkıştırma yapamadığı için JPEG-LS Çizelge 5.1 ve Çizelge 5.2'de yer almamıştır.

Üçüncü grup 24-bit renk derinliğinde olan, fakat az sayıda tekil renge sahip olan dosyalardan meydana gelmektedir. Bu gruptaki dosyalar için yapılan karşılaştırmanın sonuçları Çizelge 5.3'te verilmiştir.

Çizelge 5.3. 24-bit renk derinliğinde ve az sayıda tekil renge sahip dosyaların sonuçları

Dosya Adı		24_cizim	24_orb	24_karikatur	*TOPLAM*
Tekil Renk Sayısı		8	22	38	
Dosya Boyutu (byte)		49.206	677.358	328.206	*1.054.770*
Sıkıştırılmış Boyutlar (byte)	**RAR**	**201**	**8.818**	**19.492**	***28.511***
	ISSDC	**372**	**8.477**	**21.616**	***30.465***
	ZIP	**301**	**10.596**	21.771	***32.668***
	GIF	1.673	13.237	**20.252**	*35.162*
	PNG	445	12.538	27.655	*40.638*
	TIFF-ZIP	781	19.392	23.811	*43.984*
	TIFF-LZW	3.124	34.710	29.890	*67.724*
	TIFF-LZWP	2.200	42.429	31.911	*76.540*
	JPEG2000	4.926	56.462	99.044	*160.432*
	JPEG-LS	1.166	75.657	97.587	*174.410*
	TIFF-RLE	49.191	48.390	252.874	*350.455*
	TIFF-JPEG	8.392	315.210	246.530	*570.132*

Bu grup için de en iyi sıkıştırma yöntemi RAR olmuş, fakat önceki çizelgelerden farklı olarak ZIP ikinciliği ISSDC'ye kaptırmıştır. Önceki grupta TIFF-LZW ve TIFF-LZWP'yi geçmiş olan GIF, bu grupta TIF-ZIP'i de geçmiştir. 24-bit renk derinliğinde daha iyi sıkıştırma oranları sağlamaya başlayan JPEG ailesi, buna rağmen düşük tekil renk sayılarına sahip olan bu grupta da diğer yöntemlere göre geride kalmıştır.

Arka planı beyaz olan 24_orb dosyasında 1/14 gibi bir sıkıştırma oranı sağlayan TIFF-RLE, arka planı beyaz olmayan 24_cizim ve 24_karikatur dosyalarında çok düşük sıkıştırma oranları sağlayabilmiştir. Bunun nedeni 24-bit renk derinliğinde RGB formatının kullanılmasıdır. Beyaz renk RGB formatında 255,255,255 olarak saklandığı için, beyaz bir arka plan dosyada birbirini takip eden yüzlerce 255 karakteri anlamına gelmektedir. Bu da RLE için aranılan durumdur. Fakat 24_cizim dosyasındaki mavi arka plan 255,255,0 olarak dosyada saklanmaktadır. Bu yüzden seri bozulmakta ve sıkıştırma oranı çok düşük olmaktadır.

Dördüncü grup, 24-bit renk derinliğinde ve 200-256 arası tekil renk sayısına sahip olan dosyalardan oluşmaktadır. Bu dosyalardan biri şekil türünde (24_jpeg) diğer ikisi ise gri-tonlamalı (diğer bir deyişle siyah/beyaz) fotoğraf türündedir. Bu üç dosya için yapılan karşılaştırmanın sonuçları Çizelge 5.4'te verilmiştir.

Çizelge 5.4. 24-bit renk derinliğinde ve 200-256 tekil renge sahip dosyaların sonuçları

Dosya Adı		24_jpeg	24_fener	24_saray2	*TOPLAM*
Tekil Renk Sayısı		201	235	255	
Dosya Boyutu (byte)		432.486	840.054	1.179.702	*2.452.242*
Sıkıştırılmış Boyutlar (byte)	**JPEG2000**	28.689	**151.081**	**220.449**	***400.219***
	RAR	**11.531**	**233.453**	**329.384**	***574.368***
	GIF	**16.697**	**228.137**	**334.758**	***579.592***
	PNG	17.262	253.907	344.461	*615.630*
	ZIP	**13.890**	259.796	360.820	*634.506*
	ISSDC	17.082	270.644	408.143	*695.869*
	TIFF-ZIP	19.598	330.469	452.729	*802.796*
	TIFF-LZWP	32.983	320.514	470.641	*824.138*
	TIFF-LZW	34.169	374.172	505.145	*913.486*
	JPEG-LS	53.477	427.215	633.104	*1.113.796*
	TIFF-RLE	38.832	446.022	684.360	*1.169.214*
	TIFF-JPEG	153.590	545.628	786.432	*1.485.650*

Çizelge 5.4'te görüldüğü gibi, 24_jpeg dosyasında yedinci en iyi sıkıştırma oranına ulaşabilen JPEG2000, fotoğraf türündeki diğer iki dosyada ise açık farkla birinci olarak, toplamda da birinci olmuştur. Fotoğraf türündeki dosyalarda RAR ve GIF birbirlerine yakın sonuçlar elde etmişler, ikici ve üçüncü sıraları paylaşmışlardır. Şekil türündeki 24_jpeg dosyası her ne kadar 201 farklı renge sahip olsa da, önceki gruplardaki görüntü dosyaları gibi az detaya sahip olması sebebiyle, sözlük tabanlı yöntemler tarafından daha iyi sıkıştırılmış olması şaşırtıcı değildir.

Beşinci ve son grupta renkli fotoğraf dosyaları yer almaktadır. Bu grupta yer alan dört dosya için yapılan karşılaştırmanın sonuçları Çizelge 5.5'te verilmiştir. 8-bit renk derinliğini destekleyen GIF formatı, 256'dan fazla sayıda tekil renk içeren görüntülerde ancak kayıplı sıkıştırma yapabildiği için, bu grupta GIF ile yapılan sıkıştırma oranları dikkate alınmamıştır.

Dördüncü grupta JPEG2000'nin gösterdiği başarıyı yakalayamayan JPEG_LS, beşinci grupta JPEG2000'nin ardından ikinci sırayı alarak, esasen renkli fotoğraflar için etkin olduğunu göstermiştir. Diğer dört çizelgenin hepsinde son sırada yer alan TIFF-JPEG, ancak bu çizelgede üst sıralarda yer alabilmiştir. Yani, JPEG ailesinin üç üyesi ancak renkli fotoğraflar söz konusu olduğunda ilk beşin içine girebilmişlerdir. Tüm gruplarda ilk üç sırada yer almayı başaran RAR, bu grupta da başarılı olmuştur. İlk

gruplarda başarılı olan sözlük tabanlı yöntemlerin (LZW ve ISSDC), bu grupta başarısız olduğu görülmektedir.

Çizelge 5.5. 24-bit renk derinliğinde ve çok sayıda tekil renge sahip olan dosyaların sonuçları

Dosya Adı		24_manzara	24_saray1	24_aile	24_lena	*TOPLAM*
Tekil Renk Sayısı		46.841	85.427	101.565	148.279	
Dosya Boyutu (byte)		1.440.054	1.179.702	921.654	786.486	*4.327.896*
Sıkıştırılmış Boyutlar (byte)	**JPEG2000**	**377.390**	**649.372**	**523.733**	**444.585**	***1.995.080***
	JPEG-LS	511.338	**620.991**	**528.733**	**445.799**	***2.106.861***
	RAR	**421.700**	**642.859**	**560.331**	485.059	***2.109.949***
	PNG	496.153	682.892	565.194	**476.216**	*2.220.455*
	TIFF-JPEG	**486.958**	770.240	644.066	518.720	*2.419.984*
	TIFF-LZWP	598.830	897.161	761.487	652.358	*2.909.836*
	ZIP	750.367	847.724	764.478	733.627	*3.096.196*
	TIFF-ZIP	820.075	889.929	780.518	745.663	*3.236.185*
	TIFF-LZW	1.145.069	1.076.069	981.005	953.414	*4.155.557*
	ISSDC	1.389.214	1.065.647	980.735	846.809	*4.282.405*
	TIFF-RLE	1.455.623	1.144.516	934.179	797.941	*4.332.259*

Tüm grupların sonuçları değerlendirildiğinde;

- Sıkıştırma paketleri içinde RAR, görüntü formatları içinde ise PNG, tüm gruplarda başarılı sonuçlar ortaya koymuş, her tip görüntü için kullanılabilir olduklarını ispatlamışlardır.
- ZIP genellikle RAR'ın gerisinde kalmış, grup numarası arttıkça RAR'ın daha da gerisine düşmüş, beşinci grupta aralarındaki fark %50'ye yaklaşmıştır.
- 20 seneyi aşkın bir süredir kullanılan GIF, renkli fotoğraflar haricinde halen kullanılabilir bir alternatif olduğunu göstermiştir.
- TIFF-LZW'nin farklı bir uyarlaması olan ve ilk üç grupta onun gerisinde kalan TIFF-LZWP, renk sayısı arttıkça TIFF-LZW'ye oranla daha başarılı olmuştur.
- TIFF-ZIP, genel amaçlı veri sıkıştırma yöntemi olan ZIP'in gerisinde ama ona yakın sonuçlar elde etmiş, fakat tüm dosyalar için PNG'nin gerisinde

kaldığından dolayı, görüntü formatları arasında bir alternatif olamayacağı görülmüştür. Son grupta TIFF-LZWP'nin de gerisinde yer almıştır.

- Basit bir sıkıştırma mantığı kullanan TIFF-RLE, tüm gruplarda son sıralarda yer almıştır.
- ISSDC en büyük başarısını üçüncü grupta, yani yüksek renk derinliği ve düşük renk sayısına sahip olan dosyalarda göstermiştir.
- JPEG ailesinin kayıpsız sıkıştırma yapan uyarlamaları da, kayıplı olanlar gibi fotoğraf görüntülerinde başarılı olduklarını göstermişlerdir. İçlerinde en iyisi olan JPEG2000 kayıpsız sıkıştırması, özellikle siyah/beyaz fotoğraflar için alternatifsizdir. JPEG-LS ve TIFF-JPEG ise sadece renkli fotoğraflarda etkilidirler.

Pratikte, fotoğraflar söz konusu olduğunda, kayıpsız sıkıştırmak yerine, dikkatli bakılmadıkça fark edilemeyecek oranlarda küçük kayıplarla sıkıştırma yaparak, kayıpsız JPEG sürümlerine göre bile birkaç kat daha fazla sıkıştıran kayıplı JPEG sürümlerini kullanmak daha uygundur. Bu açıdan bakıldığında, ilk üç grubun sonuçları daha anlamlı hale gelmektedir. Çünkü kayıpsız sıkıştırmanın en çok tercih edildiği durumlar, şekiller gibi az renk kullanan ve az detay içeren dosyalardır. Giriş kısmında da belirttiğimiz gibi, bu tür görüntülerde kayıplı sıkıştırmanın yarattığı görüntüdeki bozulma daha fazla gözle görünür hale gelmekte ve kayıpsız sıkıştırmayı gerekli kılmaktadır.

ISSDC algoritması aslında görüntü sıkıştırma için özel olarak tasarlanmış bir algoritma değildir. LZW gibi hem metin, hem de görüntü sıkıştırmada kullanılabilen genel bir sıkıştırma yöntemidir. ISSDC bir görüntü formatı haline getirilirse (Birçok görüntü formatında olduğu gibi, renk derinliği, yükseklik, genişlik gibi değerler dosyanın başlığında saklanırsa), kayıpsız görüntü sıkıştırması için LZW tabanlı sıkıştırma yapan GIF ve TIFF yöntemlerine iyi bir alternatif olabileceği yapılan testlerde görülmüştür.

5.4. Kayıplı Görüntü Sıkıştırma Yöntemleri

Kayıplı görüntü sıkıştırma yöntemleri birçok ara işlemden oluşur. Bu yöntemlerde genellikle önce renk dönüşümü yapılır ve ardından fazlalıklar atılır. Daha

sonra dönüşüm (DCT, DWT, ...) yapılarak veri daha az bit ile ifade edilebilecek bir biçime getirilir. Daha sonra niceleme (quantization) işlemi ile uygun bit-oranına indirgenir ve en son olarak ta kayıpsız bir yöntem (Huffman, aritmetik, ...) ile sıkıştırılır.

Bu bölümde, önce günümüzde en yaygın olarak kullanılan kayıplı sıkıştırma yöntemi olan JPEG yöntemi, ardından da daha yeni ve daha üstün özelliklere sahip olan fakat henüz yeterince yaygınlaşamamış olan JPEG2000 tanıtılmıştır.

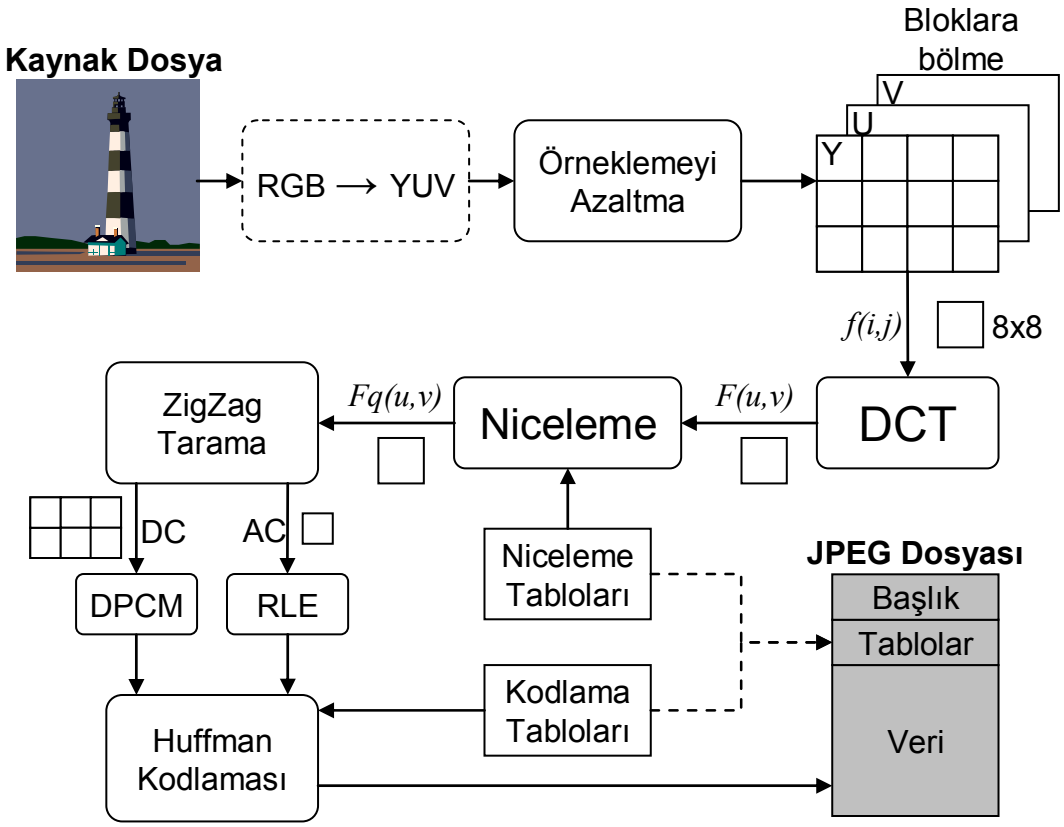
5.4.1. Yaygın Olarak Kullanılan Yöntem: JPEG Standardı

1983 yılının Nisan ayında ISO, görüntü verilerini sayısal haberleşme ağları üzerinde hızlı bir şekilde iletebilecek yöntemler geliştirmesi amacıyla PEG'i (Photographic Experts Group) kurdu. 1986'da CCITT'de oluşturulan bir alt grup, faks iletimi için gri-tonlu ve renkli veri sıkıştırma üzerine çalışmalara başladı. Renkli faks sistemleri için gerekli olan sıkıştırma yöntemleri, PEG tarafından geliştirilen yöntemlerle büyük benzerlikler içeriyordu. Böylece, iki grubun kaynaklarını birleştirerek ortak bir standart oluşturmak amacıyla beraber çalışmaları gerektiğine karar verildi ve oluşan yeni komiteye JPEG (Joint PEG) adı verildi. 1992 yılında tamamlanan ve genelde JPEG ismi ile bilinen standart resmi olarak ISO/IEC 10918-1 ve CCITT Rec. T.81 isimleri ile yayınlanmıştır [ISO, 1994 ve CCITT, 1992]. Gregory K. Wallace tarafından yayınlanan makale JPEG ile ilgili detaylı bilgi verir [Wallace, 1991]. JPEG standardının dosya biçimini tanımlayan JFIF (JPEG File Interchange Format) ise, Eric Hamilton tarafından yayınlanmıştır [Hamilton, 1992].

JPEG, DCT (Discrete Cosine Transform) tabanlı ve 24-bit RGB görüntüler üzerinde çalışabilen bir standarttır. Sıkıştırmadaki kayıp oranı (sıkıştırma miktarı) Q faktörünün değiştirilmesi ile ayarlanabilir. 24-bit renk derinliğine sahip fotoğraflarda çok iyi sonuçlar verdiği için sayısal fotoğrafçılıkta çok kullanılır. Fakat bir rengin ağırlıkta olduğu resimlerde sıkıştırma oranı azalır. JPEG, tek bir algoritmadan ibaret olmayıp, birçok görüntü sıkıştırma yönteminin bir araya getirilmesi ile oluşturulmuş bir standart olduğu için, sıkıştırma ve açma süreleri tek algoritmaya sahip olan kayıpsız sıkıştırma yöntemlerine göre daha fazladır. JPEG standardının kullanılmaya başlandığı yıllarda bu unsur zaman zaman sorun yaratsa da, günümüzün hızlı sistemlerinde bekleme süreleri büyük oranda azalmıştır.

JPEG görüntü sıkıştırma yöntemi temel olarak aşağıdaki işlemlerden oluşur:

1. Renk bilgisi RGB türünde ifade edilmişse, YUV'a çevirme. (isteğe bağlı)
2. Örneklemeyi azaltma.
3. Görüntüyü 8×8 büyüklüğünde bloklara bölme.
4. DCT ile piksel bilgisini uzamsal etki alanından frekans etki alanına çevirme.
5. Her katsayıyı belirli bir tamsayı değere bölerek ve bölümleri en yakın tamsayıya yuvarlayarak, DCT dönüşümü sonuç değerlerini nicelendirme.
6. Sonuç katsayılarına zigzag tarama düzeninde RLE uygulama ve ardından Huffman kodlamasını kullanarak kayıpsız sıkıştırma.



Şekil 5.7. JPEG sıkıştırmasının blok şeması

Şekil 5.7'de JPEG sıkıştırma işlemleri gösterilmiştir. Sıkıştırılan görüntünün kod çözme işlemi ile eski haline getirilmesi, sıkıştırmada yapılan işlemlerin ters işlemlerinin yapılması ile gerçekleşir. Bu nedenle bu bölümde sadece sıkıştırma işlemi anlatılacaktır.

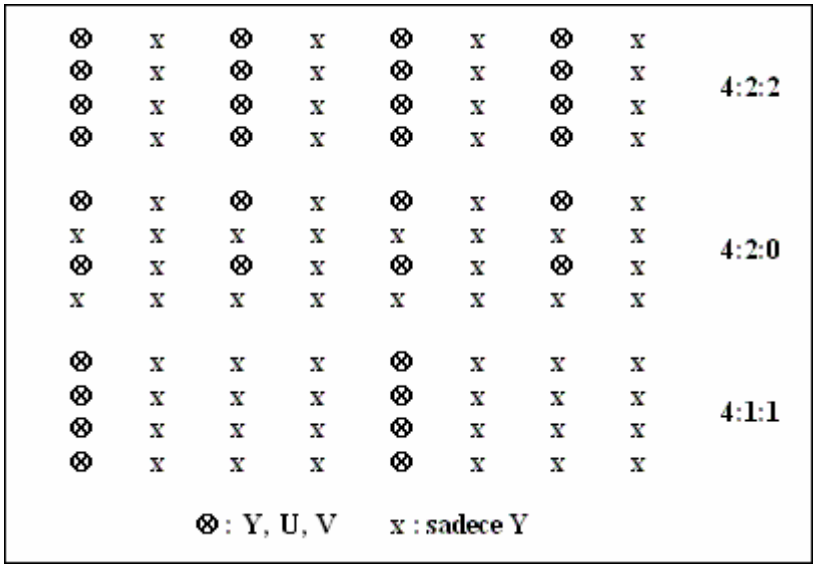
RGB'den YUV'a dönüştürme JPEG sıkıştırması için gerekli değildir, ama bu dönüşüm R, G ve B bileşenleri arasındaki korelasyonu azalttığı için sıkıştırmanın etkinliği artar. YUV renk biçimi, renkleri, aydınlık (luminance) ve renklilik (chrominance) olarak ifade eder. Y aydınlık şiddetini, U (Cb) ve V (Cr) değerlerinin birleşimleri de renkliliği ifade eder. RGB→YUV dönüşümü bir lineer dönüşümdür ve bu dönüşüm için Eric Hamilton tarafından tarif edilmiş olan denklem ve ters dönüşüm denklemi aşağıda verilmiştir.

$$Y = 0.299R + 0.587G + 0.114B$$
$$U = 0.564(B-Y) = -0.169R - 0.331G + 0.500B + 128$$
$$V = 0.713(R-Y) = 0.500R - 0.419G - 0.081B + 128$$

$$R = Y + 1.402\ (V-128)$$
$$G = Y - 0.34414\ (U-128) - 0.71414\ (V-128)$$
$$B = Y + 1.772\ (U-128)$$

Renkli televizyonlar ve CRT monitörler ters dönüşüm ile R, G ve B değerlerini elde ederek 3 farklı elektron tabancalarında bu değerleri kullanırlar. Siyah-Beyaz televizyonlar ise ters dönüşüme gerek duymadan sadece aydınlık bilgisini (Y) direkt olarak kullanabilirler.

İnsan gözü parlaklığa daha çok, renkliliğe daha az hassastır. Yani, U ve V değerleri bir miktar azaltılırsa, görüntü insan gözü için çok fazla farklılaşmayacaktır. O halde 2×2 ebatlarında 4 piksellik bir blok için Y, U ve V değerlerinden 4'er adet (4:4:4) kullanmak yerine örneklemeyi azaltma (down-sampling) işlemi ile 4:2:2 düzenine getirilerek 2'şer adet, 4:2:0 veya 4:1:1 düzenine getirilerek te 1'er adet kullanılabilir. Bu düzenler Şekil 5.8'de gösterilmiştir. JPEG'te genellikle 4:2:2 veya 4:2:0 kullanılır.



