



EGE ÜNİVERSİTESİ

DOKTORA TEZİ

VERİ SIKIŞTIRMA YÖNTEMLERİ ÜZERİNE

Tural SADIK

Tez Danışmanı: Doç. Dr. Burak ORDİN

İkinci Tez Danışmanı: Doç. Dr. Fidan NURİYEVA

Matematik Anabilim Dalı

Bilim Dalı Kodu : 619.03.03

Sunuş Tarihi : 09.01.2019

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(DOKTORA TEZİ)

VERİ SIKIŞTIRMA YÖNTEMLERİ ÜZERİNE

Tural SADIK

Tez Danışmanı: Doç. Dr. Burak ORDİN

İkinci Tez Danışmanı: Doç. Dr. Fidan NURİYEVA

Matematik Anabilim Dalı

Sunuş Tarihi: 09/01/2019

Bornova – İZMİR

2019



Tural SADIK tarafından doktora tezi olarak sunulan “Veri Sıkıştırma Yöntemleri Üzerine” başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi’nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 09/01/2019 tarihinde yapılan tez savunma sınavında aday oybirliği/oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri :

İmza :

Jüri Başkanı :

Rapörter Üye:

Üye :

Üye :

Üye :



EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ**ETİK KURALLARA UYGUNLUK BEYANI**

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Doktora Tezi olarak sunduğum “Veri Sıkıştırma Yöntemleri Üzerine” başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

12/12/2018

İmzası

Tural SADIK



ÖZET**VERİ SIKIŞTIRMA YÖNTEMLERİ ÜZERİNE****SADIK, Tural**

Doktora Tezi, Matematik Anabilim Dalı
Tez Danışmanı: Doç. Dr. Burak ORDİN
İkinci Tez Danışmanı: Doç. Dr. Fidan NURİYEVA
Ocak 2019, 115 sayfa

Günümüzde metin, parmak izi, harita gibi alanlarda geniş bir uygulamaya sahip olan ikili görüntülerin çok büyük miktarda depolanması nedeniyle sıkıştırılıp boyutlarının küçültülmesi büyük önem taşımaktadır. Görüntü sıkıştırma teknikleri görüntülerin saklanması için gerekli depolama alanının azaltılması ve ağlar üzerindeki transferlerinin hızlandırılması için yaygın olarak kullanılmaktadır. İkili görüntülerde sıkıştırma işleminde görüntüyü oluşturan bitlerin sayısının azaltılması amaçlanır.

Bu tezde veri sıkıştırma teknikleri üzerine araştırma yapılmıştır. Yeni bir sayı sistemi ortaya konularak, ikili görüntülerin kayıpsız sıkıştırılması için daha verimli yeni bir kodlama yöntemi önerilmiştir. Geliştirilen yeni kodlama sisteminin özellikle ardışık tekrarların olduğu sistemlerde yüksek sıkıştırma sağlayabileceği gözlemlenmiştir. Bu sıkıştırma yönteminin diğer bir avantajı da onun tekrar-tekrar aynı dizinin sıkıştırılmış versiyonları üzerinde birkaç defa uygulanabilmesidir.

Bu alanda en yaygın olarak bilinen koşan-uzunluk kodlama yöntemi ile yeni geliştirilen yöntem karşılaştırılmıştır. Elde edilen karşılaştırmalı sonuçlar yeni yöntemin ikili görüntülerin sıkıştırılmasında daha başarılı olduğunu göstermiştir. Özellikle uzun ardışık özdüş bitlere sahip görüntülerde yeni metodun etkiliğinin koşan-uzunluk metoduna göre daha iyi olduğu görülmüştür.

Bu çalışmada önerilen yeni sayı sistemi şifrelemede de kullanılabilir.

Anahtar Kelimeler: Karakter Tipli Veri Sıkıştırma Teknikleri, Görüntü Sıkıştırma Teknikleri, İkili Görüntüler, Yeni Sayı Sistemi, Yeni Kodlama Yöntemi, İkili Görüntülerde Yeni Sıkıştırma Teknikleri.