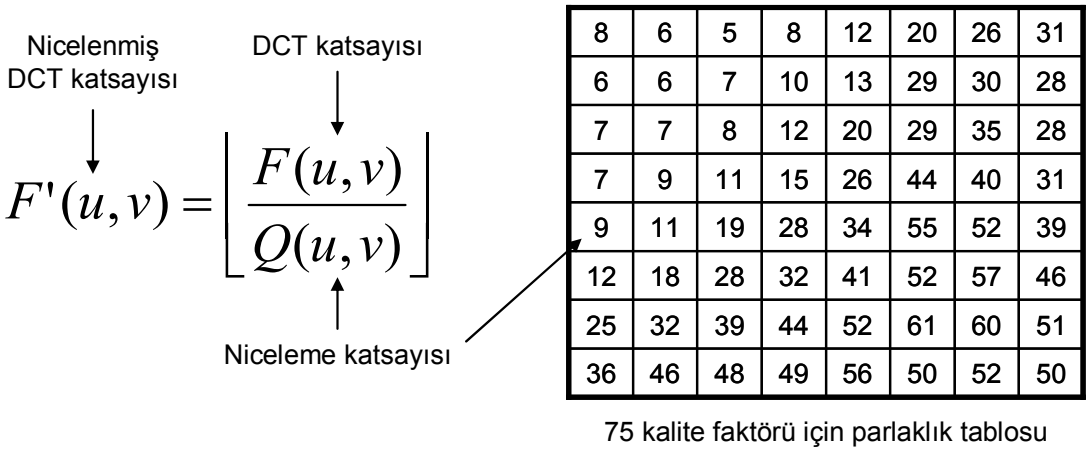
Şekil 5.8. Renklilik bilgisinin alt örneklenmesi

Örnekleme azaltıldıktan sonra görüntü 8×8 bloklara bölünür ve her blok diğerinden bağımsız olarak ayrık kosinüs dönüşümüne (DCT) tabi tutulur. Sonuçta ortaya yine 64 katsayı çıkacaktır. Bunlardan ilkine DC (ilk piksel - $F(0,0)$) ve diğerlerine de AC adı verilir. DCT dönüşümünün formülü aşağıda verilmiştir:

$$F(u,v) = \frac{1}{4} c_v c_u \sum_{i=0}^{7} \sum_{j=0}^{7} f(i,j) \cos\frac{(2i+1)u\pi}{16} \cos\frac{(2j+1)v\pi}{16}$$

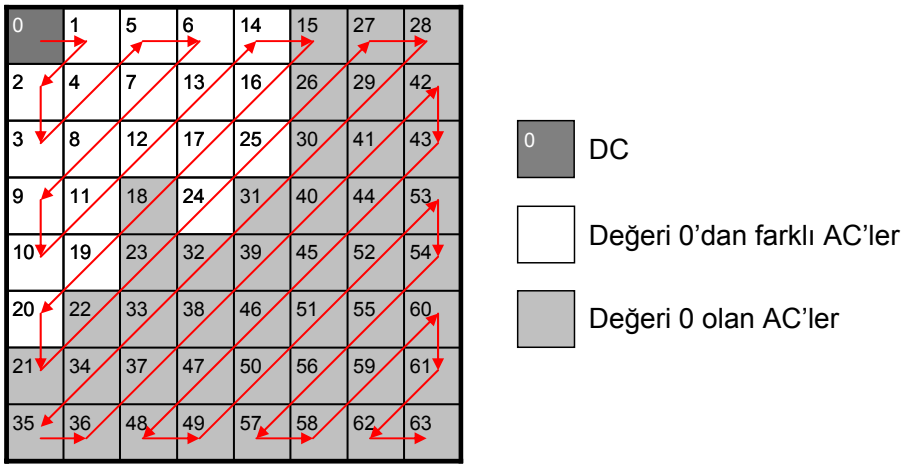
Bu formülde c_v ve c_u çarpanları DC için $1/\sqrt{2}$, AC'ler için ise 1'dir.

DCT dönüşümü sonrasında ortaya çıkan katsayılar genellikle ondalıklıdır. Bu değerler niceleme yöntemi ile tamsayılara dönüştürülür. Niceleme yönteminde öncelikle her katsayı 1 ile 255 arasında bir tamsayıya bölünür ve daha sonra en yakın tamsayıya yuvarlanır (bu esnada kayıp meydana gelir). Katsayıların hangi değere bölüneceği niceleme tabloları (8×8 büyüklüğünde bir niceleme matrisi) sayesinde belirlenir. Parlaklık ve renklilik için farklı tablolar kullanılır. Görüntüdeki kayıp miktarını belirlemek için bir kalite çarpanı (quality factor) kullanılır. Kullanılan kalite çarpanına göre tablolar değişiklik gösterir. Şekil 5.9'da nicelemenin formülü ve örnek olarak ta 75 kalite çarpanı için parlaklık tablosu verilmiştir.



Şekil 5.9. Niceleme işlemi ve 75 kalite çarpanı için parlaklık tablosu

İnsan gözü yüksek frekanslara karşı daha az hassas olduğu için, niceleme çizelgesinde de görülebileceği gibi, bloğun sağ alt yarısında (sağ alt eşkenar üçgende) yer alan yüksek frekans katsayıları daha büyük sayılara bölünerek sıfıra indirgenmeye çalışılır. Niceleme işlemi sonrasında yüksek frekans katsayıları ağırlıklı olarak sıfırlardan oluşacaktır. Bu iki boyutlu yapı tek boyuta indirgenirken Şekil 5.10'da görüldüğü gibi bir köşegen tarama işlemi uygulanır. Bu şekilde bir tarama yapılmasının amacı, düşük frekans katsayılarından yüksek frekans katsayılarına doğru sıralı bir düzen yaratmaktır. Böylece, yüksek frekans katsayılarının çoğunluğu sıfır olabileceği için, tekrar eden çok sayıda ardışık sıfırdan oluşan ve RLE ile iyi sıkıştırılabilir bir yapının oluşması da sağlanmış olacaktır.



Şekil 5.10. Köşegen (zigzag) tarama

RLE kullanılarak değeri sıfır olan AC katsayıları sıkıştırıldıktan sonra, diğer AC katsayıları için ve her bloğun sol üst pikseli olan DC katsayıları için farklı Huffman Tabloları kullanılarak, son aşama olan kayıpsız sıkıştırma gerçekleştirilecektir. DC katsayıları için, bir önceki DC değeri ile kodlanacak olan DC arasındaki farkın kodlandığı bir tablo tercih edilirken, AC katsayıları için RLE kodlaması ile uyumlu daha farklı bir tablo kullanılır.

5.4.2. Yeni Yöntem: JPEG2000 Standardı

JPEG2000 standardı, JPEG standardının kısıtlamalarını gidermek ve düşük bit-oranlarında yüksek kalitede görüntüler elde etmek amacıyla tasarlanmıştır. Ayrık Dalgacık Dönüşümü (Discrete Wavelet Transform - DWT) teknolojisini temel alarak, bilinen en iyi sıkıştırma teknolojilerinin kullanılmasıyla oluşturulmuş bir kodlama sistemidir.

JPEG2000 projesi, kayıpsız ve az kayıplı sıkıştırma yapmak üzere 1994 yılında Ricoh tarafından geliştirilen CREW (Compression with Reversible Embedded Wavelets) mimarisi üzerine inşa edilmiştir. Aslında Ricoh, 1995 yılında bu mimarinin JPEG-LS standardı olması için teklif yapmış, ama birçok farklı mimariler arasından yapılan değerlendirme sonucunda HP'nin LOCO-I mimarisi JPEG-LS standardı olarak seçilmiştir. Fakat CREW mimarisinin sahip olduğu zengin özellikler görülünce, 1996 yılında Ricoh'tan Martin Boliek tarafından yapılan teklif ile "JPEG2000 görüntü kodlama sistemi" projesi başlatılmıştır [Boliek, 1996]. Teklif edilen algoritmalar arasında en iyi düşük bit-oranı performansına sahip olan WTCQ (wavelet/trellis coded quantization) niceleme algoritması olarak benimsenmiştir. Aritmetik kodlama biçimi olarak, JBIG-2'de de kullanılmış olan MQ-Kodlaması seçilmiştir. 2000 yılında çekirdek kodlama sistemi tamamlanmış ve yayınlanmıştır [ISO/IEC, 2000].

JPEG'teki en büyük dezavantajlardan biri; görüntü önce 8×8'lik bloklara ayrılıp daha sonra DCT uygulandığı için, özellikle yüksek sıkıştırma oranlarında bloklar arası geçişin keskinleşmesi ve gözle fark edilir hale gelmesidir. Özellikle 1/30'dan yüksek oranlarda sıkıştırmış görüntülerde bloklar belli olur. JPEG2000'de böyle bir bloklara bölme olmadığı için bu dezavantaj ortadan kalkmıştır.

JPEG2000'in üstünlükleri arasında ilk akla gelenler şunlardır:

- Kaliteden ödün vermeden daha yüksek sıkıştırma oranı sağlar.
- Önemli bölgelerin belirlenerek daha yüksek kalitede, önemsiz bölgelerin daha düşük kalitede sıkıştırılmasına imkân verir. (ROI: Region of Interest)
- Görüntüyü belirli bir büyüklüğe (piksel başına bit oranına) kadar sıkıştırmaya olanak sağlar. (JPEG'te sadece kalite oranı var, "sıkıştırılmış hali 100KB olsun" şeklinde bir seçim yapılamıyor.)
- Aynı mimari içinde kayıpsız sıkıştırmaya da imkân sağlar. Eğer görüntüyü istenilen büyüklüğün daha azına kayıpsız olarak sıkıştırabiliyorsa, otomatik olarak kayıpsız sıkıştırmayı kullanır.
- Renkli görüntülerde 48-bit, gri-tonlamalı görüntülerde de 16-bit renk derinliğine izin verir.
- JPEG2000 dosya formatı (.jp2) XML tabanlıdır.
- İstemci/sunucu görüntü uygulamaları için ve sınırlı kaynağa sahip olan kablosuz cihazlar için ilave fonksiyonel özelliklere sahiptir. Örneğin, belirli bir zaman aralığında aktarabileceği en kaliteli görüntüyü aktarır.

KODLAMA

JPEG2000 temel olarak aşağıdaki işlemlerden oluşur:

1. RGB'den YUV'a ICT ya da RCT ile dönüşüm (isteğe bağlı)
2. Görüntüyü bölümlendirme (Tiling) (isteğe bağlı)
3. Her bölüme ayrık dalgacık dönüşümü (DWT) uygulanması
4. İlgili olunan bölgenin (ROI) seçimi (isteğe bağlı)
5. DWT dönüşümü sonuç değerlerini nicelendirme
6. MQ-Kodlaması

JPEG2000 standardının 2000 yılında yayınlanmış olan birinci bölümünde (Part I), iki dönüşüm tipi tanımlanmıştır. Bunlardan ilki, JPEG'te de kullanılan RGB → YUV dönüşümüdür. Bu dönüşümün ters dönüşümü içerdiği ondalık değerler nedeniyle kayıplı olabildiği için, bu dönüşüme *Ters Çevrilemez Bileşen Dönüşümü* (*Irreversible Component Transform - ICT*) denir. Diğer bir dönüşüm ise, her bileşenin kayıpsız

olarak tekrar elde edilmesine olanak sağlayan *Ters Çevrilebilir Bileşen Dönüşümü* (*Reversible Component Transform - RCT*) dönüşümüdür. Kayıpsız sıkıştırma yapılacak ise sadece RCT kullanılır. Kayıplı sıkıştırma için ise ikisinden biri seçilebilir. RCT dönüşümünün ve ters dönüşümünün formülleri aşağıda verilmiştir:

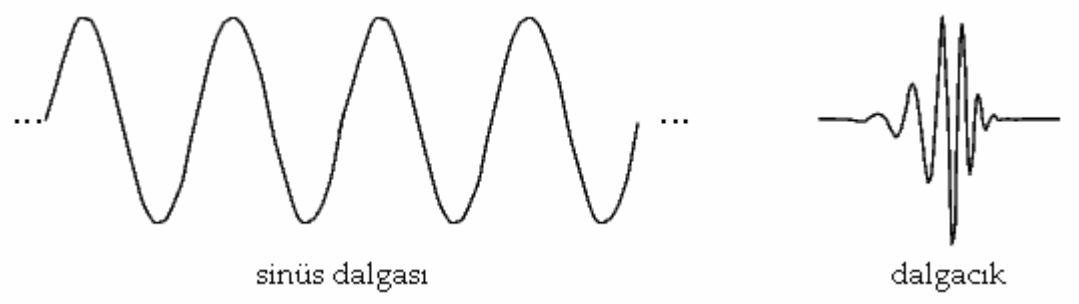
$$Y_r = \left\lfloor \frac{R + 2G + B}{4} \right\rfloor \qquad G = Y_r - \left\lfloor \frac{U_r + V_r}{4} \right\rfloor$$

$$U_r = R - G \qquad R = U_r + G$$

$$V_r = B - G \qquad B = V_r + G$$

Renk dönüşümü işleminden sonra, bölümlendirme (tiling) adı verilen bir işlem ile görüntü birbiri üzerine geçmeyen dikdörtgensel bloklara ayrılır. Bu sayede her bir bölüm sanki farklı bir görüntü imiş gibi birbirinden bağımsız olarak sıkıştırılır. Sınırlarda yer alanlar dışındaki bölümlerin boyutları ikinin katları kadardır. Bölümlendirme işlemi, bellek gereksinimlerini azalttığı için ve eğer gerekli değilse görüntünün tamamı yerine sadece belirli kısımlarını açabildiği için kullanışlıdır.

Bölümlendirme aşamasından sonra gelen ayrık dalgacık dönüşümü aşamasını anlatmadan önce, dalgacık (wavelet) ve sürekli dalgacık dönüşümü kavramlarından bahsetmek gerekir. Dalgacıklar, sinüzoidal dalgalar gibi zaman içinde sürekli olmayıp belirli bir zaman aralığında yüksek bir enerjiye sahipken diğer zaman aralıklarında sıfır enerjiye sahiptirler (yani enerjileri yoktur). Sürekli olmadıkları için dalgacık adını almışlardır. Şekil 5.11'de sinüs dalgası ve Daubechies db10 dalgacık yan yana gösterilmiştir. Daubechies, Coiflet, Morlet, Meyer gibi birçok farklı dalgacık türü içinden hangisinin kullanılacağı uygulamaya bağlıdır.

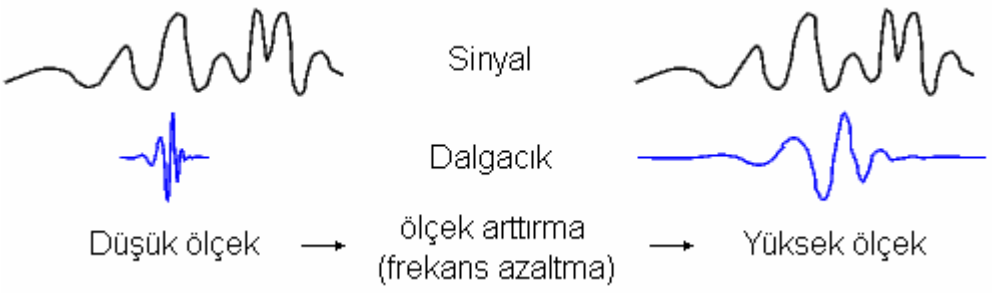


Şekil 5.11. Sinüs dalgası ve Daubechies db10 dalgacık

Sürekli Dalgacık Dönüşümü (Continuous Wavelet Transform - CWT): Bir sinyalin tamamının, bir ana dalgacık fonksiyonunun kaydırılmış (zamanı değiştirilmiş) ve ölçeklenmiş (frekansı değiştirilmiş) örnekleri ile çarpılmasının toplamı olarak tanımlanması işlemidir.

$$C(ölçek, konum) = \int_{-\infty}^{\infty} f(t)\psi(ölçek, konum, t)dt$$

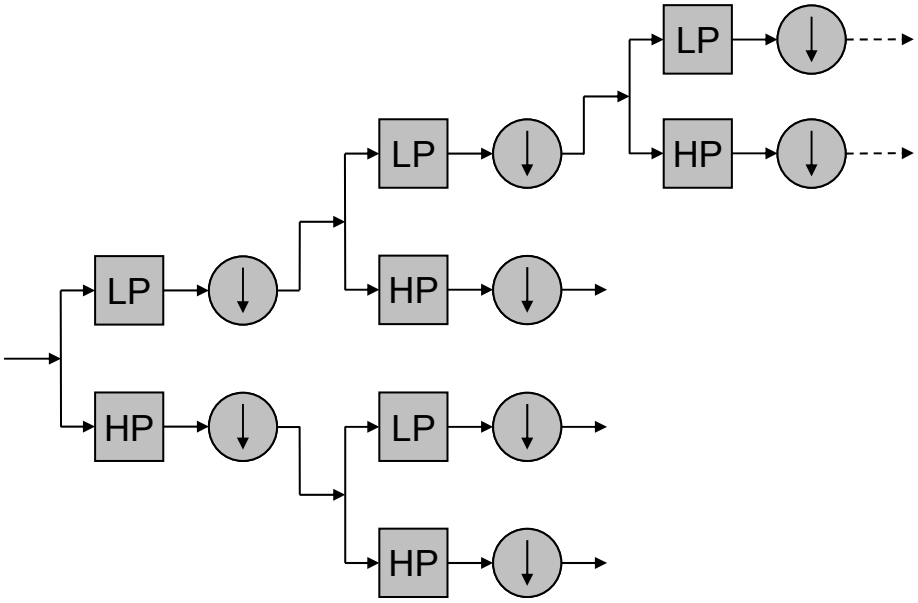
Sürekli dalgacık dönüşümünün çıktısı, ölçeğin ve konumun fonksiyonları olan birçok dalgacık katsayısıdır. Her katsayıyı uygun ölçeklenmiş ve kaydırılmış dalgacık ile çarpmak orijinal sinyalin dalgacık bileşenlerini verir. Şekil 5.12'de ölçeği arttırılarak (frekansı azaltılarak) sinyale uygun hale getirilmiş bir dalgacık gösterilmektedir. Sağ taraftaki ölçeği arttırılmış dalgacık zamanda da biraz geri doğru kaydırılırsa, üstünde yer alan sinyalin bir kısmını temsil edebileceği görülebilir.



Şekil 5.12. Dalgacıklarda ölçek arttırma

Ayrık Dalgacık Dönüşümü (Discrete Wavelet Transform - DWT): Her olası ölçeği ve konumu değerlendirerek dalgacık katsayılarını hesaplamak yerine, daha az hesaplama yapmak için sadece belirli bir ölçek ve konum kümesinin kullanılması ayrık dalgacık dönüşümü yaklaşımının temelini oluşturur. Genellikle ölçek ve konum bilgileri 2'nin pozitif ve negatif üsleri olarak seçilir. Böyle bir düzenin biri alçak geçiren ve diğeri yüksek geçiren olan iki filtreden oluşan iki kanallı alt bant kodlayıcısı kullanılarak gerçekleştirilmesi ilk kez Mallat tarafından yapılmıştır [Mallat, 1989]. Sonraları bu yöntem daha da geliştirilerek Şekil 5.13'te görüldüğü gibi birçok alçak geçiren ve yüksek geçiren sayısal filtrelerin birleştirilmesiyle oluşan filtre bankaları ile,

çok katmanlı (görüntü işlemede çok çözünürlüklü) ayrıştırma işlemlerinin de yapılması sağlanmıştır.



Şekil 5.13. DWT'de kullanılan çok katmanlı filtre bankası

JPEG2000'de DWT uygulanırken, kayıpsız sıkıştırma için ters çevrilebilir olduğundan dolayı mutlaka 5-tap/3-tap tamsayı filtresi kullanılmalıdır. Daubechies 5/3 için analiz ve sentez filtre katsayıları Çizelge 5.6'da verilmiştir. Kayıplı sıkıştırma için ise genellikle 9-tap/7-tap kayan noktalı (ters çevrilemez) Daubechies filtresi kullanılır. Ama istenirse 5/3 filtresi de kullanılabilir. Daubechies 9/7 için analiz ve sentez filtre katsayıları Çizelge 5.7'de verilmiştir [Christopoulos, 2000].

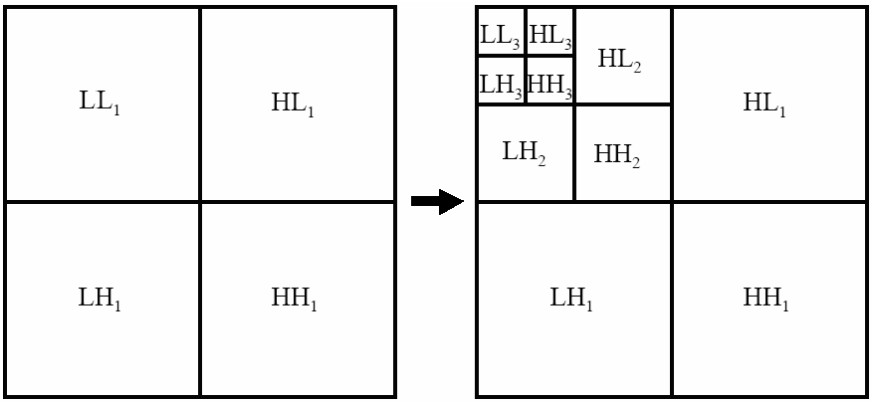
Çizelge 5.6. Daubechies 5/3 için analiz ve sentez filtre katsayıları

	Analiz filtre katsayıları		**Sentez filtre katsayıları**	
i	Alçak geçiren filtre $h_L(i)$	Yüksek geçiren filtre $h_H(i)$	Alçak geçiren filtre $g_L(i)$	Yüksek geçiren filtre $g_H(i)$
0	6/8	1	1	6/8
±1	2/8	-1/2	1/2	-2/8
±2	-1/8			-1/8

Çizelge 5.7. Daubechies 9/7 için analiz ve sentez filtre katsayıları

	Analiz filtre katsayıları		**Sentez filtre katsayıları**	
i	Alçak geçiren filtre $h_L(i)$	Yüksek geçiren filtre $h_H(i)$	Alçak geçiren filtre $g_L(i)$	Yüksek geçiren filtre $g_H(i)$
0	0.6029490182363579	1.115087052456994	1.115087052456994	0.6029490182363579
±1	0.2668641184428723	-0.5912717631142470	0.5912717631142470	-0.2668641184428723
±2	-0.07822326652898785	-0.05754352622849957	-0.05754352622849957	-0.07822326652898785
±3	-0.01686411844287495	0.09127176311424948	-0.09127176311424948	0.01686411844287495
±4	0.02674875741080976			0.02674875741080976

Ayrık dalgacık dönüşümü her bölüm (tile) için ayrı ayrı yapılır. Bir görüntünün dalgacık ayrıştırmasında, ayrışım önce satır satır, sonra sütün sütün gerçekleştirilir. Örneğin N×M boyutlarında bir görüntüde, önce her satır filtrelenir ve filtre çıkışı iki adet N×(M/2) boyutunda görüntü elde etmek için alt örneklenir. Daha sonra her sütun filtrelenir ve dört adet (N/2)×(M/2) boyutunda görüntü elde etmek için alt örneklenir. Her alt görüntü aranılan alt bant yapısı sağlanana kadar bu şekilde dörde bölünmeye devam eder. Şekil 5.14'te bu bölünmeler gösterilmiştir. L seviyeli bir dönüşümden L+1 farklı çözünürlük elde edilir. En düşük frekans alt bandı çözünürlük-0, orijinal görüntü ise çözünürlük-L olarak adlandırılır.



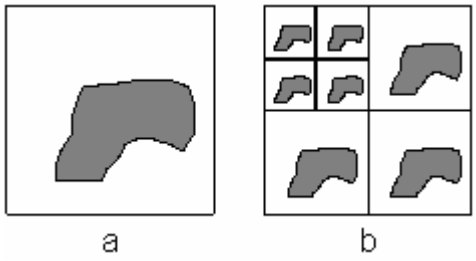
Şekil 5.14. Filtre bankasının görüntüye uygulanması

LL bandı sadece düşük çözünürlük seviyesinde görünür ve DC örnek değerlerini içerir. LH bandı yatay alçak geçiren filtre ve dikey yüksek geçiren filtre kullanır. HL bandı dikey alçak geçiren filtre ve yatay yüksek geçiren filtre kullanır. HH bandı ise her

iki yönde de yüksek geçiren filtre kullanır. Şekil 5.15'te bir fotoğraf görüntüsünün alt bantlara ayrışımı gösterilmiştir. Şekilden de anlaşıldığı gibi LL bandı aslında görüntünün çözünürlüğü yarı yarıya azaltılmış halini içerir. Diğer bantlar da bu düşük çözünürlüklü görüntüden orijinal görüntünün tekrar elde edilebilmesi için gerekli olan dikey ve yataydaki fark bilgilerini saklar.

Şekil 5.15. Bir resmin alt bantlara ayrışımı

ROI kodlamasında, görüntünün belirli bölgeleri daha yüksek kalitede, diğer yerler daha düşük kalitede kodlanır. Görüntünün belirli bir bölümü ile daha çok ilgili olduğumuz durumlarda bu işlem kullanışlıdır. İlgili olunan bölge (ROI maskesi) seçildikten sonra, dalgacık dönüşümünden elde ettiğimiz katsayılardan hangilerinin daha yüksek kalitede sıkıştırılması gerektiğini bulmak için, Şekil 5.16'da görüldüğü gibi maskeye dönüşüm uygulayıp, hangi katsayıların maskenin içine düştüğüne bakarız. Bu bilginin sıkıştırılan görüntüyü açma esnasında bilinmesine gerek yoktur.



Şekil 5.16. (a) ROI maskesi (b) Dönüştürülmüş ROI maskesi

Ayrık dalgacık dönüşümünden sonra, tüm dalgacık katsayıları belirli bir niceleme adım büyüklüğüne bölünerek ve çıkan sayılar aşağı yuvarlanarak, sayıl niceleme işlemi gerçekleştirilir. Kullanılan niceleme işlemi kademeli olarak çalışan

dalgacık dönüşümü yapısına uygun olacak şekilde tasarlanmıştır. Her *b* alt bandının $a_b(u,v)$ dönüşüm katsayıları, aşağıda verilen formül ile $q_b(u,v)$ değerlerine nicelenir:

$$q_b(u,v) = sign(a_b(u,v)) \left\lfloor \frac{|a_b(u,v)|}{\Delta_b} \right\rfloor$$

Bu formülde Δ_b niceleme adım büyüklüğünü göstermektedir. Bu değer, *b* alt bandının R_b dinamik aralığı ile alt bandın üssü ε_b ve mantisi μ_b vasıtasıyla ilişkilidir. R_b, orijinal görüntünün ilgili bölüm bileşenini temsil etmek için kullanılan bit sayısına ve ayrık dalgacık dönüşümünün seçimine bağlıdır. Niceleme adım büyüklüğünün hesaplanması aşağıdaki formül ile gerçekleştirilir:

$$\Delta_b = 2^{R_b - \varepsilon_b} \left(1 + \frac{\mu_b}{2^{11}}\right)$$

Her alt bant için sadece bir niceleme adım büyüklüğüne izin verilir. JPEG'te olduğu gibi JPEG2000'de de sıkıştırılmış resmin kalitesi bu adım büyüklüklerinin seçimi ile ilgilidir. Eğer tamsayı dalgacık dönüşümü kullanıldıysa (kayıpsız sıkıştırma yapılıyorsa), niceleme işlemi yapılmasına gerek yoktur (Δ_b=1 olarak seçilebilir). Bu durumda dönüşümden sonra elde edilen katsayılar, direkt olarak bit katarına dönüştürülebilir.

Niceleme işleminden sonra gelen kodlama işlemi iki aşamada gerçekleştirilir. Birinci aşamada her alt bant ile ilişkili olan nicelenmiş dönüşüm katsayıları, kod-blokları adı verilen dikdörtgensel bloklara bölünürler. Daha sonra üç geçişten oluşan EBCOT (Embedded Block-based Coding with Optimized Truncation) tabanlı bir bit-düzlemi kodlaması her kod-bloğuna uygulanır ve ortaya çıkan semboller aritmetik kodlama tabanlı MQ-Kodlaması ile sıkıştırılır. Kullanışlı bit dizisi oluşturma özelliklerine sahip olan MQ-Kodlaması, JBIG ve JPEG standartlarında kullanılan QM-Kodlamasına işlevsel olarak benzerdir.

5.4.3 Fraktal Tekniği

Fraktal terimi ilk defa Benoit Mandelbrot tarafından, gözlemlediği birçok yapının içinde tekrar eden desenleri tanımlamak amacıyla kullanılmıştır [Mandelbrot, 1977]. Mandelbrot aynı zamanda bu fraktalların matematiksel ifadeler şeklinde tanımlanabileceğini ve çok küçük miktarda veri ve algoritma ile yaratılabileceğini keşfetmiştir. Örneğin bir banyodaki fayansların hepsi birbirine benzerdir. Bir fayansı desen olarak kabul edip, belli bir miktar uzaklıktaki başka bir fayansı bu desenin döndürülmüş, ötelenmiş, yansıtılmış hali olarak matematiksel bir model ile temsil edebiliriz. Böylece, sadece bu deseni ve tüm fayanslar için gerekli matematiksel modeli alıcıya gönderirsek, alıcı bu verileri kullanarak tüm fayansları oluşturabilir.

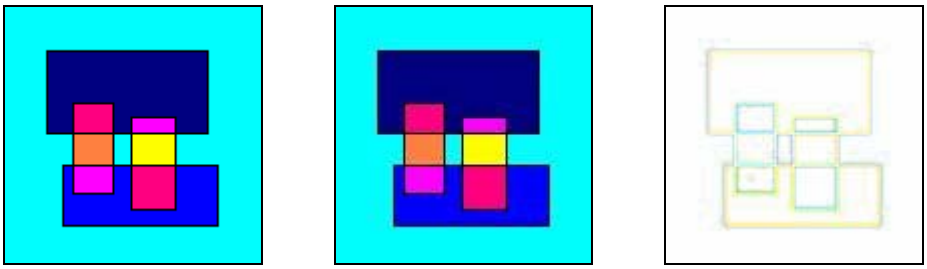
Fraktal sıkıştırmanın tekrarlanan fonksiyonlar ile gerçeklenebileceği fikri ilk defa Michael Barnsley ve Alan Sloan tarafından ortaya atılmıştır [Barnsley ve Sloan, 1987]. Barnsley'in fractal'lar ile ilgili yazdığı kitapta fraktal dönüşümünden (fractal transform) bahsedilmektedir [Barnsley, 1988]. Jacquin'in 1989'da tamamladığı doktora tezinde ve sonraki yıllarda yazdığı makalelerde, Fraktal Teorisi tabanlı görüntü kodlama üzerine yaptığı çalışmalar yer almaktadır [Jacquin, 1992, 1993].

Fraktal ile görüntü sıkıştırma, insan gözünün karakteristik özelliğine dayalı ve piksel tabanlı olan JPEG yöntemlerinden çok farklıdır. Fraktal tekniğinin en önemli avantajı, açma işleminin basit ve hızlı olmasıdır. Fakat sıkıştırma işlemi, açmanın tam tersine çok karmaşıktır. Bir resimdeki tekrar eden fraktalları bulmak milyonlarca, milyarlarca karşılaştırma işleminin yapılmasını gerektirebilir. Sıkıştırma işleminin çok zaman alması nedeniyle fraktal tekniği henüz yeterince yaygınlaşamamıştır.

5.5. JPEG ve JPEG2000 Yöntemlerinin Karşılaştırılması

Kayıplı sıkıştırma algoritmaları söz konusu olduğunda, sıkıştırma oranından çok bozulma oranı (kayıp miktarı) önemlidir. Çünkü kayıplı sıkıştırma algoritmaları belirli bir kalite çarpanını parametre olarak aldıkları için, kaliteden feragat ederek sıkıştırma oranını arttırabilmektedirler. Bu nedenle, karşılaştırma yapılırken asıl test edilmesi gereken unsur ne kadar çok sıkıştırdığı değil, ne kadar kaliteli sıkıştırdığıdır.

Bozulma miktarını çoğu zaman gözle görmek zordur. Örneğin Şekil 5.17 (a)'da gösterilen ve karşılaştırmada kullandığımız "cizim.bmp" görüntü dosyasının, JPEG - 80 kalite çarpanı ile sıkıştırılmış hali olan (b) ile aralarında fark yok gibi görünse de, b'deki renk değerlerinin a'daki değerlerden çıkarılıp tersinin alınması ile oluşturulan (c) görüntüsünden de anlaşılabileceği gibi, özellikle frekans değerlerinin yüksek olduğu kenar bölgelerde renk farklılıkları vardır.



(a) **(b)** **(c)**

Şekil 5.17. (a) cizim.bmp, (b) cizim.jpg, (c) fark.bmp

Kayıp miktarını ölçmek: Kayıplı sıkıştırma sonucunda görüntüdeki bozulma oranını ölçmenin faklı yöntemleri vardır. Bunlar arasında en bilinenleri; MSE, RMSE, PSNR, MAE ve PAE'dir.

MSE (mean squared error) hataların kareleri toplamının ortalamasıdır. MSE genellikle σ^2 olarak gösterilir. RMSE (root mean squared error) ise MSE'nin kareköküdür.

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - y_n)^2$$

Bazen MSE yerine, hatanın büyüklüğünün orijinal piksel değerinin en büyüğü (peak-tepe) ile olan ilişkisi ile ilgileniriz. Bu gibi durumlarda PSNR (peak signal-to-noise ratio) yöntemini kullanırız.

$$PSNR(dB) = 10\log_{10}\frac{x_{peak}^2}{\sigma_d^2}$$

Hataların mutlak değerinin ortalaması olan MAE (mean absolute error) ve Mutlak hataların en büyüğünü (tepe noktasını) gösteren PAE (peak absolute error), daha seyrek kullanılan yöntemlerdir. Bazı uygulamalar hataların belli bir eşik değerini aşmamasını isterler. Bu gibi durumlarda PAE önemlidir.

$$MAE = \frac{1}{N}\sum_{n=1}^{N}|x_n - y_n|$$

$$PAE = \max_n|x_n - y_n|$$

Formüllerden de anlaşılabileceği gibi eğer karşılaştırılan görüntüler birebir eşitseler PSNR değeri sonsuz, diğer değerler ise sıfır çıkacaktır.

5.5.1. Amaç ve Yöntem

Bu karşılaştırmanın amacı düşük oranlı (yüksek kaliteli) sıkıştırmada hangi yöntemin daha etkili olacağını görmektir. Çünkü JPEG2000'in sıkıştırma oranının yüksek olduğu durumlarda JPEG'e olan üstünlüğü zaten bilinmektedir. Şekil 5.18'de de görüldüğü gibi sıkıştırma oranı arttıkça JPEG yöntemi blok görünümlü olmaya başlarken, JPEG2000'de ise görüntü bulanıklaşmaya başlamaktadır. Yüksek frekanslı bölümleri bulanık olan görüntü, genel görünüm olarak blok yapılı görüntüden daha iyi görünmektedir [Ebrahimi vd., 2004].

Şekil 5.18. A) Orijinal görüntü B) JPEG ile 1/100 oranında sıkıştırılmış görüntü C) JPEG2000 ile 1/100 oranında sıkıştırılmış görüntü

JPEG Mimarisinin görüntüyü önceden belirlenen bir büyüklüğe kadar sıkıştırma yeteneği olmadığı için, önce JPEG ile 80 kalite çarpanına göre sıkıştırma yapıldı. Ardından JPEG'in sıkıştırdığı büyüklükler parametre olarak JPEG2000'e verilerek, her bir dosyanın aynı büyüklüğe kadar JPEG2000 tarafından da sıkıştırılması sağlandı.

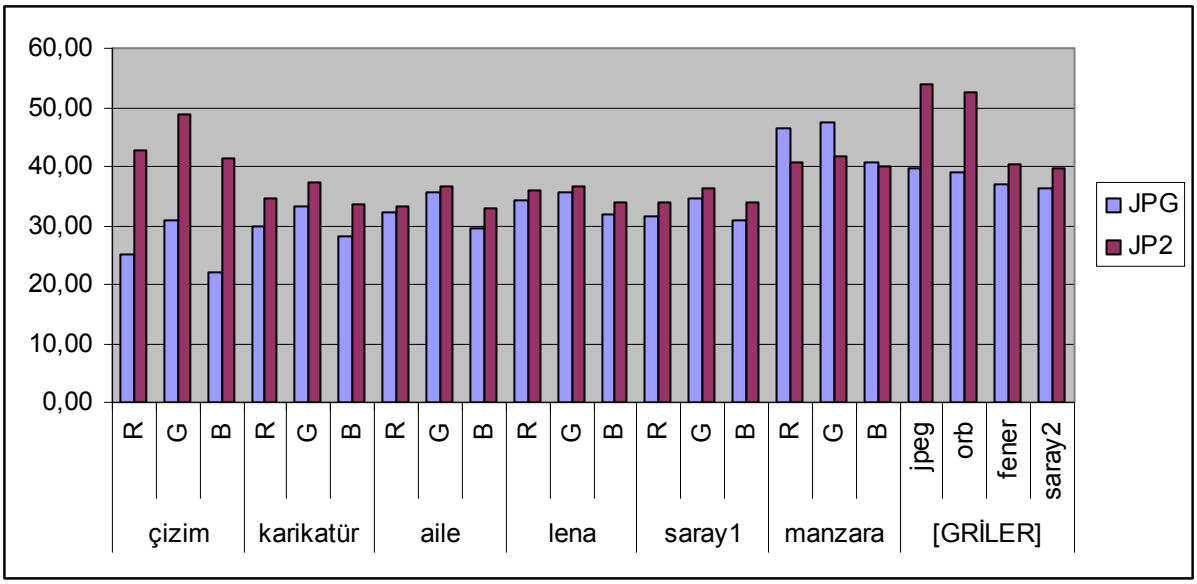
Karşılaştırmada örnek görüntü olarak, daha önce kayıpsız görüntü sıkıştırma yöntemlerini karşılaştırırken kullandığımız 16 adet görüntü dosyasından 24-bit renk derinliğine sahip olan 10 tanesini kullandık. Bunlardan fener ve saray2 gri-tonlamalı fotoğraf görüntüleri, orb ve jpeg ise renksiz şekil görüntüleri oldukları için kayıp miktarlarının tespitinde R, G, B olarak ayrı ayrı değerlendirilmemişlerdir. Çizelge 5.8'de kullanılan görüntü dosyaları ve sıkıştırma oranları verilmiştir.

Çizelge 5.8. Kullanılan görüntü dosyaları ve sıkıştırma oranları

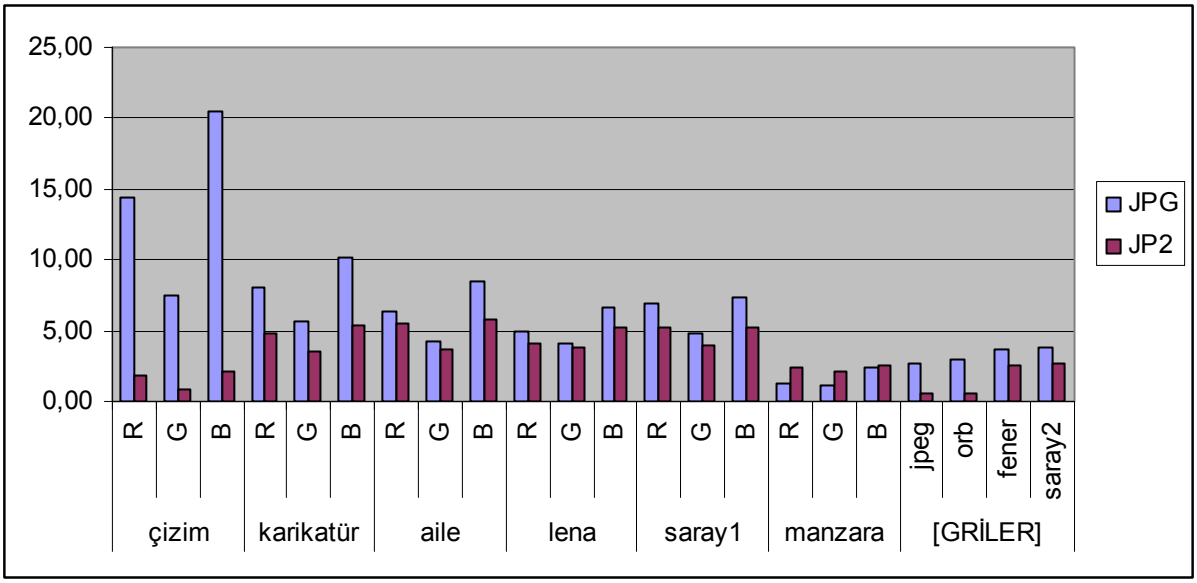
dosya adı	orijinal büyüklük (byte)	sıkıştırılmış büyüklük (byte)		sıkıştırma oranı (bbp)	
		JPG	JP2	JPG	JP2
cizim	49.167	3.014	3.059	1,47	1,49
karikatur	327.804	34.391	34.805	2,52	2,55
aile	921.615	61.029	61.309	1,59	1,60
lena	786.447	44.124	44.874	1,35	1,37
saray1	1.179.663	78.297	78.727	1,59	1,60
manzara	1.440.015	58.780	59.386	0,98	0,99
jpeg	432.447	24.196	24.571	1,34	1,36
orb	677.319	47.942	48.109	1,70	1,70
fener	840.015	50.667	51.053	1,45	1,46
saray2	1.179.663	70.593	70.389	1,44	1,43

Karşılaştırmada JasPer Software tarafından geliştirilmiş olan "Image Comparison Program" kullanılarak PSNR, RMSE, MAE ve PAE değerleri hesaplanmıştır. Bu program RGB türündeki görüntü dosyalarını karşılaştırırken, Kırmızı, Yeşil ve Mavi renk değerlerinin her biri için ayrı sonuçlar üretmekte, RGB türünde olmayanlarda ise tek bir sonuç değeri üretmektedir.

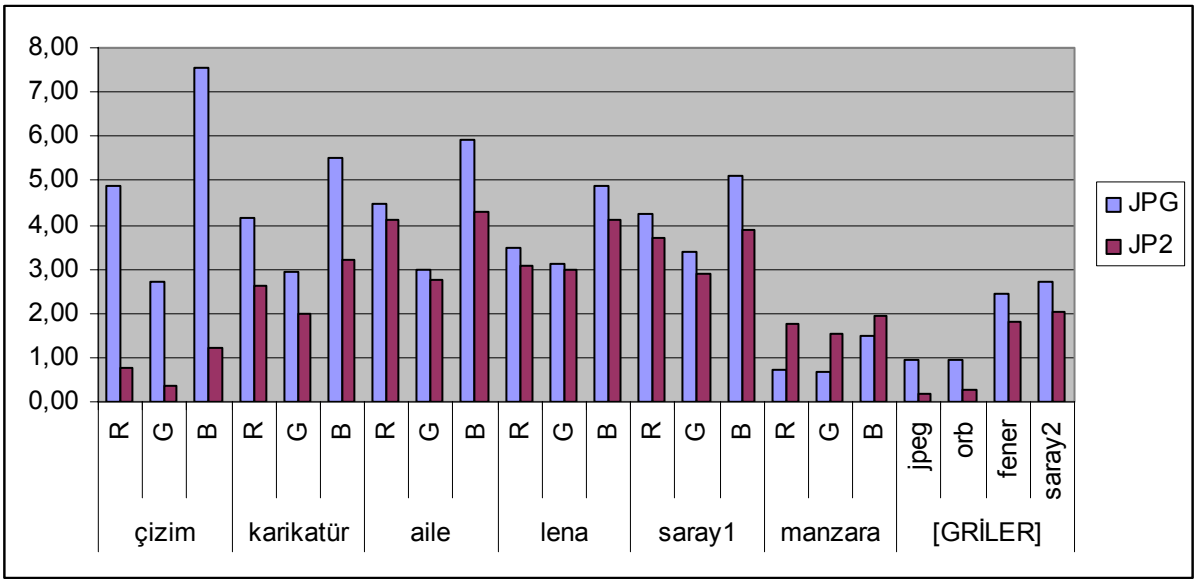
5.5.2. Sonuçlar



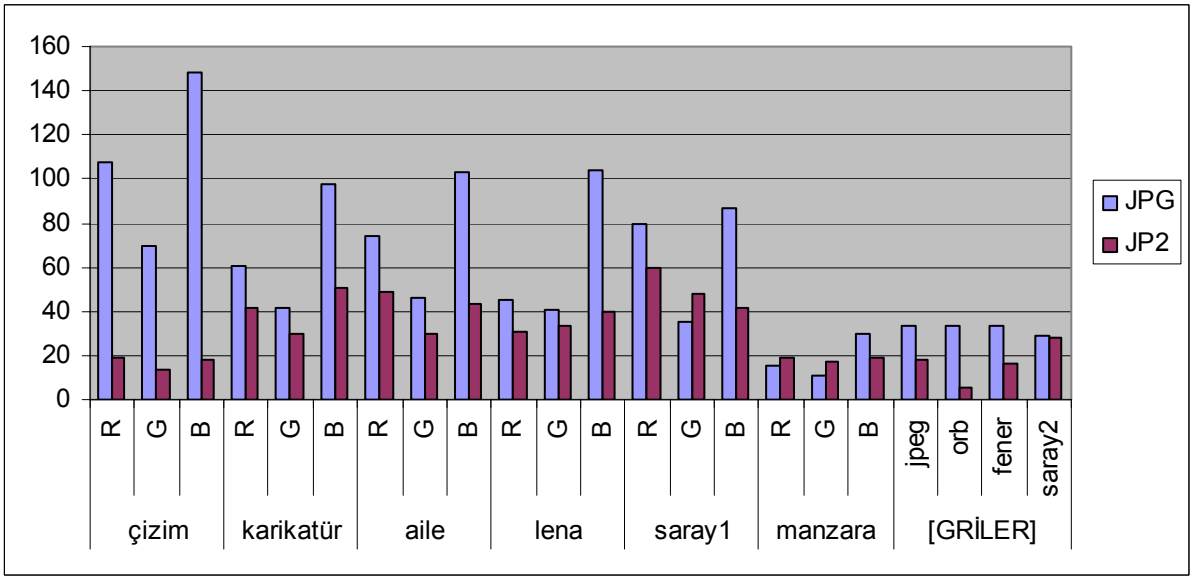
Şekil 5.19. PSNR karşılaştırması sonuçları



Şekil 5.20. RMSE karşılaştırması sonuçları



Şekil 5.21. MAE karşılaştırması sonuçları



Şekil 5.22. PAE karşılaştırması sonuçları

Sonuçlar değerlendirildiği zaman, *manzara* görüntüsü haricinde, JPEG2000'in üstün olduğunu görülmektedir. Özellikle fotoğraf türünde olmayan *cizim*, *karikatur*, *jpeg* ve *orb* dosyalarında JPEG2000, JPEG'ten çok daha iyi performans göstermiştir. JPEG yöntemi 1/25 ile en yüksek sıkıştırma oranına ulaştığı *manzara* görüntüsünde aynı zamanda kalite olarak ta JPEG2000'den daha iyi sonuç vermiştir. JPEG yönteminin yüksek oranda sıkıştırabildiği ayrıntısı az olan fotoğraf tipi görüntülerde JPEG2000'den daha başarılı olduğu söylenebilir.