ABSTRACT
ON DATA COMPRESSION METHODS
SADIK, Tural

Doctoral Thesis, Department of Mathematics
Thesis Advisor: Assoc. Prof. Dr. Burak ORDİN
Second Thesis Advisor: Assoc. Prof. Dr. Fidan NURİYEVA
January 2019, 115 Page

It is very important to compress and size the images because they are stored in very large quantities of binary images that have a wide range of applications in fields such as text, fingerprint, and map. With increased ever increasing of binary images in a wide range of applications like text, fingerprint, and mapping, its storage and accumulation in huge amounts makes it important to compress and reduce size of the images.

In this thesis, data compression hechniques are investigated. By introducing a new coding system, a technique has been proposed that can commpress binary images more efficiently. It has been shown that the new coding system devoloped can provide high compression especially for systems with consecutive repetitions. Anohter advantage of this compression method is its iterative applicability on compressed versions of the same data several times.

In this work, a new coding method for lossless compression of binary images is proposed. The most commonly known run-length encoding method in this area is compared with the newly developed method. The comparative results obtained show that the new method is successful in compressing binary images. Especially in images with long consecutive identical bits, the effectiveness of the new method has been shown to be better than the run-length method.

The new coding system proposed in this study can also be used for encryption.

Key Words: Character Typing Data Compression Techniques, Bit-Level Huffman Coding, Image Compression Techniques, Binary Images, New Number System, New Coding Method, New Compression Techniques in Binary Images.

x





TEŞEKKÜR

Bu tez çalışması süresince benden desteğini esirgemeyen danışman hocam sayın Doç. Dr. Burak ORDİN'e, gösterdiği anlayışından dolayı, hocam sayın Doç. Dr. Fidan NURİYEVA'ya ve her zaman yanımda olan aileme teşekkür ederim.



İÇİNDEKİLERSayfa

ÖZET	vii
ABSTRACT	ix
TEŞEKKÜR	xi
1. GİRİŞ	1
2. VERİ SIKIŞTIRMA İLE İLGİLİ TEMEL KAVRAMLAR.....	3
2.1. Entropi	3
2.2. Ortalama Mesaj Uzunluğu.....	4
2.3. Tekrarlanma.....	5
2.4. Sıkıştırma Oranı.....	5
2.5. Sıkıştırma Faktörü....	5
2.6. Sıkıştırma Verimi.....	5
2.7. Kraft-Macmillan Eşitsizliği	6
3. KARAKTER TİPLİ VERİ SIKIŞTIRMA TEKNİKLERİ.....	8
3.1. Olasılık Tabanlı Teknikler.....	8
3.1.1. Huffman Kodlaması	8

İÇİNDEKİLER (DEVAM)

	<u>Sayfa</u>
3.1.1.1. Huffman Kodlaması Hakkında Genel Bilgi.....	8
3.1.1.2. Huffman Kodlaması.....	8
3.1.1.3. Huffman Ağacının Oluşturulması.....	9
3.1.1.4. Huffman Kodunun Çözülmesi	13
3.1.2. Aritmetik Kodlama	14
3.1.3. Shannon-Fano	16
3.1.4. BWCA (Burrows-Wheeler Sıkıştırma Algoritması)	17
3.2. Dinamik Sözlük Tabanlı Teknikleri.....	21
3.2.1. Lempel-ziv algoritmaları.....	21
3.2.2. LZ-77	21
3.2.3. LZ-78	26
3.2.4. LZSS	30
3.2.5. LWZ.....	31
3.2.6. LWZR1	35
3.2.1. DEFLATE.....	35
4. GÖRÜNTÜ SIKIŞTIRMA TEKNİKLERİ	36

İÇİNDEKİLER (DEVAM)