Sıkıştırma zamanlarını değerlendirmek gerekirse, JPEG2000'in çok daha yavaş sıkıştırdığı gözlenmiştir. Karmaşık bir yapıya sahip olması nedeniyle, JPEG2000'in sayısal fotoğraf makinelerinde efektif olarak kullanılabilmesi için çok hızlı işlemcilere ihtiyaç duyulmaktadır. Her ne kadar bazı sayısal fotoğraf makinesi üreticileri JPEG2000 standardını destekleyen ürünleri geliştirmekte olduklarını açıklamış olsalar da, bu standardı destekleyen tamamlanmış bir ürüne günümüzde pek rastlanmamaktadır.

6. HAREKETLİ GÖRÜNTÜ SIKIŞTIRMA

Sıkıştırılmamış hali çok büyük olan video dosyaları, kayıpsız yöntemler ile düşük oranlarda sıkıştırılabildikleri için, hareketli görüntü sıkıştırma işlemi çoğunlukla kayıplı yöntemler ile gerçekleştirilir. Bununla birlikte hızlı kodlama ihtiyacı olduğu durumlarda veya en küçük bir kaybın bile istenmediği profesyonel amaçlı uygulamalar için kayıpsız sıkıştırma da gerekli olabilmektedir. Şimdiye kadar üretilmiş en etkili kayıplı sıkıştırma yöntemi olan H.264/MPEG-4 AVC, istenirse kayıpsız sıkıştırma yapabilmektedir.

6.1. Kayıpsız Hareketli Görüntü Sıkıştırma Yöntemleri

Kayıpsız hareketli görüntü sıkıştırma yöntemleri genellikle gerçek zamanlı video yakalama gibi hızlı kodlama yapmanın gerekli olduğu durumlarda kullanılmaktaydı. Pratikte, kayıpsız bir yöntem ile sıkıştırılmış olan bir hareketli görüntü çok yer kapladığı için, ve saklama ünitelerinin kapasiteleri 10 sene öncesine kadar fazla büyük olmadığı için, saklama amacıyla daha sonra kayıplı bir yöntem ile tekrar sıkıştırılmaktaydı. Son yıllarda üretilen yüksek hızlı işlemciler sayesinde, kayıplı bir yöntem ile de gerçek zamanlı video yakalama işlemleri yapılabilmektedir. Bu nedenle günümüzde hareketli görüntü yakalama işlemlerinde de genellikle kayıplı kodlayıcılar tercih edilmektedir.

Huffyuv, SheerVideo, CorePNG, MSU Lossless Video Codec, LCL, Lagarith, SKYUV [Smith, 2000], Lossless Motion JPEG gibi birçok kayıpsız sıkıştırma yöntemi vardır.

Huffyuv: Ben Rudiak-Gould tarafından 90'lı yıllarda geliştirilmiş olan Huffyuv, en çok kullanılan kayıpsız hareketli görüntü (video) sıkıştırma algoritmalarından biridir. Sıkıştırma yapmayan YUV'un alternatifi olmuştur. Sıkıştırma ve açma işlemlerinde hızlı bir algoritmadır. Kaynak kodu açık, dağıtımı serbesttir.

Huffyuv çerçeve-içi (intra-frame) öngürü kullanarak sıkıştırma yapar. Yani her çerçeveyi bir önceki çerveden bağımsız olarak kodlar. Kullandığı öngörü modeli Kayıpsız JPEG ile benzerdir. Öngörülen değer ile orijinal değerin farkını Huffman kodlaması ile sıkıştırır. Temel olarak 3 öngörü modeline sahiptir:

- Left: Sol taraftaki pikselleri kullanır.
- Gradient: Sol+Üst-ÜstSol kullanır.
- Median: Sol, Üst, ve gradient öngörüsünün ortalamasını kullanır.

RGB türünde bir hareketli görüntü daha iyi sıkıştırılabilmesi için R-G, G ve B-G şeklinde kodlanabilir. Her bileşen için (R,G,B veya Y,U,V bileşenleri) ayrı Huffman tablosu kullanılır. Kodlayıcı her bileşen için koleksiyonunda hazır bulunan Huffman tablolarından en uygun olanını seçer. İleriki sürümlerde bu tabloların değişebileceği düşünülerek bu tablolar istenirse kodlanan hareketli görüntüde saklanabilir.

Renk dönüşümü aşamasında renklilik bilgileri alt örnekleme ile azaltılırsa düşük oranda kayıplı sıkıştırma yapılmış olacaktır.

MSU: "MSU (Moscow State University) Graphics and Media Lab." tarafından geliştirilmiştir. Her çerçeveyi ayrı bir resim gibi kodlamayıp, çerçeveler arasındaki yüksek oranda benzerliklerden faydalanarak çerçeveler-arası (inter-frame) öngörüsü kullanan bu yöntem, Huffyuv'a göre daha yavaş çalışmasına rağmen daha iyi sıkıştırma oranı sağlamaktadır. Kayıplı sıkıştırma seçeneği de bulunan MSU, istenirse Huffyuv gibi, tüm çerçeveleri ayrı bir resim gibi de kodlayabilir.

6.2. Kayıplı Hareketli Görüntü Sıkıştırma Yöntemleri

Kayıplı hareketli görüntü sıkıştırma üzerine geliştirilmiş olan ilk standart ITU-T H.261 standardıdır. DCT tabanlı bir video kodlama standardı olan H.261'in ilk sürümü (v1) 1990'da, ikincisi (v2) 1993'te tamamlanmıştır. H.261, CIF (Common Interchange Format – 352x288 piksel) ve QCIF (Quarter CIF – 176x144 piksel) formatlarını desteklemekte, ve 64-2048Kbps aralığında çalışmaktadır. Geliştirilmesinde görüntülü telefon (videophone) ve görüntülü konferans (videoconferencing) uygulama alanları hedef alınmıştır. ISO'nun geliştirdiği MPEG-1 ve MPEG-2 standartları H.261'i temel almıştır. Görüntülü telefon sistemlerinde ileri geri sarma gibi işlemler olmadığı için çerçeveler arası hareket farkı hesaplanması işlemleri daha basittir. Daha basit olması sayesinde daha hızlı kodlanabilir ki bu özellik, gecikme olmaması açısından görüntülü telefon için gerekli bir durumdur.

ITU-T H.263: H.261'in yerini alan ve 1995 yılında ilk sürümü tamamlanan H.263, sub-QCIF (128x96), QCIF, CIF, 4CIF (704x576) ve 16CIF (1408x1152) video formatlarını desteklemektedir. 1997'de ikinci sürüm (v2 yada H.263+ olarak bilinir), ve 2000'de üçüncü sürüm (v3 – H.263++) tamamlanmıştır. ISO'nun geliştirdiği MPEG-4 standardı H.263'ü temel almıştır.

6.2.1. Yaygın Olarak Kullanılan Yöntemler: MPEG-1 ve MPEG-2

MPEG (Moving Pictures Experts Group), 1988 yılında hareketli görüntü ve sesin sayısal saklama birimlerinde farklı sıkıştırma oranları ile saklanabilmesi için gerekli standart algoritmaları oluşturmak amacıyla ISO/IEC tarafından kurulmuştur. MPEG-1 tamamlanmadan 1990 yılında MPEG-2 projesi de başlatılmıştır. MPEG-1 1993 yılında, MPEG-2'nin ilk sürümü de 1994 yılında bitirilmiştir. Sonraki yıllarda MPEG-2 standardına yeni özellikler eklenerek geliştirilmesine devam edilmiştir. HDTV için geliştirilmeye başlanan MPEG-3 projesi, MPEG-2'nin HDTV'nin ihtiyaçlarını karşılayabileceği görülünce iptal edilmiştir.

MPEG-1 standardı, 1-1,5Mbps aralığında çalışması için genellikle 352x288 çözünürlükte 25fps (frame-per-second; kare/saniye) ile veya 352x240 çözünürlükte 30fps ile kullanılır. VHS kalitesini sağlayabildiği için VCD'lerin temel standardı haline gelmiştir. MPEG-1, temelde H.261 standardına benzemekle birlikte, yüksek bit oranlarında çok daha iyi sonuçlar verir.

MPEG-2 standardı, MPEG-1 gibi temel bir uygulama alanını hedeflemediği için uygulamadan bağımsız, genel bir standart olmuştur. Kullanıcı farklı sıkıştırma oranları ve profiller arasından seçim yapabilmektedir. Ayrıca MPEG-1'de olmayan geçmeli tarama (interlace) özelliği vardır. ITU-T ile ISO'nun ortak çalışmasının bir ürünü olduğu için H.262 ile aynı özelliklere sahip olan MPEG-2, Sayısal Uydu Alıcılarının, DVD'nin (Digital Versatile Disc) ve HDTV'nin (High Definition TeleVision) temel standardı olmuştur.

Resmi olarak ISO/IEC 11172 diye bilinen MPEG-1, 5 kısımdan oluşur. İlk üçü Sistemler, Video ve Ses'tir. Dördüncüsü, standarda uygun ürünler geliştiren üreticilerin

ihtiyaç duyduğu bilgilerin tanımlandığı Uygunluk ve Testler kısmıdır. Beşincisi ise, hem sıkıştırma hem de açma için C-kodlarının yer aldığı Yazılım Simülasyonu kısmıdır.

Hareketli görüntü sıkıştırma algoritmalarının temel çalışma prensibi, ardışık çerçeveler (frame) arasındaki farkın kodlanmasına dayanır. Eğer bir sahne değişimi meydana gelmediyse, kodlaması yapılacak olan çerçeve büyük ölçüde önceki çerçeve ile benzer olacaktır. O halde, bir çerçevenin tamamını kodlamak yerine, önceki çerçeveden farkını kodlamak daha mantıklıdır. MPEG sıkıştırma algoritması I, P ve B olmak üzere üç farklı çerçeve biçimi kullanır.

I çerçevesi (intra-frame), çerçevedeki piksellerin tamamının JPEG benzeri bir yöntem ile kodlandığı çerçeve biçimidir. P çerçevesi (predictive coded frame), bir önceki I veya P çerçevesi ile arasındaki harekete dayalı farkın kodlandığı bir çerçeve biçimidir. B çerçevesi (bidirectionally predictive coded frame) ise, hem bir önceki hem de bir sonraki I veya P çerçevesi ile arasındaki harekete dayalı farkın kodlandığı bir çerçeve biçimidir.

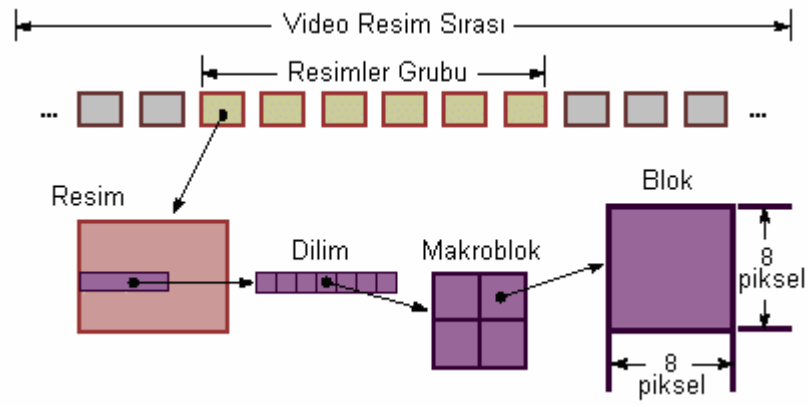
Farklı çerçevelerin bir araya gelmesi ile bir resimler grubu (GOP – group of pictures) oluşur. GOP, bir hareketli görüntüde rasgele olarak erişilebilecek en küçük birimdir. Bir GOP ya bir I çerçevesi ile, ya da sadece o I çerçevesini temel alarak harekete dayalı tahmin yürüten B çerçeveleri ile başlar. Bir B çerçevesi kendisinden sonra gelen I veya P çerçevesine bağlı olduğu için, kodlama düzeninde bağlı olduğu çerçeveden sonra yer almalıdır. Bu nedenle kodlama sırası, gösterim sırasından farklıdır. Örnek bir GOP için kodlama sırası ve gösterim sırası Çizelge 6.1'de verilmiştir.

Çizelge 6.1. Kodlama Sırası ve Gösterim Sırası

Gösterim Sırası	1	2	3	4	5	6	7	8	9	10	11	12	13
Çerçeve Tipi	I	B	B	P	B	B	P	B	B	P	B	B	I
Kodlama Sırası	1	3	4	2	6	7	5	9	10	8	12	13	11

Temel kodlama birimi *blok* adı verilen, 8×8 büyüklüğünde bir matristir. DCT ve sonrasındaki tüm işlemler her blok için ayrı ayrı yapılır. Bir *makroblok* ise, 4:2:0 alt örnekleme modu için; 4 Y, 1 Cb ve 1 Cr'den oluşan 6 blokluk bir yapıdır. Bu yapı

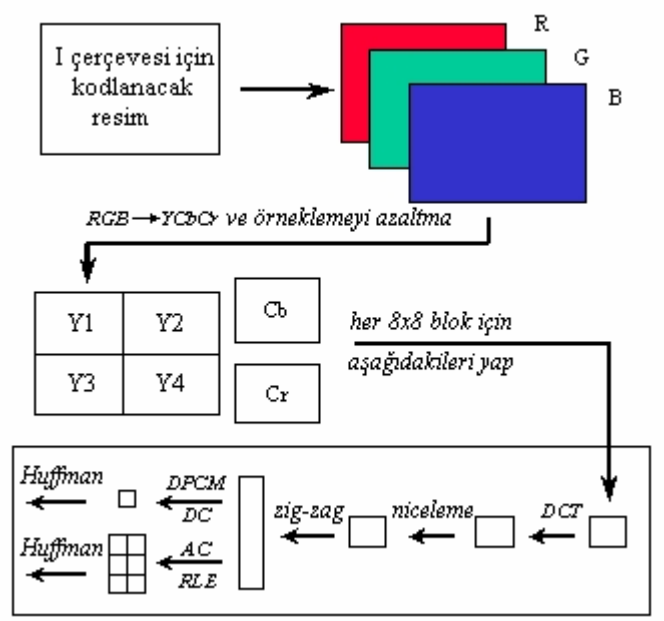
görüntünün 16×16 büyüklüğündeki alt görüntülerini temsil etmek için kullanılır. Soldan sağa veya yukarıdan aşağıya doğru dizilmiş birçok makroblok bir *dilim* (*slice*) oluşturur. Dilimler hata idaresinde önemli rol oynarlar. Eğer bit katarında bir hata ile karşılaşılırsa, kod çözücü bir sonraki dilime atlayabilir. Şekil 6.1'de bir MPEG videosunda yer alan bileşenler gösterilmiştir.



Şekil 6.1. Video serisinin bileşenleri

I çerçevesi (intra-frame)

Diğer çerçevelerden bağımsız olduğu için ayrı bir resim gibi düşünülmesi gereken I çerçevesi, sahne değişimi gibi, bir önceki çerçeveye göre çok fazla farklılıkların bulunduğu durumlarda kullanılır. I çerçevesi, Şekil 6.1'de gösterildiği gibi JPEG ile benzer bir yöntem ile kodlanır. Sadece renk dönüşümü ve niceleme aşamaları biraz farklıdır. Sıkıştırma işlemi süresince, farklı çerçeveler için farklı niceleme çizelgeleri kullanılması gerekebilir.



Şekil 6.2. I çerçevesi için kodlama şeması

RGB→YUV dönüşümü için, JFIF'te kullanılan dönüşümden farklı olarak, ITU-R BT.601 (CCIR 601 olarak ta bilinir) tavsiyenamesinde belirtilen katsayılar benimsenmiştir.

```
Y =  0.257R + 0.504G + 0.098B + 16
U = -0.148R - 0.291G + 0.439B + 128
V =  0.439R - 0.368G - 0.071B + 128

R = 1.164(Y - 16) + 1.596(V - 128)
G = 1.164(Y - 16) - 0.813(V - 128) - 0.391(U - 128)
B = 1.164(Y - 16) + 2.018(U - 128)
```

Bu katsayılar aslında JFIF'te kullanılan katsayıların aşağıdaki denklem ile dönüşümden geçirilmiş halidir.

```
Y = (219/256)*(Y in JFIF) + 16
U = (224/256)*(U in JFIF) + 16
V = (224/256)*(V in JFIF) + 16
```

Bu dönüşüm sonucunda Y değeri [16,234] aralığında, U ve V değerleri de [16,239] aralığında olabilir. [0,255] aralığının tamamının kullanılmamasının nedeni, aydınlık şiddeti ve renklilik değerleri ile birlikte, bazı kontrol değerlerinin de bileşenler içinde saklanmasıdır.

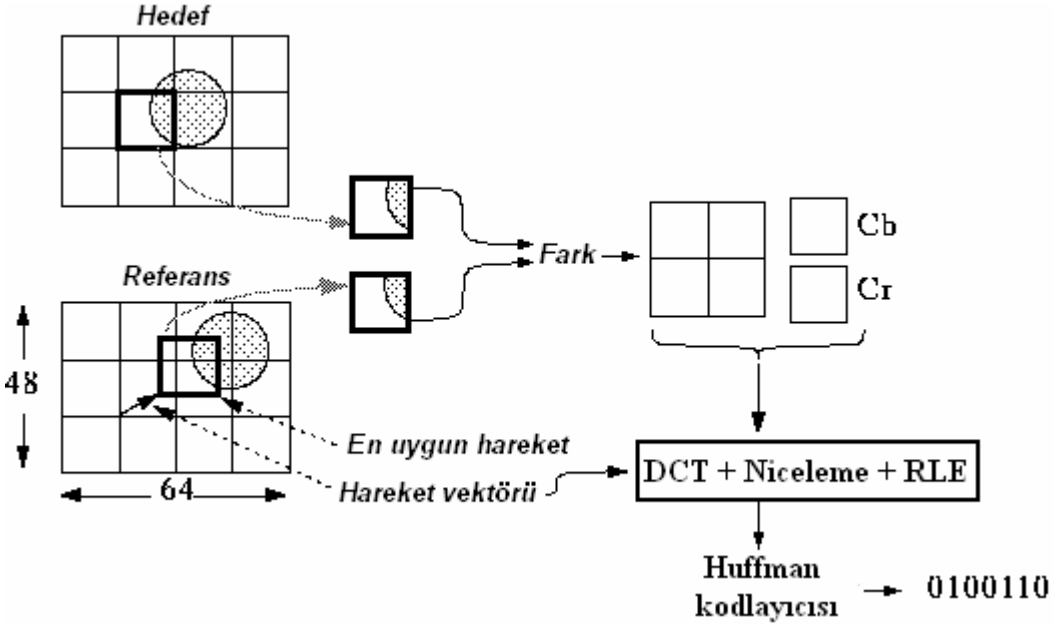
P çerçevesi (predictive coded frame)

P çerçevesi, önceki çerçeveye göre görüntüdeki farklılığın tespit edilmesi ve aradaki farkın kodlanmasını gerektirdiği için, I çerçevesine göre biraz daha karmaşık bir yapı ile kodlanır. Kodlayıcı, P çerçevesindeki her makroblok için, bir önceki P veya I çerçevesinde arama yaparak, o makrobloğun eşleniği olabilecek uygun bir referans makroblok bulmaya çalışır. Eğer bir makroblok içinde kalan nesne herhangi bir yöne hareket etmişse, ya da kameranın herhangi bir yöne doğru kayması durumu oluşmuşsa, tam olarak aynı koordinatlarda uygun bir referans makroblok bulunamayacaktır. Bu nedenle, o makrobloğun koordinatlarına yakın olan koordinatlarda da arama yapılmalıdır. Görüntüde hareketlenmenin hangi yöne doğru ve ne kadar uzaklıkta olduğunun kestirilmesi, çok sayıda karşılaştırmanın yapılmasını gerektirebilen bir durumdur. Fakat, saniyede 25-30 çerçeve işlendiği için, nesnenin bir sonraki çerçevede genellikle bir önceki çerçeveye yakın bir noktadadır, ve uygun referans blok çok fazla arama yapmadan bulunabilir. Bulunduğu zaman, iki makroblok arasındaki piksellerin farkını hesaplanarak, bu fark ile oluşturulan bir makroblok JPEG benzeri bir yöntem ile kodlanır. Eğer iki makroblok arasında fark yoksa veya küçük farklılıkların tamamı niceleme sonucunda sıfırlandıysa, sadece hareketin yönünü ve uzaklığını tayin eden hareket vektörü Huffman yöntemi ile kodlanır. Eğer karşılaştırılan iki makrobloğun koordinatları da aynıysa, hareket vektörü de sıfır olacağı için, o da kodlanmaz. Şekil 6.3'te bir P çerçevesinin nasıl kodlandığı gösterilmiştir.

Önceki çerçevede uygun bir referans makroblok bulunamazsa, o makroblok bir I çerçevesi makrobloğu gibi kodlanacaktır. Bu da sıkıştırma oranını düşürecektir. Burada önemli olan nokta; "Benzer bir makroblok için daha çok arama yapıp zaman kaybetmeli mi? Yoksa daha az sıkıştırmaya razı olup zamandan tasarruf mu yapmalı?" konusunda karar verilmesidir. Özetle, P çerçevesi kodlanırken aşağıdaki konularda kararlar

verilmeli ve bu kararların sonuçlarını oluşturan bilgiler de kontrol değerleri olarak her blok için saklanmalıdır.

- Resim parçalanırken 8 makroblok tipinden hangisi seçilmeli?
- Her makroblok için, intra kodlama mı yoksa tahminsel kodlama mı yapılmalı? (Intra kodlama seçilirse, o makroblok bir I çerçevesi makrobloğu gibi kodlanacaktır.)
- Aktarım yapılmalı mı? (Eğer niceleme sonucunda bloktaki tüm katsayılar sıfır ise aktarım yapılmasına gerek yoktur.)
- Hangi hareket vektörünü seçmeli?
- Nicelemenin derecesi ne olmalı?



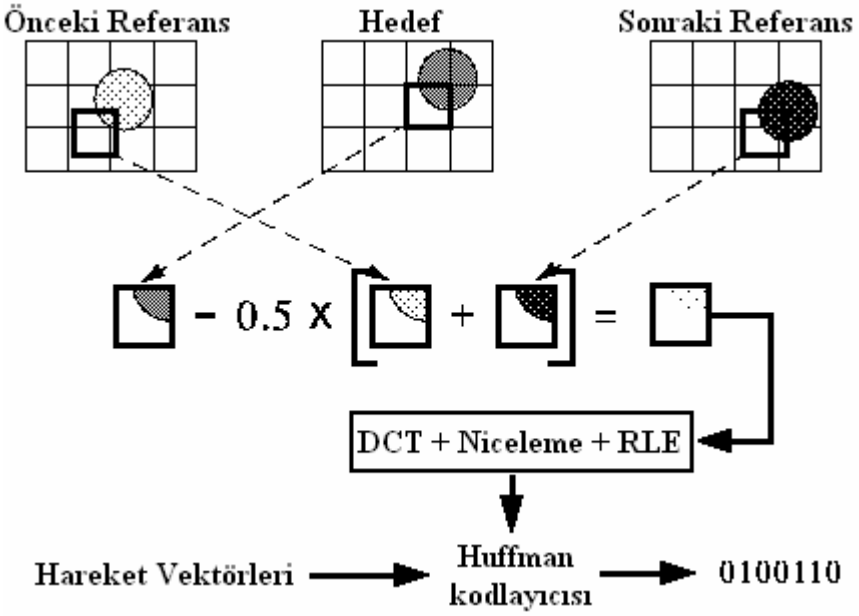
Şekil 6.3. P çerçevesi için kodlama şeması

B çerçevesi (bidirectionally predictive coded frame)

Önceki çerçevede bulunmayan bir nesnenin birden bire ortaya çıkması gibi durumlarda, önceki çerçevede o nesnenin eşleniğini aramak yerine, sonraki çerçevede aramak daha doğru bir yaklaşım olacaktır. Örneğin bir I çerçevesi kodlandıktan hemen sonra bir arabanın kenardan çerçeveye girmesi gibi bir durumda, bir sonraki I çerçevesi kodlanana kadar görüntüde arabanın bulunduğu kısımlar, P çerçeveleri tarafından intra

kodlama ile kodlanmak zorunda kalınacak ve sıkıştırma oranı azalacaktır. P çerçevesi sadece önceki çerçevelerde uygunluk araması yaptığı ve bu nedenle sonraki I çerçevesinde bulunan araba bilgisine ulaşamadığı için, bu gibi durumlarda sonraki çerçevelere de bakabilen B çerçevesinin kullanılması sıkıştırma oranını arttıracaktır.

Eğer hem ileriye hem geriye doğru iki uygun eşlenik makroblok bulunduysa, bu iki makrobloğun ortalaması alınıp, hedef makrobloktan çıkarılarak fark kodlanır. Eğer sadece bir yönde uygun eşlenik makroblok bulunduysa, sadece o makroblok referans olarak kullanılır. Kodlayıcı aynı zamanda, hangi referansı veya referansları kullandığını ve duruma göre bir veya iki hareket vektörünü de bilgi olarak göndermelidir. Şekil 6.4'te bir B çerçevesinin nasıl kodlandığı gösterilmiştir.



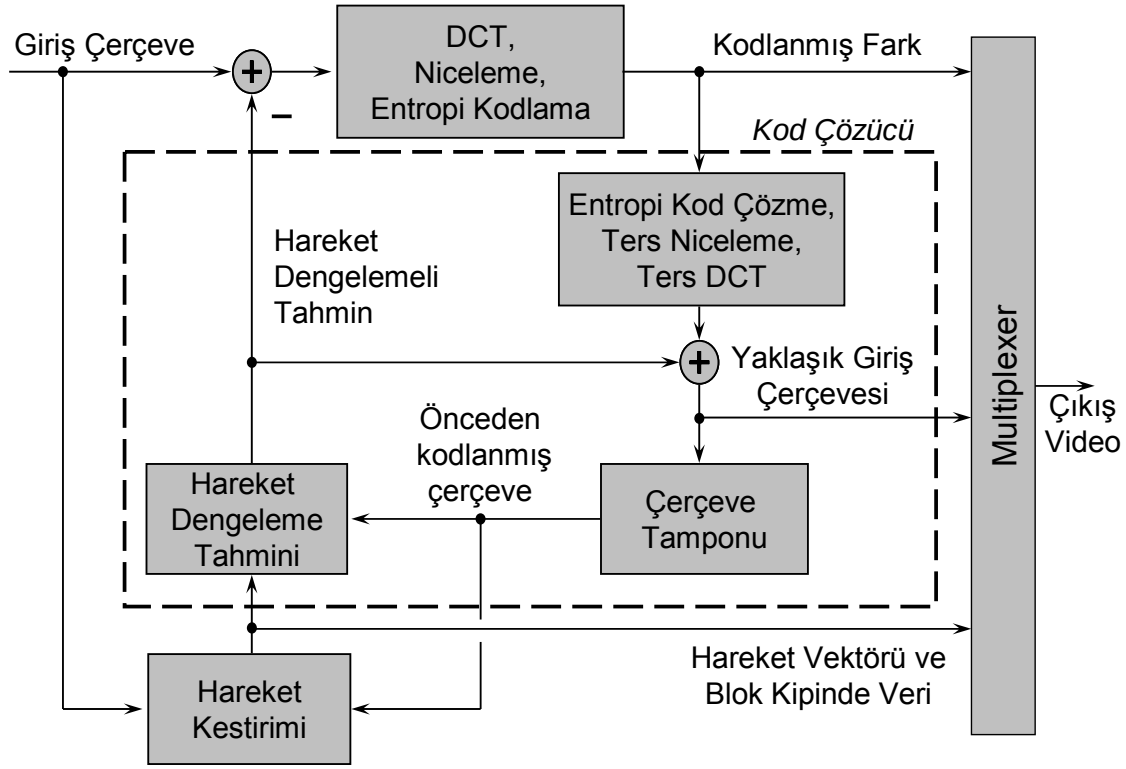
Şekil 6.4. B çerçevesi için kodlama şeması

B çerçevesi üzerinden başka bir çerçeveye yönelik hareket tahmininde bulunulmadığı için, B çerçeveleri daha fazla hataya müsaade edebilir. Bu sebeple, B çerçeveleri için daha az sayıda dilim kullanılması uygundur. B çerçevesi kodlanırken, karar verilmesi gereken noktalar şunlardır:

- Resim parçalanırken 12 makroblok tipinden hangisi seçilmeli? (Geriye dönük hareket vektörü de işin içine girdiği için P çerçevelerine göre daha fazla makroblok tipi vardır.)

- Referans makrobloğu, ileri yönde mi, geri yönde mi, yoksa her iki yönde mi?
- Aktarım yapılmalı mı?

P ve B çerçevelerinin kodlanmasına çerçeveler-arası (inter-frame) kodlama denir. Çerçeveler arası kodlama yapısının akış şeması Şekil 6.5'te verilmiştir. Şekilde görüldüğü gibi, bir sonraki çerçeve için eğer gerekli olacaksa çerçeveler geri dönüşüm ile tekrar elde edilip çerçeve tamponunda saklanmalıdır.



Şekil 6.5. Çerçeveler arası (inter-frame) kodlama akış şeması

Aktarım oranını 1.2 Mbit/s gibi sabit bir hızda tutabilmek için oran kontrolü yapmak gerekir. Eğer görüntüde fazla hareket yoksa sıkıştırma artacak, aktarım oranı düşecektir. Fazla hareket olduğu durumlarda da sıkıştırma azalacak ve aktarım oranı artacaktır. Oranı sabitlemek için niceleme katsayılarının değiştirilmesi gerekebilir. Her makroblok için farklı niceleme katsayılarının kullanılması mümkündür. Eğer aktarım oranını arttırmak gerekliyse, niceleme katsayıları düşürülerek sıkıştırma azaltılır. Aktarım oranını azaltmak gerekliyse de tam tersi yapılır. Bu işlemler için değişken uzunluklu kodlama (VLC) tabloları kullanılır.

MPEG komitesinin MPEG-1 için bildirdiği parametrik değerlere CPB (constrained parameter bitstream) denir. Bu değerler arasında en önemlisi saniyede 10.000 makroblok sınırıdır. Buna göre; 25fps (saniyede 25 çerçeve) oranında 352×288 çözünürlük, 30fps oranında ise 352×240 çözünürlük desteklenmektedir. Çizelge 6.2'de görüldüğü gibi her iki durumda da saniyede 9.900 blok işlenmektedir.

Çizelge 6.2. MPEG-1 için CPB

Çerçeve/Saniye	Video Çözünürlüğü	Makroblok Çözünürlüğü	Makroblok Sayısı	Makroblok/Saniye
25	352×288	22×18	396	9900
30	352×240	22×15	330	9900

MPEG-2

MPEG-2 temelde MPEG-1'in görüntü sıkıştırma yöntemini kullanır. Bununla beraber MPEG-1'nin yetenekleri MPEG-2 için geliştirilmiş, olmayan birçok özellik bu standarda ilave edilmiştir. Bu eklentiler ve iyileştirmeler kısaca aşağıda verilmiştir:

- MPEG-2'de geçmeli tarama (interlace) özelliği desteklenmektedir ve bu özelliğin iki farklı alanı (field) olduğu için, bu iki alanı farklı değerlendiren birçok uyarlanır hareket dengelemesi yöntemi eklenmiştir.
- Daha yüksek veri oranı ve çözünürlüklerde daha iyi sıkıştırma performansı için değişken uzunluklu kodlama tabloları geliştirilmiştir.
- 4:2:2 renklilik desteğinin sağlanması için mevcut makroblok yapısına ilave olarak 8×8 aktarım blokları tanımlanmıştır.
- Videonun hata denetimini iyileştiren DCT katsayıları için veri bölümleme yöntemleri tanımlanmıştır.
- Ölçeklenebilir kodlama yapılabilmesi için çeşitli yöntemler ilave edilmiştir. Eğer MPEG-1 yapısı temel katmanda kullanıldıysa, ölçeklenebilir yöntemler MPEG-1 ve MPEG-2'nin birleşiminin ileri ve geri yönde uyumluluğunu gerçekleştirmek için kullanılabilir.

MPEG-2'de farklı seviyeler ve profiller arasından seçim yapılabilmesine imkân verilmiştir. Profiller kullanılacak algoritmaları belirlerken, seviyeler ise MPEG-1'deki

CPB gibi parametrik değerleri tayin eder. Başlangıçta 5 olan profil sayısı sonradan 4:2:2 ve MVP'nin de eklenmesi ile 7'ye çıkarılmıştır. Hangi profilin hangi seviyede kullanılabildiği (profil@seviye şeklinde ifade edilir) Çizelge 6.3'te 'X' ile gösterilmiştir.

Çizelge 6.3. MPEG-2 Video Profilleri ve Seviyeleri

		PROFİLLER						
		Simple	**Main**	**SNR Scalable**	**Spatial Scalable**	**High**	**4:2:2**	**MVP**
SEVİYELER	**Low** 352×288, 30 fps, 4 Mbit/s		X	X				
	Main 720×576, 30 fps, 15 Mbit/s	X	X	X		X	X	X
	High-1440 1440×1152, 60 fps, 60 Mbit/s		X		X	X		
	High 1920×1152, 60 fps, 80 Mbit/s		X			X		

Minimum kaynak kullanan *Basit* (*Simple*) profilde B çerçeveleri kullanılmaz. Bu açıdan bakıldığında MPEG sıkıştırma yönteminin atası olan ITU-T H.261 ile benzerdir. *Ana* (*Main*) profil, geçmeli tarama desteği dışında MPEG-1 kodlaması ile benzerdir. Ölçeklenebilir kodlamayı destekleyen diğer profiller, ana profilin özelliklerine tamamen sahip olup, bir bit katarının içinde yer alan birden çok katmanı kullanabilme yeteneğine sahiptirler. Temel katman videonun düşük oranda kodlanmış şeklidir. Sadece bu katmanla ilgili bitlerin kodunun çözülmesi ile video eski haline getirilebilir. Fakat daha kaliteli bir görüntü sağlamak için, orijinal görüntü ile temel katmanla olan farkının tekrar kodlanması ile elde edilebilen ikinci bir bit katarı da ilave edilebilir. *SNR Ölçeklenebilir* (*SNR Scalable*) profili böyle bir yaklaşım kullanırken, *Uzamsal Ölçeklenebilir* (*Spatial Scalable*) profili ise hem bu yaklaşımı kullanabilir, hem de örneklemeyi arttırarak (upsampling) yeni bir bit katarı oluşturabilir. *Yüksek* (*High*) profil ise diğer tüm profillerin yeteneklerine sahip olduğu gibi, daha çok bit katarı katmanı kullanabilir ve daha kaliteli görüntü için 4:2:2 YUV alt örneklemesini destekler (diğer profiller sadece 4:2:0 destekler). 4:2:2 alt örneklemesi veri miktarını arttırdığı için, 4:2:2 kullanıldığında maksimum veri oranları da High@Main için 20 Mbit/s, High@High-1440 için 80 Mbit/s ve High@High için 100 Mbit/s olarak belirlenmiştir.

1996 yılının Ocak ayında standarda eklenen 4:2:2 profili, önceki 5 profilin hiyerarşik yapısına uygun değildir. Profesyonel amaçlar için geliştirilmiş olan bu profil,

yüksek profilde olan 4:2:2 YUV alt örneklemesini desteklemesine rağmen, yüksek profilde olan ölçeklenebilir yapılarını kullanmaz. Sadece temel seviyede kullanılabilen 4:2:2 profili, video kalitesinin yüksek olmasını gerektiren durumlarda kullanıldığı için temel seviyenin 15 Mbit/s olan genel veri oranı yerine 50 Mbit/s veri oranını destekler.

1996 yılının Temmuz ayında standarda eklenen *MVP* (*Multi View Profile*) ise, aralarında küçük bir açı bulunan iki kameradan aynı sahnenin çekilmesi durumunda bir kameranın diğerinin görüntüsünü kullanarak *fark dengelemeli tahmin* (*disparity compensated prediction*) yöntemi ile efektif bir şekilde kodlama yapmasını sağlar.

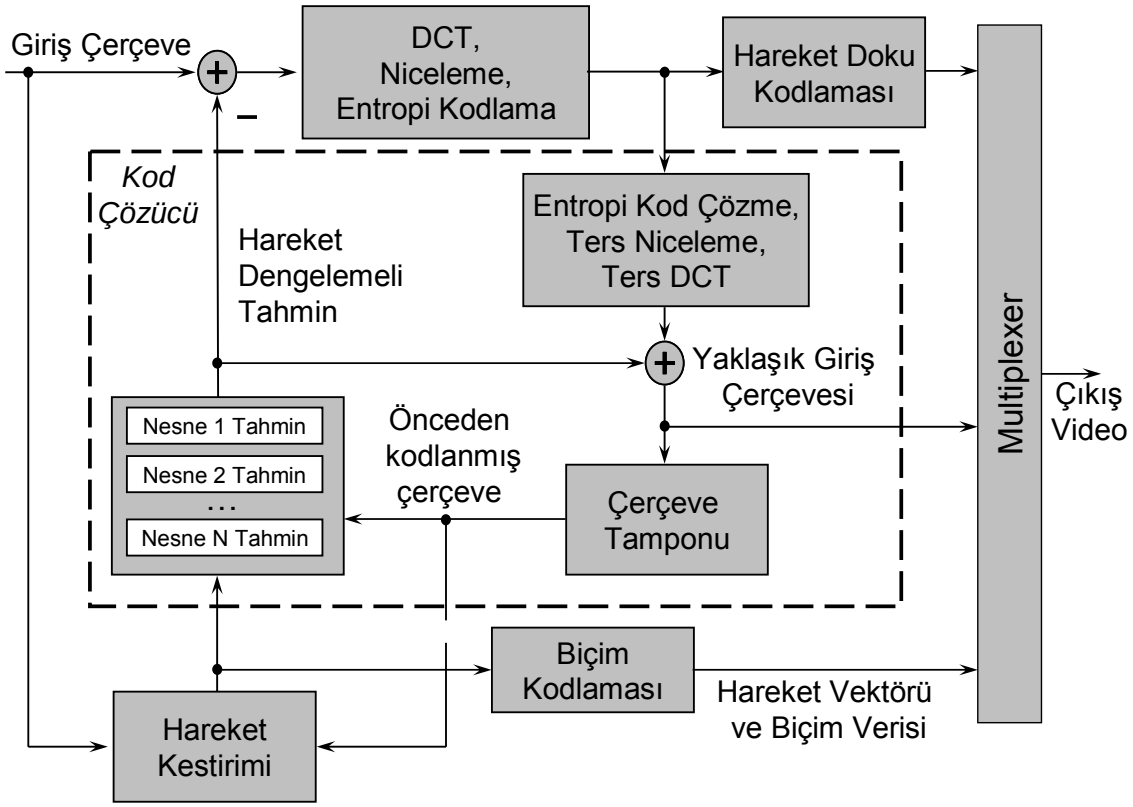
6.2.2. Yeni Yöntem: MPEG-4

ITU-T VCEG'in (Video Coding Experts Group) H.263 standardını geliştirmeye başladığı 1993 yılında, ISO MPEG ise MPEG-4'ü geliştirmeye başlamıştır [Koenen, 2002]. Her iki standart temel yapı olarak birbirine benzer, fakat MPEG-4 daha fazla özelliğe sahiptir. 1999'da ilk sürümü (v1) tamamlan MPEG-4'ün, 2000'de ikinci (v2), 2001'de üçüncü (v3) sürümleri tamamlandı. DivX, XviD, QuickTime, WMV gibi günümüzde en çok kullanılan video sıkıştırma formatları MPEG-4 tabanlıdır.

MPEG-4, hem MPEG-1 ve MPEG-2'dekine benzer şekilde hareket dengelemeli ve DCT-tabanlı sıkıştırma yapabilir, hem de çerçevenin kendi içerdiği nesnelere göre parçalanmasına izin veren nesne-tabanlı kodlamaya izin verir [m_robers-md.pdf]. Nesne-tabanlı kodlamada temel olarak, bir sahne içinde yer alan araba ve insan gibi nesneler arka plandan ayrı birer video nesnesi (video object - VO) olarak düşünülür. Sıkıştırma etkinliğini arttırmak için gerekirse video nesnelerinin her biri için farklı nesne-tabanlı tahmin araçları kullanılıp, farklı niceleme ve kodlama yapılabilir. Şekil 6.6'da MPEG-4'te nesne-tabanlı kodlamanın şeması verilmiştir. Şekilde giriş çerçeve olarak adlandırılan kısım, geleneksel yapıda bir çerçeve olabileceği gibi, dikdörtgensel yapıda bir nesne veya rasgele biçimli bir nesne de olabilir.

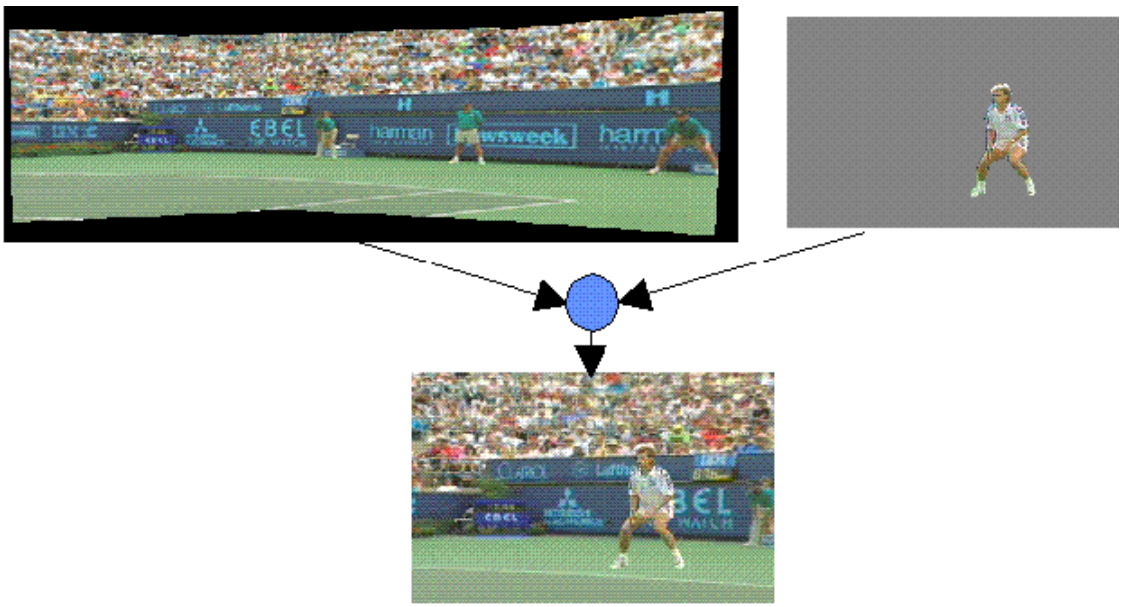
Nesnelerin gösteriminin esnekliğine ve etkin şekilde kodlanmasına izin veren birçok farklı hareket dengeleme teknikleri mevcuttur:

- Hareket arama doğruluğu çeyrek piksele indirgenmiş standart 8×8 veya 16×16 piksel tabanlı hareket kestirim ve dengelemesi. Çeyrek piksel keskinliğinde kestirim yapmak, tahmin hatalarının küçülmesine, ve dolayısı ile görüntü kalitesinin artmasına neden olur.
- Video nesneleri için Global Hareket Dengeleme (GMC): Şekillerin bükülmesi, hareket yörünge kodlanması ve tahmin hataları için doku kodlanması temeline dayanır. Bu teknik ile bir nesne için global hareket küçük miktarda parametreler ile kodlanabilir.
- Panoramik arka plan gibi büyük sabit görüntüler (sprite'lar) için GMC: Bir serideki her ardıl görüntü için kamera hareketini tarif eden sadece 8 global hareket parametresi kodlanır. Bu parametreler ilk çerçevede gönderilen sprite'ın uygun dönüşümlerini gösterir.
- Biçime uyarlanabilir DCT (SA-DCT): Doku kodlaması alanında, rasgele biçimli nesnelerin kodlama etkinliğini arttırır.