Sayfa

4.1. Kayıpsız Görüntü Sıkıştırma	38
4.1.1. RLE (Run-length coding)	38
4.1.2. Entropi Kodlaması	39
4.1.3. LZW Kodlaması	40
4.1.4. Aritmetik Kodlama	40
4.1.5. Alan Kodlaması	41
4.1.6. Çok Çözünürlüklü Kodlama	41
4.1.7. Kayıpsız Tahmin Edici Kodlama	42
4.1.8. JPEG-LS	42
4.1.9. JBIG	44
4.2. Kayıplı Görüntü Sıkıştırma	48
4.2.1. Dönüştürme Kodlaması	48
4.2.2. Ayrık Kosinüs Dönüşümü(DCT).....	49
4.2.3. Ayrık Dalgacık Dönüşümü (DWT)	49
4.2.4. Fraktal Kodlama	52
4.2.5. Blok Kesme Kodlaması	53

İÇİNDEKİLER (DEVAM)

	<u>Sayfa</u>
4.2.6. Altbant Kodlama	53
4.2.7. Vektör niceleme sıkıştırma	54
4.2.8. Çeyrek Ağaç Ayrımı	54
4.2.9. JPEG.....	55
4.2.10. JPEG2000.....	56
4.2.11. Kayıplı ve Kayıpsız Karşılaştırması	58
5. VERİ SIKIŞTIRMADA YENİ BİR TEKNİK.....	60
5.1. Yeni Bir Sayı Sistemi Hakkında Genel Bilgi.....	60
5.1.1. Sayı Sistemleri	60
5.1.1.1. Konumsal Olmayan Sayı Sistemleri	60
5.1.1.2. Konumsal Sayı Sistemleri.....	61
5.1.2. Yeni Sayı Sistemi.....	62
5.1.3. Yeni Kodlama Sistemi	66
5.1.4. Yeni Kodlama Yönteminin Uygulama Alanları	66
5.2. İkili Görüntülerin Kayıpsız Sıkıştırılması İçin Yeni Bir Yöntem.....	67
5.2.1. Önerilen Yöntem.....	69

İÇİNDEKİLER (DEVAM)Sayfa

5.2.1.1. Kodlama Aşaması.....	69
5.2.1.2. Sıkıştırma Aşaması	72
5.2.1.3. Dönüştürücü Aşaması.....	73
5.3. Hesaplama Denemeleri.....	74
6. SONUÇ.....	75
KAYNAKLAR DİZİNİ.....	76
ÖZGEÇMİŞ	79
EKLER	



ŞEKİLLER DİZİNİ

<u>Şekil</u>	<u>Sayfa</u>
3.1	LZ77 uygulamasını kullanarak kodlama 22
3.2	Kodlama süreci.....24
3.3	(7, 4, C(r)) Üçlüsünün kod çözümü.....25
3.4	(7, 4, C(r)) Üçlüsünün kod çözümü.....25
3.5	LZW sözlük başlangıcı.....32
3.6	İlk 12 giriş için LZW sözlüğünün inşası.....33
3.7	Kodlama için LZW sözlüğü.....33
3.8	11 giriş için LZW sözlüğü kod çözüm inşası.....34
4.1	Görünüt Sıkıştırma Aşamaları.....37
4.2	Desen sözlüğü.....40
4.3	Kayıpsız kodlayıcı besileştirilmiş diyagram.....44
4.4	JPEG-LS Algoritmasının blok diyagramı.....44
4.5	JBIG sıkıştırmasının blok diyagramı.....45
4.6	PRES veya QT yöntemlerini kullanarak çözünürlüğü azaltılmış görüntüler.....47
4.7	DP, TPD veya TPD/DP tahminini kullanarak PRES ve QT azaltımı için aritmetik kodlanmış pikseller, RR önceki şekildeki ile aynıdır.....47
4.8	Band dalgacık analizi ve sentezi filtresi.....50

ŞEKİLLER DİZİNİ (DEVAM)