Şekil 6.6. MPEG-4'te nesne-tabanlı kodlama

Şekil 6.7'de panoramik arka plan görüntü kullanan bir MPEG-4 video serisinin temel yaklaşımı gösterilmiştir. Ön plandaki tenis oyuncusu haricinde kalan arka plan görüntüsü (sprite) sadece ilk çerçevede gönderilecektir. Daha sonraki çerçevelerde ise sadece tenis oyuncusunun hareketleri nesne-tabanlı olarak kodlanacak ve kameranın hareketleri de parametre olarak gönderilecektir. Kod çözücü panoramik görüntüyü diğer çerçeveler için sprite tamponundan elde eder ve kamera hareketlerini ifade eden parametrelerin yardımı ile ilgili yere yerleştirilir. Ön plandaki tenis oyuncusunun nesne tabanlı kodlanmış olan verilerinin de kodunu çözerek arka plan görüntüsünün üzerine yerleştirir. Gecikme miktarının düşük olması isteniyorsa sprite bilgisi ilk çerçeve yerine parçalara ayrılarak ilk birkaç çerçevede gönderilebilir.



Şekil 6.7. Panoramik görüntü ile kodlama

MPEG-4 dokuları ve sabit görüntüleri kodlarken dalgacık dönüşümü tabanlı bir algoritma kullanır. Dalgacık dönüşümü, sadece daha iyi sıkıştırma etkinliği sağladığı için değil, aynı zamanda uzamsal ölçeklenebilirlik için gerekli olan katmanlı yapıya uygun bir yöntem olduğu için tercih edilmiştir. Yüksek derecede ölçeklenebilirlik ihtiyacı, daha çok iki veya üç boyutlu bir nesnenin statik bir doku ile kaplanması istenilen durumlarda gereklidir. MPEG-4, bu gibi durumlarda kaplanacak olan statik dokuyu 11 katmana kadar çıkabilen uzamsal ölçeklenebilirliği destekleyen ayrık dalgacık dönüşümü ile kodlar.

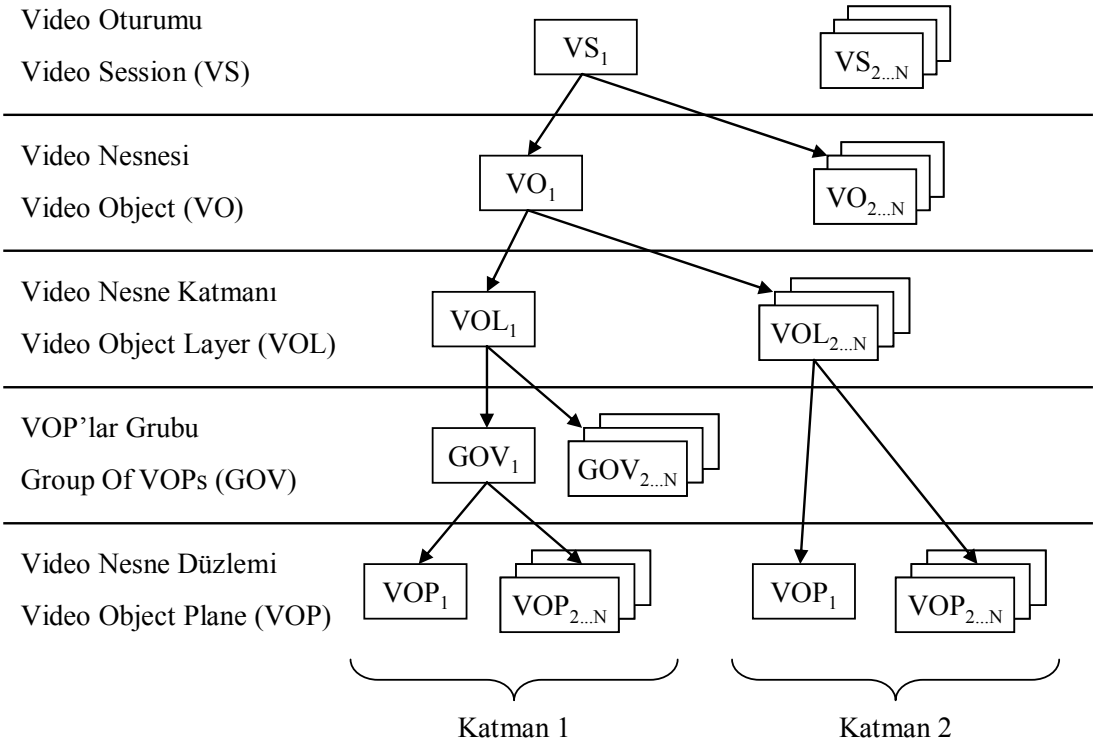
MPEG-4'te, hem geleneksel çerçeve yapısı hem de nesneler için, uzamsal ölçeklenebilirlik (çerçeve/nense gösterim oranının değiştirilmesi) ve zamansal ölçeklenebilirlik (bir saniyede gösterilen çerçeve/nesne sayısının değiştirilmesi) desteklenmektedir. Ayrıca öğe boyu ölçeklenebilirliği (Fine Granularity Scalability - FGS) gibi faklı seçenekler de yer almaktadır. Ölçeklenebilir yapı sayesinde, birçok farklı bant genişliği ve/veya hesaplama karmaşıklığı ihtiyaçları tek bir MPEG-4 video bit katarı ile karşılanabilir. Kod çözücü ölçeklenebilir yapıda oluşturulmuş olan bir MPEG-4 bit katarını açarken, bit katarının sadece belirli kısımlarını ters kodlayarak, uzamsal çözünürlüğü, zamansal çözünürlüğü veya her ikisini birden azaltabilir. İşlemci gücünün tüm bit katarının kodunu çözebilecek kadar yeterli olmadığı veya görüntüleme aygıtının çözünürlüğünün düşük olduğu durumlarda (örneğin taşınabilir cihazlarda), ya da bant genişliğinin yetersiz olduğu durumlarda böyle bir ölçeklenebilirlik yaklaşımı gerekli olabilmektedir.

Görüntü kalitesini düşürmeden zamansal ölçeklenebilirlik yapabilmek için, arka planın çerçeve oranı ile ön plandaki nesnenin çerçeve oranı farklı belirlenebilir. Bu sayede 30fps hızında bir video gösteriminde ön plandaki nesne her çerçevede gösterilirken, arka plan görüntüsü ölçekleme oranına göre belirli aralıklarla (örn. 5., 10., 15. … çerçevelerde) gösterilebilir. Internet üzerinde video akışının kesintisiz olması için geliştirilen, düşük bit oranlarında çalışan, hem kendi başına hem de zamansal ölçeklenebilirlik ile birlikte kullanılabilen FGS, temel katmandan tahmin yolu ile ikinci katmanı oluşturur [Peng, 2004].

Nesne-tabanlı uzamsal ölçeklenebilirlik sayesinde, tüm çerçeve yerine sadece ilgili nesnenin çözünürlüğü değiştirilebilir. Nesne-tabanlı uzamsal ölçeklenebilirliğin desteklenmesi için, bir video nesnesi birden çok katmandan (*multi-layer*) oluşmalıdır. Bir video nesnesinin sadece bir *Video Nesne Katmanı* (*Video Object Layer - VOL*) ile kodlandıysa ölçeklenemez yapıdadır. İki tip VOL vardır; Tam MPEG-4 işlevselliğine sahip VOL, ve H.263 ile uyumlu olan işlevselliği azaltılmış VOL.

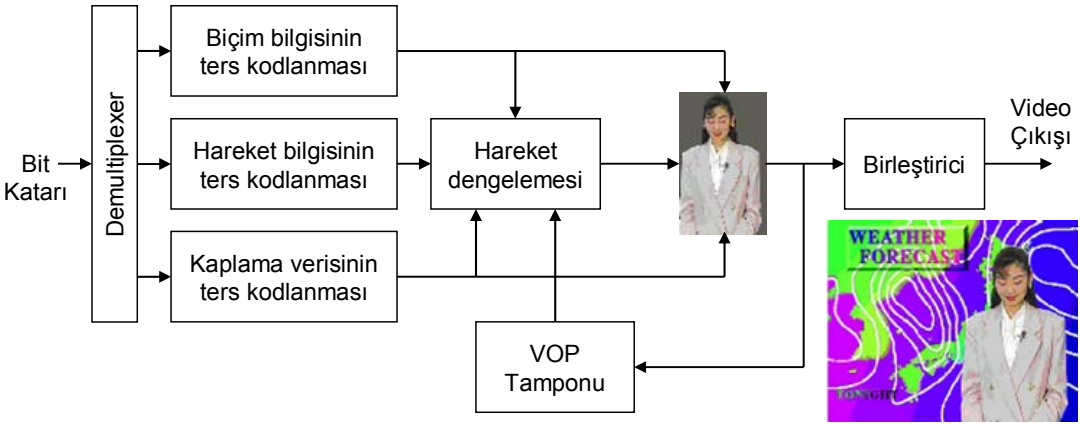
Tüm video nesneleri zamanda örneklenir ve her bir örneğine *Video Nesne Düzlemi* (*Video Object Plane - VOP*) adı verilir. VOP'lar I çerçevesinde birbirinden bağımsız olarak, hareket dengelemesi yapılan P ve B çerçevelerinde ise birbirlerine

bağımlı olarak kodlanırlar. Geleneksel video çerçevesi, dikdörtgen biçimli bir VOP ile temsil edilebilir. VOP'lar istenirse gruplandırılabilir (*Group Of VOPs - GOV*). MPEG-4 bit katarının mantıksal hiyerarşik yapısı Şekil 6.8'de gösterilmektedir.



Şekil 6.8. MPEG-4 bit katarının mantıksal yapısı

En küçük mantıksal birim olan VOP'lar, en küçük fiziksel kodlama birimi olan makrobloklar ile kodlanırlar. Her VOP, biçim bilgisi, hareket parametreleri ve kaplama verisinden oluşur. Şekil 6.9'da MPEG-4'ün kod çözme şeması gösterilmiştir. Geriye doğru uyumluluğun ve etkin kodlamanın sağlanması için, video nesneleri kendi video nesne düzlemleri içinde önceki MPEG standartlarına benzer bir yapı ile kodlanırlar.



Şekil 6.9. MPEG-4'te nesne-tabanlı kod çözme

MPEG-4'ün getirdiği en büyük yeniliklerden biri de, mobil iletişim gibi hata olasılığı yüksek ortamlarda çok gerekli olan, hata giderici yaklaşımları ihtiva etmesidir. Bu yaklaşımlar, tekrar eşleme (resynchronization), veri kurtarma (data recovery) ve hata gizleme (error concealment) olarak üç grupta toplanabilir.

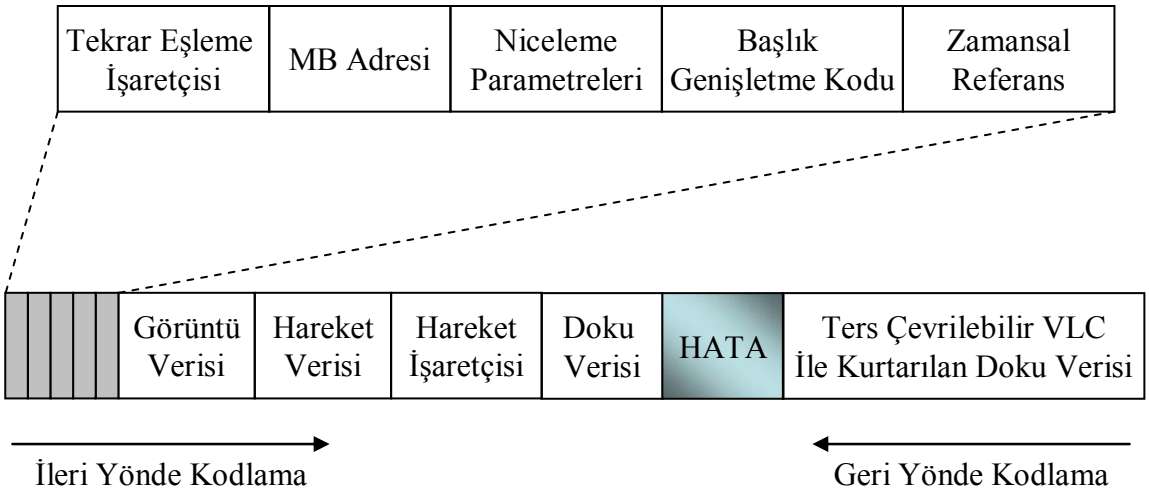
En çok kullanılan yöntem olan tekrar eşleme, bit katarının belirli noktalarına tekil işaretçiler yerleştirilmesi ve hata olması durumunda, bir sonraki işaretçiye kadar olan tüm bitlerin yok sayılıp, bir sonraki işaretçiden tekrar kod çözmenin yapılmaya başlanması prensibine dayanır. MPEG-2 ve H.263'te belirli sayıda makrobloktan oluşan blok grubunun başlığı bir işaretçi olarak kabul ediliyordu. Bu yaklaşım, değişken oranlı kodlama yapıldığı durumlarda işretçilerin birbirine olan uzaklıkları eşit olmadığı için problem yaratmaktaydı. Hareketliliğin yoğun olduğu bölümlerde hataları gizlemek daha zor olmasına rağmen, bu bölümler daha çok bit ile kodlanacağı için, bir sonraki işaretçi daha uzakta kalıyor, ve bu bölümler hataya karşı daha hassas hale geliyordu. MPEG-4 kullanılan video paketi sayısına göre işaretçi belirleme yaklaşımı, işretçilerin bit katarında birbirine eşit uzaklıkta olmasını sağlamaktadır.

Veri kurtarma araçları aslında hatayı tam olarak gidermek yerine, hatanın etkisini azaltmayı amaçlar. En çok kullanılan veri kurtarma aracı ters çevrilebilir değişken uzunluklu kodlardır (Reversible Variable Length Codes - RVLC). Bir hata ile karşılaşıldığında bir sonraki işaretçiye kadar olan tüm veriyi kaybetmek yerine, bir

sonraki işaretçiden itibaren geriye doğru ters kodlama yapan RVLC kullanılarak hatanın etkisini azaltmak mümkündür.

Hata gizleme yöntemleri, önceki çerçeveden blokların kopyalanması veya veri bölümlendirme gibi farklı türlerde olabilmektedir. Hareket bilgisi ve doku bilgisinin farklı saklanması ve aralarında bir işaretçi bulunması ile yapılan bir veri bölümlendirme, doku bilgisinde bir hata olması durumunda, hareket verisi ve daha önce kodlanmış olan VOP'un kullanılarak bir hareket dengelemesi yapılması ile hatayı gizleyebilir.

Anlatılan hata giderici yaklaşımlar Şekil 6.10'da gösterilmiştir. Şekilde yer alan başlık genişletme kodu, sıkıştırılan videonun kodunun düzgün olarak çözülebilmesi için gerekli olan başlık bilgisinin bozulması gibi durumlarda kullanılabilecek fazladan bir başlık bilgisidir.



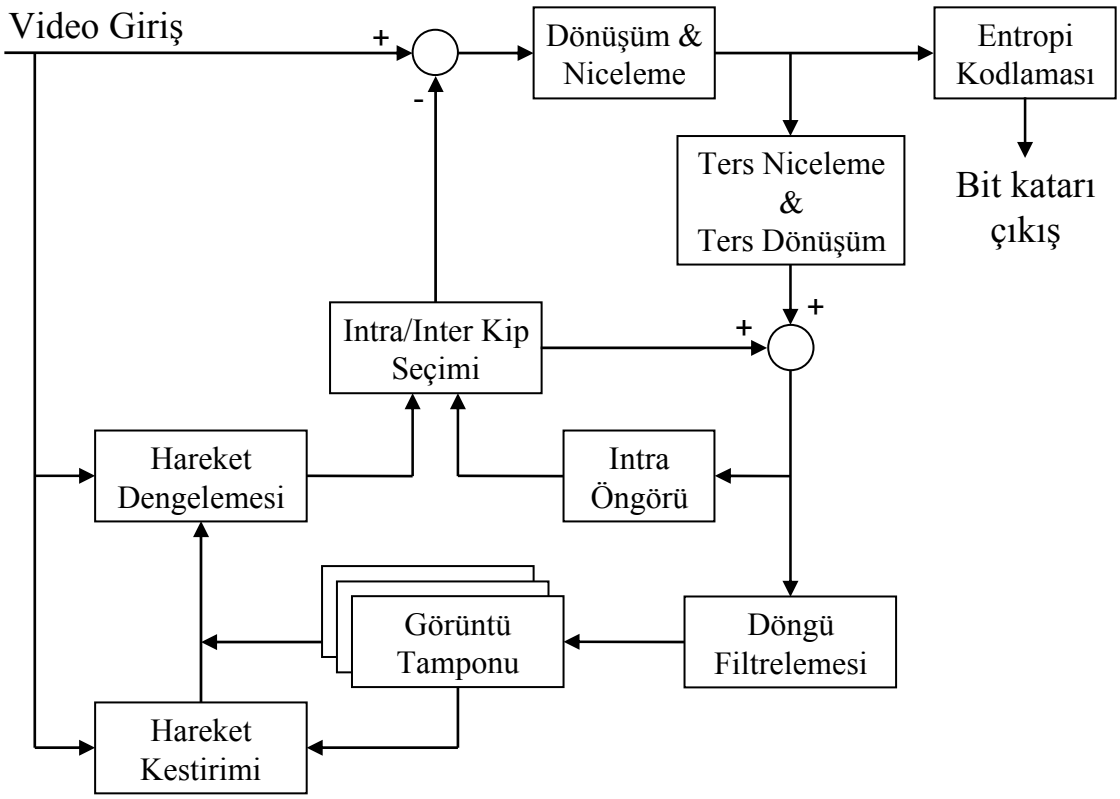
Şekil 6.10. MPEG-4'te kullanılan hata giderici yaklaşımlar

MPEG-4 standardında doğal nesneler dışında yapay nesnelere de yer verilmiştir. Sadece dikdörtgen ve küre gibi iki ve üç boyutlu temel nesnelerin değil, insan yüz ve vücudunun da tanımlama parametreleri ve animasyonları için gerekli dönüşüm bilgileri yer almaktadır [Tekalp, 2000]. Sıkıştırma açısından bakıldığında, bu nesnelerin ağ örgüsünün tanımları ve kaplamaları daha az bit ile ifade edilebileceği için, resim çerçevesi biçimine dönüştürülerek kodlanmasındansa, ağ örgüsünün parametrelerinin kodlanması daha etkili olmaktadır.

AVC (ITU-T H.264, ISO/IEC MPEG-4 Part 10)

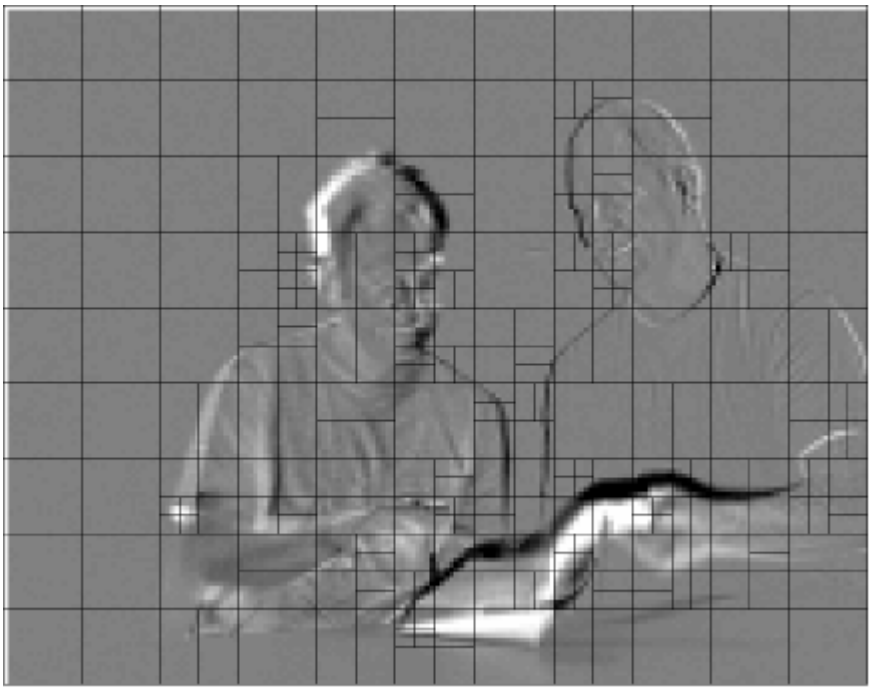
1997 yılında ITU-T VCEG H.26L hareketli görüntü sıkıştırmasını geliştirmeye başladı ve 1999 yılında ilk test sürümü tamamlandı. 2001 yılında ISO MPEG ile ITU-T VCEG gruplarından uzmanların bir araya gelmesiyle JVT (Joint Video Team) kuruldu, ve ortak bir standart geliştirilmeye başlandı. Bu standart, ITU-T tarafında H.264 (eski adı H.26L) [ITU, 2003], ISO tarafında ise MPEG-4 Part 10 (ISO 14496-10) [ISO, 2004] olarak isimlendirilmiştir. MPEG, daha önce geliştirmiş olduğu ve MPEG-4'te de kullanılan AAC kayıplı ses sıkıştırma standardının video karşılığı gibi düşünerek sonradan AVC (Advanced Video Coding) ismini kullanmaya başlamıştır. MPEG-4 Part 2, AVC'nin geliştirilmesinden sonra ASP (Advanced Simple Profile) olarak bilinmektedir.

Bu bölümde önceki standartlarda olmayıp AVC'de olan yeni özelliklerden bahsedilecektir. AVC kodlayıcısının blok şeması Şekil 6.11'de verilmiştir.



Şekil 6.11. AVC kodlayıcısının blok şeması

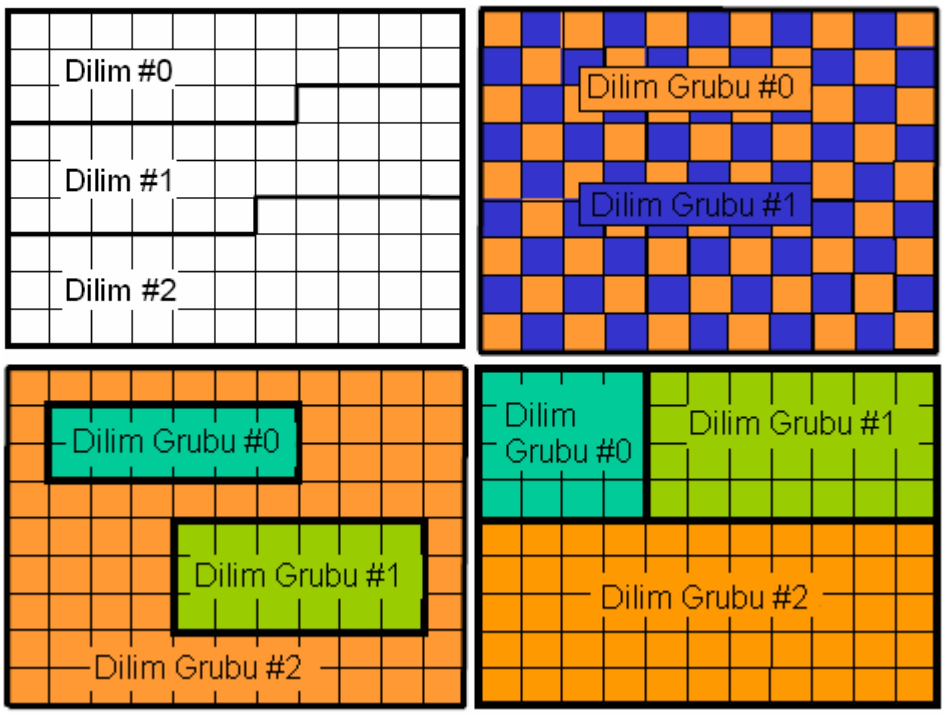
Değişken Blok Boyları / Makroblok Bölümleri: Blok büyüklükleri sadece 16x16 veya 8x8 olabilen ASP'nin tersine, AVC'deki Hareket Arama Kesinliği (Motion Search Precision) özelliği sayesinde makroblok bölünmesi 4×4 veya 8×4 gibi daha küçük ve değişken büyüklüklerde olabilmektedir. Her makroblok için en uygun bölünmeye kodlayıcı karar verir. Örneğin Şekil 6.12'de yer alan fark çerçevesinde, kodlayıcı renk farklılığı yüksek olan kesimlerde 4×4, 8×4 veya 4×8 gibi küçük alt bölümler seçerken, az değişkenlik olan bölümlerde 8×8, 16×8 veya 8×16 gibi daha büyük alt bölümler, rengin sabit kaldığı kesimlerde ise 16×16 büyüklüğünde bölümler seçmiştir. Böylece, hareket vektörlerinin ve kodlanan farkın minimum düzeyde kalması sağlanmıştır.