<u>Şekil</u>	<u>Sayfa</u>
4.9	Görüntü sıkıştırma için blok diyagramı 51
4.10	Soldaki orijinal resim ve sağdaki JPEG ile sıkıştırılmış resim..... 55
4.11	JPEG kodlama ve kod çözme algoritması..... 56
4.12	JPEG-2000 kodlama ve kod çözme yapısı..... 57
4.13	JPEG2000 ve TIFF arasındaki depolama boyutunun karşılaştırılması.. 58
4.14	JPEG ve JPEG2000 dosya boyutu karşılaştırılması..... 59
4.15	JPEF ve JPEG2000 resim kalitesi karşılaştırması..... 59
5.1	Sıkıştırma metotlarının ağaç görünümü..... 68
5.2	Test görüntüler (a) “Baloncuklar”; (b) “Daireler” ; (c) “Uydu görüntüsü” ; (d) “Dart” ; (e) “Harfler” ; (f) “Metin”..... 74
5.3	Karşılaştırma sonuçları..... 74

1. GİRİŞ

Bu tezin amacı var olan veri sıkıştırma teknikleri hakkında bilgi vermek ve yeni bir sıkıştırma tekniği ortaya koymaktır.

Tez Giriş, 4 Bölüm, Sonuç, Kaynaklar Listesi ve Ekler bölümlerinden oluşmaktadır.

Tezin giriş bölümünde, tezin içeriğine kısa bir şekilde değinilmiştir.

Bölüm 1’de veri sıkıştırma ile ilgili entropi, sıkıştırma oranı, sıkıştırma verimi gibi konu içerisinde kullanılan önemli temel kavramlardan bahsedilmiştir.

Bölüm 2’de ise karakter tipli teknikler olasılık tabanlı ve sözlük tabanlı teknikler olarak incelenmiş, olasılık tabanlı tekniklerde veri sıkıştırma konusunu ilk ortaya atan ve şu ana kadarki en verimli algoritma özelliği taşıyan Huffman algoritması ve kodlaması detaylı olarak açıklanmıştır.

Görüntüler piksel piksel kaydedilip ve aktarılırken de bu kaydetme ile aktarıldığı için sıkıştırılması büyük önem taşımaktadır. Ayrıca, görüntülerin daha kaliteli veya görselliğin daha iyi olması için piksel sayısı arttırıldığında depolamanın problem olması kaçınılmazdır. Bununla beraber, ikili görüntülerin depolanmasında kullanılan koşan uzunluk kodlaması daha verimli sonuçlar vermektedir. Bu tezde de bu koşan uzunluk kodlamasının avantajı kullanılarak yeni bir sayı sistemi ortaya konularak verimli olan koşan uzunluk kodlaması daha verimli hale getirilmiştir. Bölüm 3’de görüntü sıkıştırma teknikleri tek tek incelenmiş, kayıplı ve kayıpsız olmak üzere iki kısımda ele alınmıştır.

Son zamanlarda çeşitli sektörlerde teknoloji kullanımı giderek artmaktadır ve bunun sonucu olarak bazı alanlarda ikili görüntüler ortaya çıkmaktadır. Bu durumda büyük miktardaki ikili görüntülerin depolanması çok önemlidir. Bu sebeple, depolama alanının azaltılması amacıyla veri sıkıştırma teknikleri kullanılır. Bu sadece depolama alanının azaltılmasında değil ağlar üzerinde iletişim sağlanırken de transfer hızının arttırılmasında kullanılabilir. Bu nedenlerden dolayı Bölüm 5’de, ikili görüntülerin kayıpsız sıkıştırılması için yeni bir kodlama yöntemi önerilmiştir. Görüntü sıkıştırma algoritmalarının amacı, ikili rakamları aynı bilgiyi belirten ancak daha az basamak içeren yeni bir ikili sayıya dönüştürmektir. İkili görüntüleri kayıpsız sıkıştırma yöntemlerinden biri olan koşan-uzunluk yöntemi ile

önerilen yöntem karşılaştırılmıştır ve elde edilen karşılaştırmalı sonuçlar yeni yöntemin ikili görüntülerin sıkıştırılmasında başarılı olduğunu göstermiştir.