Şekil 6.12. Makroblokların bölümlenmesi

FMO (Flexible Macroblock Ordering): Bir makroblok serisi olan dilimler içerik olarak bağımsızdırlar. Yani bir dilimin kodlanması için diğer dilimlerde olan herhangi bir bilgiye ihtiyacı yoktur. Fakat döngü filtrelemesi diğer dilimlerdeki bilgiye ihtiyaç duyabilir. Dilimler ya eski MPEG standartlarında olduğu gibi soldan sağa tarama düzeninde makroblokları birleştirirler, ya da *Esnek Makroblok Düzenlemesi* (FMO) kullanılarak içeriklerine göre dilim grupları haline getirilirler. Bu dilim grupları MPEG-4 standardının nesnelerine karşılık gelir. Dilim veya dilim grupları bağımsız olarak

kodlanabildikleri için çerçeve gibi de düşünülebilirler. Şekil 6.13'te dilimlerin tarama düzeninde seçilmesi ve gruplandırılarak belirlenmesi yöntemleri gösterilmiştir.



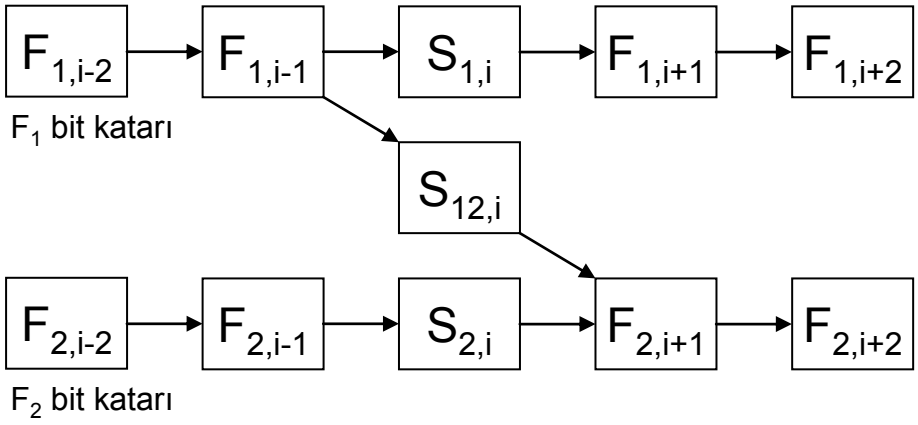
Şekil 6.13. Dilim ve dilim gruplarının seçimi

ASO (Arbitrary Slice Ordering): Bir dilimin ilk makrobloğunun adresi, her zaman sol üst köşedeki makroblok olmak zorunda değildir. Aynı resim içinde, farklı dilimler farklı ilk makroblok adresleri kullanabilirler. Bu özelliğe *Rasgele Dilim Düzenleme* (ASO) denir.

RS (Redundant Slice): Bir dilimin daha önce kodlanmış verileri ile karşılaştırma yapıldığı zaman arada fazla farklılık olmadığı görülürse, bu dilim *Gereksiz Dilim* (RS) olarak adlandırılır. Eğer veri oranı düşürülmek istenirse gereksiz dilimler kodlanmadan sadece diğer dilimler kodlanabilir.

Çok Referanslı Çerçeveler: ASP ve daha önceki MPEG standartlarında olduğu gibi sadece bir önceki çerçeveye değil, daha önceki çerçeveler de referans çerçeve olarak kullanılabilir. B çerçevesine benzer şekilde kodlanan çok referanslı çerçevelerde, kaç çerçeve öncesi referans alındıysa, bu değer kodlamaya dâhil edilmelidir. Bu yaklaşım, kodlanması gereksiz olan çerçevelerin atılmasına yardım eder.

Geçiş Çerçeveleri (veya Dilimleri): AVC standardının genişletilmiş profilinde I, P ve B çerçevelerinin dışında iki adet geçiş (switch) çerçevesi yer almaktadır. SP ve SI çerçeveleri, birbirine benzer video'lar arasında geçiş yapılırken kullanılırlar. Aralarındaki fark SP'nin P çerçevesi gibi, SI'nın ise I çerçevesi gibi kodlanmasıdır. S çerçeveleri, bit katarlarını birbirine ekleme veya birinden diğerine geçme, hızlı ileri ve geri sarma, rasgele erişim, hata giderme gibi işlemlerde kullanılırlar. Şekil 6.14'te geçiş çevrelerinin kullanımı gösterilmiştir.



Şekil 6.14. Geçiş çerçevelerinin kullanımı

Ağırlıklı Öngörü: Ağırlıklı öngörü ile referans çerçeveye ağırlık uygulanabilir. Özellikle parlaklığın azaldığı sahne sonları gibi durumlarda, bir sonraki çerçeve daha karanlık olması dışında bir önceki çerçeve ile çok benzer olduğu için, bu tür durumlarda kullanılması elverişlidir. P veya B dilimlerindeki hareket dengelemeli öngörü verilerinin örneklerine bir ağırlık çarpanı uygulanarak işlem yapıldığı için ağırlıklı öngörü denilmiştir.

Döngü (bloklamayı azaltma) Filtrelemesi: Döngü filtrelemesi (loop - deblocking filter), kodlayıcıda ön işlem, kod çözücüde de son işlem olarak kullanılan filtrelemeden farklı olarak, kodlama aşamasında her çerçeve üzerinde gerçekleştirilir. Bir çerçeve kodlandıktan sonra, sonraki çerçevelere referans olmadan önce gerçekleştirildiği için, özellikle düşük veri oranlarında ortaya çıkan blok kenarlarının belirginleşmesini azaltır. Şekil 6.15'te görüldüğü gibi resim kalitesini arttıran döngü filtrelemesi, kodlama ve kod çözme işlemlerini yavaşlatır.

Filtrelemesiz *Döngü Filtrelemeli*

Şekil 6.15. Geçiş çerçevelerinin kullanımı

Oran bozulma iyileştirmesi (Rate Distortion Optimisation - RDO): RDO, farklı seçenekler olduğu durumlarda (inter/intra, hareket arama, ... gibi) kodlayıcının en etkili kodlama kararlarını vermesini sağlar. RDO, AVC standardında tanımlanmış bir araç değildir, sadece ilk defa H.264 referans yazılımı tarafından kullanılmış yeni bir karar verme yaklaşımıdır. XviD gibi başka kodlayıcılar da RDO kullanır.

Tamsayı Dönüşüm: DCT dönüşümünün irrasyonel sayılar oluşturduğu ve bu sayıların da hesaplamayı yavaşlattığı bilinmektedir. AVC'de, DCT yerine kodlama etkinliğini arttırmak için tamsayı dönüşüm işlemi uygulanır [Malvar vd., 2003]. Bu dönüşüm işlemi için 4×4 ve 8×8 büyüklüğünde iki farklı matris belirlenmiştir [Sullivan vd., 2004]. Eğer makroblok 4×4 büyüklüğünde bloklara bölünürse 4×4'lük dönüşüm matrisi, 8×8 büyüklüğünde bloklara bölünürse 8×8'lik dönüşüm matrisi (sadece yüksek profilde) kullanılır. Bu matrisler aşağıda verilmiştir:

$$T_{4\times4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \quad T_{8\times8} = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & 10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix}$$

Hadamard Dönüşümü: Eğer 16×16 büyüklüğünde makroblok ve 4×4 tamsayı dönüşümü kullanıldıysa, 16 adet 4×4 büyüklüğünde parlaklık bloğunun ve 2×2 büyüklüğünde renklilik bloğunun DC katsayıları için ikinci bir dönüşüm daha kullanılır. Yukarıda bahsedilen tamsayı dönüşümüne çok benzer olan bu dönüşüm Hadamard dönüşümü olarak bilinir.

$$H_{2\times 2} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad H_{4\times 4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

Niceleme işlemi temel olarak önceki standartlarda kullanılan niceleme işlemine benzer yapıdadır. Değerleri 0.625 ile 224 arasında değişen 52 farklı niceleme adım büyüklüğü tanımlanmıştır. Adım büyüklükleri niceleme parametresinin 6 artması ile 2 katına çıkacak şekilde logaritmik bir sisteme oturtulmuştur. Örneğin 24. parametre 10 değerine sahipken, 30. parametrenin değeri 20'dir.

Entropi Kodlaması aşamasında, dönüşüm katsayıları haricinde tüm semboller için Exp-Golomb kodlaması kullanılır. Dönüşüm katsayıları için ise, içeriğe uyarlanabilir entropi kodlaması kullanılır. Exp-Golomb kodları, düzgün bir yapıya sahip olan değişken uzunluklu kodlardır. Yapısı [M adet sıfır][1][Bilgi] şeklinde olan Exp-Golomb kodlarının ilk 9 tanesi Çizelge 6.4'te verilmiştir.

Çizelge 6.4. Exp-Golomb kodları

Kod Numarası	Kod
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
…	…

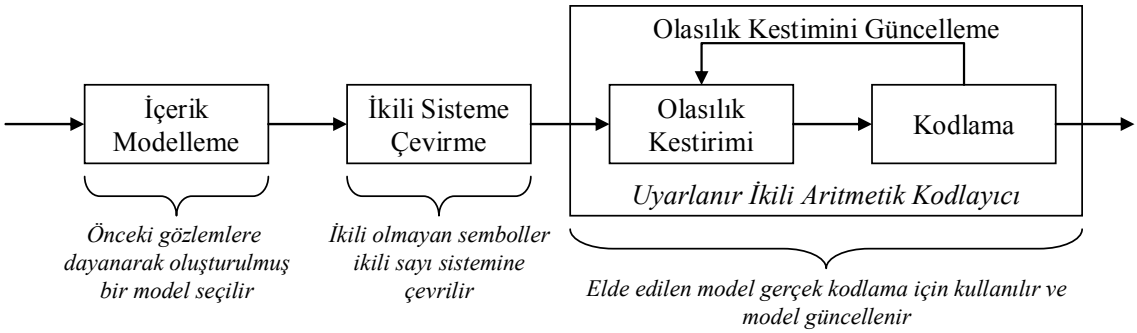
AVC standardında, dönüşüm katsayılarını kodlamak için, ASP'de olmayan iki farklı entropi kodlaması yöntemi tanımlanmıştır; CAVLC ve CABAC.

CAVLC (Context-Adaptive Variable Length Coding): Varsayılan yöntem olan CAVLC, 0 ve +/-1 katsayılarını diğer katsayılardan farklı bir biçimde kodlar. İşlem adımları aşağıda verilmiştir:

1. Sıfır olmayan katsayıların toplam sayısını ve +/-1 değerlerini kodla.
2. Her 1'in işaretini ters sırada kodla.
3. Diğer sıfırdan farklı katsayıların derecelerini ters sırada kodla.
4. Son katsayıdan önceki sıfırların sayısını kodla.
5. Tüm ardıl sıfır koşularını kodla.

AVC'de, 4'ü katsayıların sayısı için ve 7'si katsayılar için olmak üzere toplam 11 farklı kod seti yer almaktadır. Belirli bir içeriğe uygun kod seçildiği için *İçeriğe Uyarlanır VLC* (CAVLC) adını almıştır.

CABAC (Context-Adaptive Binary Arithmetic Coding): Her sembol için olasılık modelini güncelleyen bu yöntem CAVLC'den daha yavaştır, ama yüksek bit oranlarında %10-15 oranında daha iyi sıkıştırma sağlar. Bu yöntemin blok şeması Şekil 6.16'da verilmiştir.



Şekil 6.16. CABAC yönteminin blok şeması

AVC Standardında 4 farklı profil yer alır: Temel (Baseline), Ana (Main), Genişletilmiş (Extended) ve Yüksek (High). Bu profillerin özellikleri Çizelge 6.5'te verilmiştir.

Çizelge 6.5. AVC profillerinin özellikleri

Özellikler	Temel	Ana	Genişletilmiş	Yüksek
I & P Çerçeveleri	✓	✓	✓	✓
Döngü Filtrelemesi	✓	✓	✓	✓
¼ Pel Hareket Dengeleme	✓	✓	✓	✓
Değişken Blok Büyüklüğü (4×4 ile 16×16 arasında)	✓	✓	✓	✓
CAVLC	✓	✓	✓	✓
CABAC		✓		✓
Hata Giderme Araçları FMO & ASO & RS	✓		✓	
Geçmeli Tarama		✓	✓	✓
Ağırlıklı Öngörü		✓	✓	✓
B Çerçevesi		✓	✓	✓
SP & SI Çerçeveleri			✓	
Veri Bölümleme			✓	
Uyarlanır Dönüşüm 4×4 veya 8×8				✓
Kayıpsız Sıkıştırma				✓
Özel Niceleme Katsayıları				✓
4:2:2 ve 4:4:4 YUV				✓
10 bit renk desteği				✓

En düşük karmaşıklığa sahip olan temel profil, görüntülü konferans ve mobil video uygulamaları için uygundur. Ana profil, televizyon yayınları ve DVD gibi video saklama yapıları için düşünülmüştür. Hataların en yoğun olduğu ağ üzerinden akıcı video gösterimi için, hataya karşı en tutarlı olan genişletilmiş profili kullanmak uygundur. Yüksek profil ise, kayıpsız sıkıştırma özelliği ve diğer kaliteyi arttırıcı seçenekleri ile, sinema çekimi gibi profesyonel uygulamalar için tasarlanmış bir profildir.

6.2.3. MPEG Hareketli Görüntü Kodlayıcılarının Özelliklerinin Karşılaştırılması

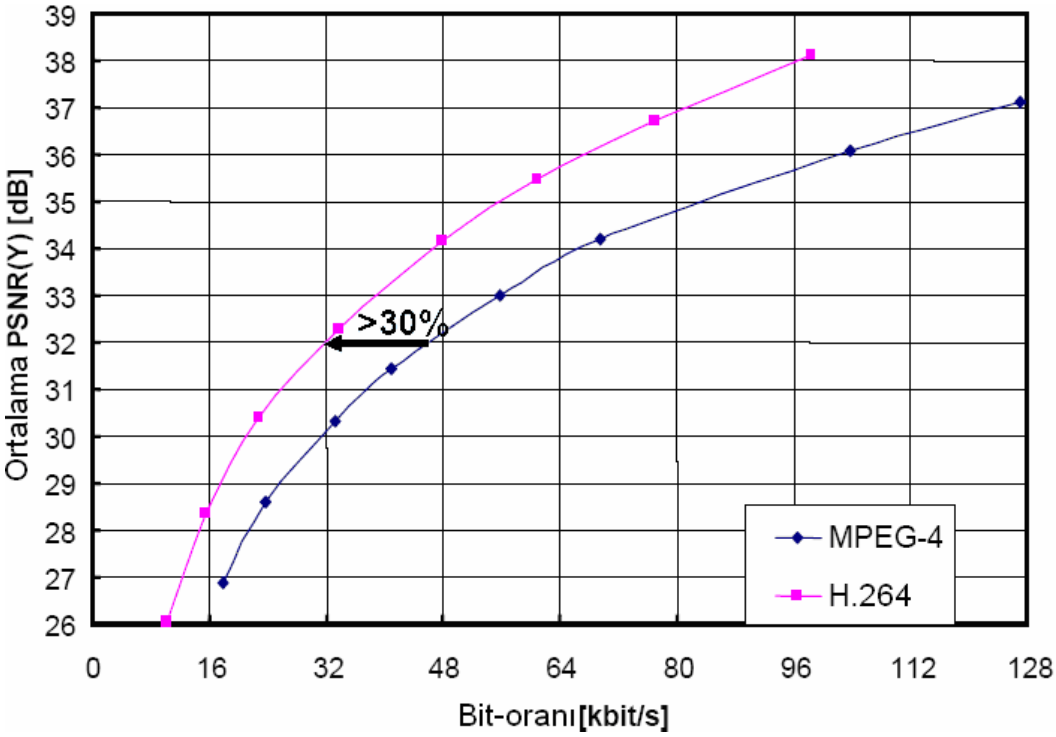
Tüm MPEG kodlayıcılarının özelliklerinin bir arada görülebilmesi amacıyla Çizelge 6.6'da MPEG-1, MPEG-2, MPEG-4 Part 2 ve MPEG-4 Part 10 kodlayıcılarının özellikleri yan yana verilmiştir.

Çizelge 6.6. MPEG hareketli görüntü kodlayıcılarının özellikleri

Özellikler	MPEG-1	MPEG-2	MPEG-4 Part 2	MPEG-4 Part 10
Makroblok Büyüklüğü	16×16	16×8 (geçmeli tarama), 16×16	16×16	16×16
Blok Büyüklüğü	8×8	8×8	16×16, 16×8, 8×8	16×16, 16×8, 8×16, 8×8, 8×4, 4×8, 4×4
Uzamsal Dönüşüm	DCT 8×8	DCT 8×8	DCT 8×8	Int DCT 4×4, 8×8 Hadamard 2×2, 4×4
Niceleme	Sayıl	Sayıl	Vektör	Sayıl
Entropi Kodlaması	Huffman	Huffman	Huffman	Exp-Golomb, CAVLC, CABAC
Arama Keskinliği	1 ve ½ pel	½ pel	1, ½ ve ¼ pel	1, ½ ve ¼ pel
Profil Sayısı	yok	6	8	4
Çerçeve Tipleri	I, P, B	I, P, B	I, P, B	I, P, B, SI, SP
Çok Referanslı Çerçeveler				✓
Geçmeli Tarama		✓	✓	✓
Döngü Filtrelemesi				✓
Ağırlıklı Öngörü				✓
Kayıpsız Sıkıştırma				✓
GMC			✓	
RDO				✓
Veri Oranı	< 1.5 Mbit/s	2 Mbit/s – 15 Mbit/s	64 Kbit/s – 4 Mbit/s	64 Kbit/s – 240 Mbit/s

6.2.4. Kayıplı Hareketli Görüntü Sıkıştırma Yöntemlerinin Karşılaştırılması

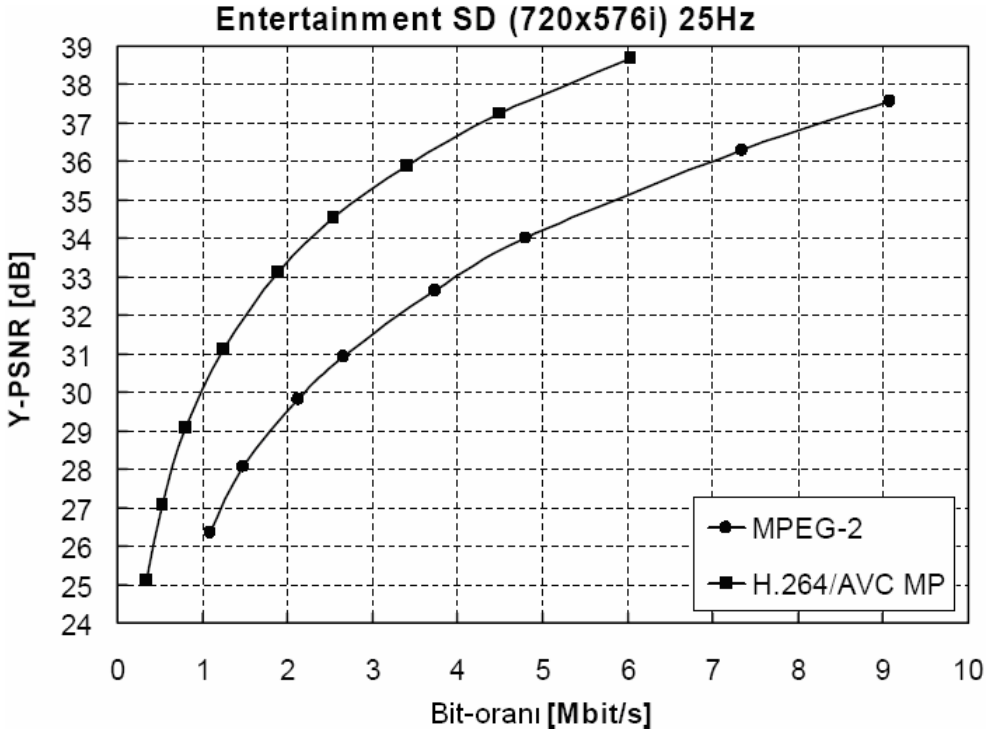
Kayıplı hareketli görüntü sıkıştırma yöntemleri karşılaştırılırken, genellikle çerçevelerin sadece aydınlık (Y) bilgisi kullanılarak PSNR değerleri ölçülür ve bu değerlerin ortalaması alınır. Şekil 6.17'de ITU-T VCEG tarafından yapılan, görüntülü konferans gibi uygulamalar için gerekli olan düşük bit oranlarında MPEG-4 ASP ve MPEG-4 AVC yöntemlerinin elde ettikleri sonuç gösterilmektedir. Şekil 6.15'te bir çerçevesinin karşılaştırıldığı *Foreman* isimli örnek hareketli görüntünün 10 Hz frekansında ve QCIF çözünürlüğünde kodlandığı teste göre, AVC ve ASP ile bir hareketli görüntü aynı kalitede kodlandığında, AVC'nin ASP'ye göre %30 daha düşük bit oranı kullandığı görülmektedir.



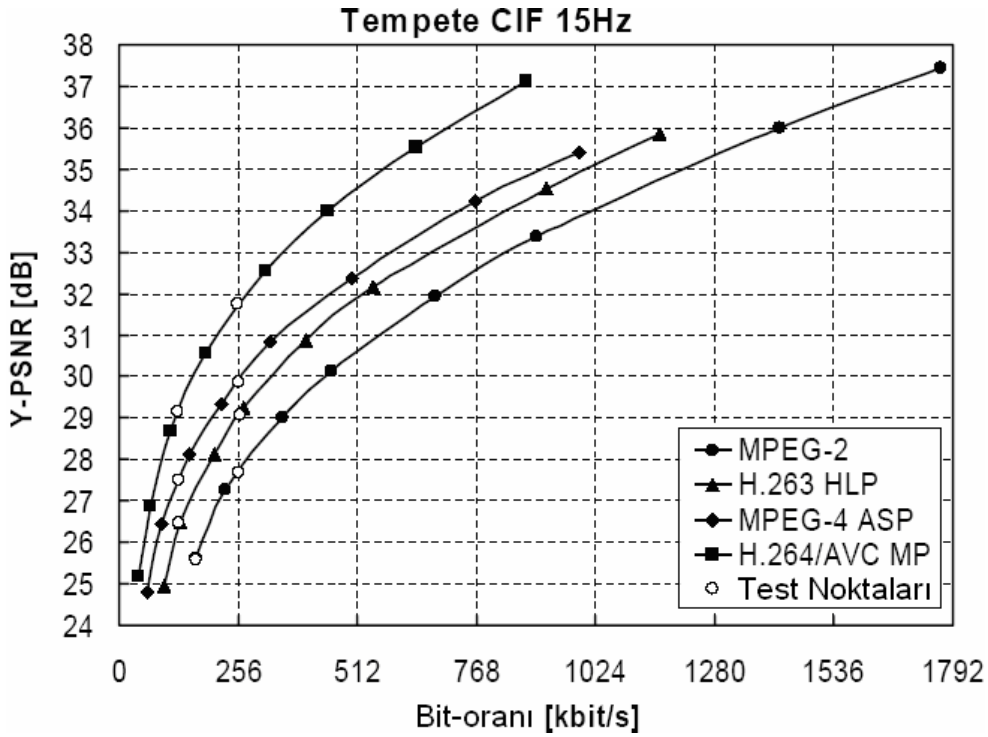
Şekil 6.17. MPEG-4 ASP ve AVC karşılaştırması (Foreman, QCIF, 10Hz)

JVT ve VCEG gruplarında aktif görev yapan uzmanlar tarafından yapılan kayıplı hareketli görüntü sıkıştırma yöntemlerinin karşılaştırma sonuçları Şekil 6.18 ve Şekil 6.19'da gösterilmiştir [Wiegand vd., 2003]. Yüksek çözünürlük ve yüksek bit oranlarının gerekli olduğu DVD (Sayısal Video Disk) ve HD-TV (Yüksek Tanımlı Televizyon) gibi standartların tek sıkıştırma yöntemi olan MPEG-2 ile, HD-DVD (Yüksek Tanımlı DVD) standardında MPEG-2 dışında bir seçenek olarak sunulan AVC'nin karşılaştırması Şekil 6.18'de verilmiştir. Şekilden de görüldüğü gibi AVC 1 Mbit/s veri oranında 30 PSNR kalite değeri elde ederken, MPEG-2 bu kaliteyi 2 Mbit/s'nin üzerinde sağlayabilmektedir.

Şekil 6.19'da, dört farklı kayıplı hareketli görüntü sıkıştırma yönteminin 0-1792 kbit/s veri oranlarında gösterdikleri sıkıştırma performansı yer almaktadır [Wiegand vd., 2003]. En iyi sonuçları yine AVC elde ederken, MPEG-4 ASP ikinci sırada, H.263 üçüncü sırada ve MPEG-2 ise son sırada yer almıştır.



Şekil 6.18. Yüksek bit oranlarında MPEG-2 ve AVC karşılaştırması



Şekil 6.19. Dört farklı yöntemin karşılaştırması

7. SONUÇLAR

Kayıpsız sıkıştırma yöntemlerinin başarısından bahsedilirken iki önemli ölçüt vardır: sıkıştırma oranı ve sıkıştırma süresi. Sıkıştırma oranı ölçütünde en başarılı olan algoritmalar PPM yaklaşımını temel alan istatistiksel tabanlı algoritmalardır. Ne var ki, bu algoritmaların sıkıştırma süresi sözlük tabanlı algoritmalara göre daha fazladır. Bu nedenle, bilgisayar ağlarındaki veri trafiğini hafifletme amacıyla yapılan gönderim öncesi sıkıştırma işlemlerinde verinin hızlı sıkıştırılması gerekli olduğu için, bu durumlarda sözlük tabanlı yaklaşımlar daha çok kullanılmaktadır. Verileri sıkıştırarak saklamak söz konusu olduğunda ise, kullanıcılar genellikle daha çok sıkıştırabilmek uğruna daha fazla beklemeyi göze almaktadırlar. *Yavaş ama etkili* sıkıştırma yapan WinRAR'ın kullanıcılar tarafından daha çok tercih edilmeye başlanması, önceki sürümlerinde sadece sözlük tabanlı *Deflate* algoritmasını kullanan popüler sıkıştırma programı WinZip'in, 10.0 sürümüne istatistiksel tabanlı yöntemleri de dâhil etmesine yol açmıştır.

Ses sıkıştırma yöntemleri temel olarak konuşma verilerini ve audio[*] türü ses verilerini sıkıştırmak için farklı yaklaşımlar kullanılırlar. Konuşma verileri, genellikle insan sesinin analizine dayalı bir yaklaşım kullanan ve *vocoder* olarak bilinen sıkıştırma yöntemleri ile sıkıştırılırlar. Bit oranının çok düşük olmasını sağlayan bu yöntemler çoğunlukla telefon haberleşmesinde tercih edilir. Fakat audio türü sesler daha karmaşık yapıda oldukları için, bu tür seslerin vocoder'lar ile sıkıştırılması kaliteyi çok fazla düşürmektedir. Bu nedenle audio türü sesler için ses dalgası kodlamasına dayalı yapılar tercih edilir. Bu tezde ses dalgası kodlamasına dayalı yöntemler incelenmiştir. MP3 kodlayıcısı, yeni geliştirilen AAC ve WMA gibi kodlayıcılara göre çok daha düşük bir sıkıştırma performansına sahip olsa da, yazılım ve donanım üreticileri tarafından geçmişte en çok desteklenen yöntem olması nedeniyle halen en yaygın olarak kullanılan yöntemdir. Fakat, yazılım üreticileri geliştirdikleri yazılımlarının yeni sürümlerinde ve müzik çalar cihaz üreticileri de yeni ürünlerinde AAC ve WMA yöntemlerine destek vermektedirler. Bu nedenle bu yöntemlerin kısa sürede yaygınlaşması beklenebilir.

---

[*] Audio terimi genellikle müzik ve sinema gibi yüksek frekanslı sesleri tarif eder.

Kayıpsız görüntü sıkıştırma yöntemlerini karşılaştırmak üzere yaptığımız çalışmada görüldüğü gibi, bir görüntünün renk derinliği ve karmaşıklığına göre o görüntüyü en çok sıkıştırabilen yöntem belirlenebilmektedir. Bu nedenle, eğer her tür görüntü için iyi oranlarda sıkıştırma yapabilecek genel amaçlı bir kayıpsız görüntü sıkıştırma uygulaması geliştirilmek istenirse, bu uygulamaya birçok farklı yöntem dâhil edilmeli ve görüntü tipi algılandıktan sonra ilgili yöntem ile sıkıştırma işlemi yapılmalıdır. PNG kayıpsız görüntü sıkıştırma yöntemi, gelişmiş özellikleri, yüksek oranda sıkıştırma yeteneği ve patent koruması altında olmaması nedeniyle günümüzün en çok kullanılan kayıpsız sıkıştırma yöntemleri arasındadır.

JPEG2000 kayıplı görüntü sıkıştırma yöntemi sahip olduğu özellikler ve sıkıştırma performansı olarak JPEG yönteminden çok daha üstün olsa da, karmaşık yapısı nedeniyle sıkıştırma süresi çok daha uzun olmaktadır. Ayrıca, tezde yer alan bu iki yöntemin karşılaştırılmasına yönelik çalışmada da (Bölüm 5.5) görülmüştür ki, görüntü kalitesinin kapasiteden daha önemli olduğu durumlarda kullanılan düşük sıkıştırma oranlarında, JPEG ile JPEG2000 arasında çok büyük fark bulunmamaktadır. Sayısal fotoğraf makinesi üreticilerinin, geliştirilmesinin üzerinden 5-6 yıl gibi bir süre geçmesine rağmen halen JPEG2000 standardına destek vermemeleri, büyük oranda JPEG2000'in daha çok işlemci gücüne (dolayısıyla daha çok enerjiye) gereksinim duymasından kaynaklanmaktadır. Sürekli çekim kipinde bir saniyede kaç fotoğraf çekilebildiği, tam şarj edilmiş pil ile toplam kaç adet fotoğraf çekilebildiği gibi değerlerin sayısal fotoğraf makinelerinde yüksek olması istendiği için, JPEG kodlayıcısının daha hızlı işlem yapması onu cazip kılmaktadır.

Kayıplı hareketli görüntü sıkıştırma kodlayıcıları arasında günümüzde en yaygın olarak kullanılan kodlayıcılar (DivX, XviD, …) MPEG-4 ASP tabanlıdır. DVD'nin temelini oluşturan MPEG-2 yöntemine göre daha yüksek sıkıştırma yeteneğine sahip olan MPEG-4, DVD'nin görüntü kalitesine yakın kaliteyi çok daha düşük veri oranlarında sağlayabildiği için, Internet üzerinden film paylaşımına hız kazandırmış, bu nedenle de film üreticilerinin tepkisini çekmiştir. AVC standardı, sahip olduğu yüksek oranda sıkıştırma yeteneği sayesinde yasal olmayan yollardan film paylaşımının yeni gözdesi olmaya adaydır. Diğer yandan, AVC standardı sahip olduğu diğer üstün özellikler sayesinde gelecekte HD-DVD gibi birçok uygulama alanında yer alacaktır.

Bu sayede AVC standardı, MPEG-4 ASP gibi çoğunlukla yasadışı film paylaşımına hizmet eden bir standart olmanın dışında, yasal çerçevede yaygınlaşma imkânı bulacaktır. Günümüzde mobil cihazlar ve cep telefonlarının en çok destekledikleri 3GPP hareketli görüntü sıkıştırma standardının temel sıkıştırma yöntemleri, H.263 ve MPEG-4 ASP yöntemleridir. Sahip oldukları düşük işlemci gücü nedeniyle ancak düşük kaliteli hareketli görüntüleri yürütebilen bu cihazlar, AVC'nin temel profili sayesinde, küçük boyutlu belleklerine daha çok video saklayabileceklerdir. Bu nedenle AVC'nin mobil cihazlarda da yaygınlaşması beklenebilir.

Tez çalışmasının bir parçası olarak geliştirilen ISSDC yönteminin, kayıpsız görüntü sıkıştırma yöntemlerinin karşılaştırıldığı çalışmada (Bölüm 5.3), özellikle fotoğraf haricindeki görüntülerde başarılı olduğu görülmüştür. Fotoğraf görüntülerinde de başarılı olabilmesi için, algoritmanın bu tür yapılara göre özelleştirilmesi gerekmektedir. Algoritmaya sözlükteki fazlalıkları eleyebilen bir yapı eklenmiş, bu yapı sayesinde sıkıştırma oranı bir miktar daha artmış, fakat sıkıştırma hızı yavaşlamıştır. Algoritmanın daha az geçişle çalışabilmesini sağlayacak bir iyileştirme çalışması yapılırsa, algoritmanın en önemli dezavantajı olan yavaşlığına da çözüm üretilmiş olunacaktır. Ayrıca, algoritmanın büyük sözlükler ile çalışırken daha hızlı bir arama ve sıralama yapısını kullanması da sıkıştırma hızını arttıracaktır.

KAYNAKLAR

1. Abramson, N., 1963, "Information Theory and Coding", McGraw-Hill, New York.
2. Aizawa, K., Huang, T.S., 1995, "Model–based image coding: Advanced video coding techniques for very low bit–rate applications", Proceedings of the IEEE, 83(2), 259–271.
3. Aldus Developers Desk, 1992, "TIFF Revision 6.0", Aldus Corporation, Seattle.
4. Barnsley, M. F., Sloan, A. D., 1987, "Chaotic Compression", Computer Graphics World, November 1987.
5. Barnsley, M. F., 1988, "Fractals Everywhere", Academic Press Professional, San Diego, CA, USA.
6. Boliek, M., 1996, "New work item proposal: JPEG2000 image coding system", ISO/IEC JTC1/SC 29/WG1 N390.
7. Burrows M., Wheeler, D. J., 1994, "A Block-sorting Lossless Data Compression Algorithm", Technical report, Digital Equipment Corporation, Palo Alto, California.
8. Capon, J., 1959, "A Probabilistic Model for Run-Length Coding of Pictures", IRE Transactions on Information Theory, 157-163.
9. Christopoulos, C., Skodras, A., Ebrahimi, T., 2000, "The JPEG2000 still image coding system: An Overview", IEEE Transactions on Consumer Electronics, 46(4), 1103-1127.
10. Cleary, J. G., Witten, I. H., 1984, "Data Compression Using Adaptive Coding and Partial String Matching", IEEE Transactions on Communications, 32(4), 396-402.
11. CompuServe, 1990, "Graphics Interchange Format(sm), Version 89a", CompuServe Incorporated, Columbus, Ohio.
12. Deutsch, P., 1996, "DEFLATE Compressed Data Format Specification, version 1.3", Network Working Group, Request for Comments 1951.
13. Ebrahimi, F., Chamik, M., Winkler, S., 2004, "JPEG vs. JPEG2000: An objective comparison of image encoding quality", Proc. SPIE Applications of Digital Image Processing, 5558, 300-308, Denver.
14. Faller, N., 1973, "An Adaptive System for Data Compression", Seventh Asilomar Conference on Circuits, Systems and Computers, 593-597.
15. Galleger, R. G., 1978, "Variations on a Theme by Huffman", IEEE Transactions on Information Theory, 24(6), 668-674.
16. Hamilton, E., 1992, "JPEG File Interchange Format Version 1.02".

17. Herre, J., Johnston, J. D., 1996, "Enhancing the Performance of Perceptual Audio Coders by Using Temporal Noise Shaping (TNS)", 101st AES Convention (Los Angeles), preprint 4384.

18. Herre, J., Schulz, D., 1998, "Extending the MPEG-4 AAC Codec by Perceptual Noise Substitution", 104th AES Convention (Amsterdam), preprint 4720.

19. Howard, P. G., Vitter, J. S., 1994 "Arithmetic coding for data compression", Proc. of the IEEE, 82(6), 857-865.

20. Huffman, D. A., 1952, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of IRE, 40(9), 1098-1101.

21. ISO/IEC 10918-1, 1994, CCITT Rec. T.81, 1992, "Digital Compression And Coding of Continuous-Tone Still Images – Requirements And Guidelines (JPEG)".

22. ISO/IEC 11172-3, 1993, "Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s".

23. ISO/IEC 11544-3, 1993, CCITT Rec. T.82, 1993, "Coded representation of picture and audio information – Progressive bi-level image compression (JBIG-1)".

24. ISO/IEC 13818-3, 1994, "Information Technology: Generic coding of Moving pictures and associated audio - Audio Part".

25. ISO/IEC 13818-7, 1997, "MPEG-2 Advanced Audio Coding, AAC".

26. ISO/IEC 14492, 2001, ITU-T Rec. T-88, 2000, "Lossy/lossless coding of bi-level images (JBIG-2)"

27. ISO/IEC 14496-3, 1999, "Coding of Audiovisual Objects - Part 3: Audio".

28. ISO/IEC 14496-3, 2005, Amd 2, 2006, "Audio Lossless Coding (ALS), new audio profiles and BSAC extensions"

29. ISO/IEC 15444-1:2000, "JPEG 2000 image coding system: Core coding system"

30. Iwakami, N., Moriya, T., 1996, "Transform domain weighted interleave vector quantization (Twin VQ)", 101st AES Convention (Los Angeles), preprint 4377.

31. Jacaba, J. S., 2001, "Audio Compression Using Modified Discrete Cosine Transform: The Mp3 Coding Standard", BSc Research Paper, The University of the Philippines, Diliman, Quezon City.

32. Jacquin, A. E., 1989, "A Fractal Theory of Iterated Markov Operators with Applications to Digital Image Coding", Ph.D. Thesis, Georgia Institute of Technology.

33. Jacquin, A. E., 1992, "Image Coding Based on Fractal Theory of Iterated Contractive Image Transformations", IEEE Transactions on Image Processing, 1, 18-30.

34. Jacquin, A. E., 1993, "Fractal image coding: a review", Proceedings of the IEEE, 81(10), 1451-65.

35. Jelinek, F., 1968, "Probabilistic Information Theory", McGraw-Hill, New York.

36. Knuth, D. E., 1985, “Dynamic Huffman Coding”, Journal of Algorithms, 6(2), 163-180.

37. Koenen, R., 2002, “Overview of the MPEG-4 Standard”, ISO/IEC JTC 1/SC 29/WG 11/N4668.

38. Liebchen, T., Moriya, T., Harada, N., Kamamoto, Y., Reznik, Y. A., 2005, “The MPEG-4 Audio Lossless Coding (ALS) Standard - Technology and Applications”, 119th AES Convention (New York).

39. Lin, X., 1991, “Dynamic Huffman Coding for Image Compression”, MS Thesis, University of Nebraska.

40. Mallat, S., 1989, “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation”, IEEE Pattern Analysis and Machine Intelligence, 11(7), 674-693.

41. Malvar, H. S., Hallapuro, A., Karczewicz, M., Kerofsky, L., 2003, “Low-Complexity Transform and Quantization in H.264/AVC”, IEEE Transactions On Circuits And Systems for Video Technology, 13(7), 598-603.

42. Mandelbrot, B., 1977, “Fractals: Form, Chance and Dimension”, W. H. Freeman and Co.

43. Marcellin, M. W., Gormish, M. J., Bilgin, A., Boliek, M. P., 2000, “An Overview of JPEG-2000”, Proceedings of IEEE Data Compression Conference (Utah), 523-541.

44. Meltzer, S., Moser, G., 2006, “MPEG-4 HE-AAC v2 – audio coding for today’s digital media world”, EBU Technical Review.

45. Mesut, A., Carus, A., 2004, “A New Approach to Dictionary-Based Lossless Compression”, UNITECH’04 International Scientific Conference, Gabrovo.

46. Moffat, A., 1990, “Implementing the PPM data compression scheme”, IEEE Transactions on Communications, 38(11), 1917-1921.

47. Moffat, A., Neal, R. M., Witten, I. H., 1995, “Arithmetic Coding Revisited”, ACM Transactions on Information Systems, 16, 256-294.

48. Ojanperä, J., Väänänen, M., 1999, “Long Term Predictor for Transform Domain Perceptual Audio Coding,” 107th AES Convention (New York), Preprint 5036

49. Pennebaker W. B., Mitchell, J. L., Langdon, G. G., Arps, R. B., 1988, “An overview of the basic principles of the Q-coder adaptive binary arithmetic coder”, IBM Journal of Research and Development, 32(6), 717-726.

50. Pasco, R., 1976, “Source Coding Algorithms for Fast Data Compression”, Ph.D. Thesis, Stanford University.

51. Randers-Pehrson, G., 1999, “PNG (Portable Network Graphics) Specification, Version 1.2”, PNG Development Group.

52. Rissanen, J. J., 1976, “Generalized Kraft Inequality and Arithmetic Coding”, IBM Journal of Research and Development, 20, 198-203.

53. Rissanen, J. J., Langdon, G. G., 1979, “Arithmetic Coding”, IBM Journal of Research and Development, 23(2), 149-162.

54. Sayood, K., 1996, “Introduction to Data Compression”, Morgan Kaufman, San Francisco, California.

55. Shannon, C. E., 1948, “A Mathematical Theory of Communication”, The Bell System Technical Journal, 27, 379–423.

56. Shkarin, D., 2002, “PPM: One Step to Practicality”, Proceedings of IEEE Data Compression Conference (Utah), 202-211.

57. Smith, M., 2000, “Lossless Realtime Video Compression On The Pentium III”, Imperial College Computing Dept., MEng Final Year Project.

58. Storer J. A., Szymanski, T. G., 1982, “Data compression via textual substitution”, Journal of the ACM, 29, 928-951.

59. Sullivan, G. J., Topiwala, P., Luthra, A., 2004, “The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions”, SPIE Conference on Applications of Digital Image Processing XXVII.

60. Tekalp A. M., Ostermann, J., 2000, “Face and 2-D mesh animation in MPEG-4”, Signal Processing: Image Communication, Special Issue on MPEG-4, 15, 387-421.

61. Vitter, J. S., 1987, “Design and Analysis of Dynamic Huffman Codes”, Journal of ACM, 34(4), 825-845.

62. Wallace, G. K., 1991, “The JPEG Still Picture Compression Standard”, Communications of the ACM, 34(4), 30-44.

63. Weinberger, M. J., Seroussi, G., Sapiro, G., 1998, “The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS”, Computer Systems Laboratory, HPL-98-193.

64. Welch, T. A., 1984, “A Technique for High-Performance Data Compression”, IEEE Computer, 17(6), 8-19.

65. Wiegand, T., Sullivan, G. J., Bjontegaard, G., Luthra, A., 2003, “Overview of the H.264 / AVC Video Coding Standard”, IEEE Transactions On Circuits And Systems For Video Technology.

66. Wiegand, T., Schwarz, H., Joch, A., Kossentini, F., Sullivan, G. J., 2003, “Rate-constrained coder control and comparison of video coding standards”, IEEE Transactions on Circuits, Systems and Video Technology, 13(7): 688-703.

67. Witten, I. H., Bell, T., “The Calgary/Canterbury text compression corpus”, Anonymous ftp, ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/

68. Witten, I. H., Neal, R. M., Cleary, R. J., 1987, “Arithmetic Coding for Data Compression”, Communications of the ACM, 30, 520-540.

69. Ziv, J., Lempel, A., 1977, “A Universal Algorithm for Sequential Data Compression”, IEEE Transactions on Information Theory, IT-23(3), 337-343.

70. Ziv, J., Lempel, A., 1978, “Compression of Individual Sequences via Variable-Rate Coding”, IEEE Transactions on Information Theory, IT-24(5), 530-536.

ÖZGEÇMİŞ

1976'da Bulgaristan'ın Varna şehrinde dünyaya gelen Altan Mesut, 1978'de ailesi ile birlikte Türkiye'ye gelip İstanbul'a yerleşti. 1994'te İstanbul Üniversitesi Bilgisayar Bilimleri Mühendisliği Bölümü'nde lisans eğitimine başladı ve 1998'de bu bölümden mezun oldu. Bir yazılım firmasında kısa süre çalıştıktan sonra, 1998 yılında Trakya Üniversitesi Bilgisayar Mühendisliği Bölümü'nde araştırma görevlisi olarak göreve başladı. 1999 yılında bu bölümde yüksek lisans eğitimine başladı ve 2002 yılında yüksek lisansını tamamladı. 2005 yılında Trakya Üniversitesi Bilgisayar Mühendisliği Bölümü'ne öğretim görevlisi olarak atandı ve halen bu görevini sürdürmektedir.