Sonuç bölümünde ise yapılanların kısa özeti verilmiştir.



2. VERİ SIKIŞTIRMA İLE İLGİLİ TEMEL KAVRAMLAR

2.1 Entropi

Shannon entropinin tanımını istatistiksel fizikten almıştır; burada entropi bir sistemin rastlantısallığını veya bozukluğunu temsil eder. Bir sistemin, olabilecek bir dizi olası duruma sahip olduğu varsayılır. Bu durumların belirli bir zamanda olasılık dağılımları mevcuttur. Entropi şu şekilde tanımlanır:

$$H(S) = \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)}$$

Burada S olası durumların kümesidir ve $p(s)$, $s \in S$ durumunun olasılığıdır. Tanımdan da anlaşılacağı üzere, olasılık değerleri ne kadar yüksek olursa entropi değeri o kadar düşük olur. Örneğin; sistemin tam olarak hangi durumda olduğu biliniyorsa, $H(S) = 0$ olur. Shannon, aynı zamanda, bir mesajın “öz bilgi” bilgisini aşağıdaki gibi tanımlamıştır (Blelloch, 2001).

$$i(s) = \log_2 \frac{1}{p(s)}$$

Öz bilgi, içerdiği bilgi bitlerinin sayısını yani daha açık ifadeyle, konuşmada mesajı kodlamak için kullanılması gereken bit sayısını temsil eder. Kişisel bilginin tanımı, daha yüksek ihtimalli iletilerin daha az bilgi içerdiğini gösterir (örneğin, yarın hava güneşli olacağını söyleyen bir mesaj, kar yağacağını söyleyen mesajdan daha bilgilendirici olduğunu belirtir).

Bununla beraber, entropi her mesajın kendi bilgisinin olasılık ağırlıklı ortalamasıdır. Dolayısıyla, olasılık dağılımından rastgele seçilen bir mesajda bulunan ortalama bilgi bit sayısıdır. Daha büyük entropiler, daha büyük ortalama bilgiyi temsil eder ve aslında sezgisel olarak bir mesaj kümesi ne kadar rastgele olursa (ihtimaller o kadar olasıdır), içerdikleri bilgi o kadar çok olur (Blelloch, 2001).

Beş ileti üzerinde farklı olasılık dağılımları için bazı entropi örnekleri aşağıdaki gibidir:

$$p(s) = \{0.25, 0.25, 0.25, 0.125, 0.125\}$$

$$H = 3 \times 0.25 \times \log_2 4 + 2 \times 0.125 \times \log_2 8$$

$$= 1.5 + 0.75$$

$$= 2.25$$

$$p(s) = \{0.5, 0.125, 0.125, 0.125, 0.125\}$$

$$H = 0.5 \times \log_2 2 + 4 \times 0.125 \times \log_2 8$$

$$= 0.5 + 1.5$$

$$= 2$$

$$p(s) = \{0.75, 0.0625, 0.0625, 0.0625, 0.0625\}$$

$$H = 0.75 \times \log_2 \frac{4}{3} + 4 \times 0.0625 \times \log_2 16$$

$$= 0.3 + 1 = 1.3$$

Dağılım ne kadar pürüzlü olursa entropi de düşer. Ayrıca hepsi aynı uzunluğa ait bit mesajı kodlamak için $\log_2 n$ bit gerekir (Blelloch, 2001).

2.3 Ortalama Mesaj Uzunluğu

Huffman ortalama mesaj uzunluğunu, kaynak alfabedeki her bir sembolün oluşma olasılığı ile bu sembolü kodlamak için gerekli olan bit sayısının (β_i), çarpımlarının toplamı olarak ifade etmiştir. Ortalama mesaj uzunluğu aşağıdaki gibi ifade edilir (Diri, 1999):

$$S = \sum_{i=1}^n P_i \beta_i$$

Bu durumda entropi ile ortalama mesaj uzunluğu arasında aşağıdaki gibi bir bağıntı söz konusudur:

$$H \leq S \leq \lceil \log_2 n \rceil$$

2.4 Tekrarlanma

Tekrarlanma, ortalama mesaj uzunluđu ile entropi arasındaki fark olarak ifade edilmiştir (Shanner and Weaver, 1949). Tekrarlanma aşağıdaki şekilde formüle edilir (Diri, 1999):

$$R = - H + S$$

2.5 Sıkıştırma Oranı

Sıkıştırma oranı, çıkış bilgisi uzunluğunun, giriş bilgisi uzunluğuna oranı olarak aşağıdaki gibi tarif edilir:

$$p = \text{Çıkış Bilgisi Uzunluğu} / \text{Giriş Bilgisi Uzunluğu}$$

Sıkıştırma oranının %60 olarak gerçekleştirildiği bir sistemde, 100 birim uzunluğundaki bir metin, sıkıştırma sonunda %60 birime inmiş demektir. Bir metnin sıkıştırılmış hali aslından büyükse bu işlem “genişletme” adını alır. Bu durumda $p > 1$ olur (Diri, 1999).

2.6 Sıkıştırma Faktörü

Sıkıştırma faktörü, sıkıştırma oranının tersi olup aşağıdaki gibi ifade edilir (Diri, 1999).

$$v = 1/p$$

2.7 Sıkıştırma Verimi

Sıkıştırma sonunda elde edilen başarıyı, sıkıştırma verimi olarak aşağıdaki gibi formüle edebiliriz (Diri, 1999):

$$\eta = 100 \times (1 - p)$$

2.8 Kraft-Macmillan Eşitsizliği

Bu eşitsizlik, değişken uzunluklu tek çözümü olan kodlarla ilgilidir. n adet L_i uzunluğundaki değişken uzunluklu kodlar için Kraf-Macmillan eşitsizliği denklem (2.1)'deki gibi yazılır. Denklem (2.1)'i sağlayan değişken uzunluklu kod için, tek çözümü olan kod diyebiliriz.

$$\sum_{i=1}^n 2^{-L_i} \leq 1 \quad (2.1)$$

Denklem (2.1)'deki eşitsizlik entropi ile ilişkilendirildiğinde L_i kod uzunluğu Denklem (2.2)'deki gibi yazılır:

$$L_i = -\log_2 P_i - E_i \quad (2.2)$$

Burada E_i , L_i 'nin entropiden büyük kısmı olup,

$$2^{-L_i} = 2^{(-\log_2 P_i - E_i)} = 2^{\log_2 P_i} / 2^{E_i} = P_i / 2^{E_i} \quad (2.3)$$

olarak yazılabilir.

Özel bir durum olarak bütün fazla uzunlukların eşit olduğunu kabul edelim. ($E_i = E$) olarak aldığımızda Kraft eşitsizliği

$$1 \geq \sum_{i=1}^n P_i / 2^E = \left(\sum_{i=1}^n P_i \right) / 2^E = 1 / 2^E \Rightarrow 2^E \geq 1 \Rightarrow E \geq 0 \quad (2.4)$$

olur. Tek çözümü olan kod negatif olmayan ekstra uzunluğa sahip, kodun uzunluğu entropi ile tanımlanan uzunluğa eşit veya daha fazla demektir. (2.1)'deki eşitsizliğin kullanımını basit bir örnek ile gösterelim. n tane eşit uzunluklu ikili kod alalım. Her kodun uzunluğu $L_i = \log_2 n$ olsun. Kraft-Macmillan eşitsizliği toplamı,

$$\sum_{i=1}^n 2^{-L_i} = \sum_{i=1}^n 2^{-\log_2 n} = \sum_{i=1}^n (1 / \log_2 n) = 1 \quad (2.5)$$

olur. Bir diğer örnek, n kodun birincisi sıkıştırılacak, ikincisi genişletilecek olsun. Bunu $L_1 = \log_2 n - a$, $L_2 = \log_2 n + e$ ve $L_3=L_4=\dots=L_n=\log_2 n$ şeklinde gösterelim. Burada a ve e pozitif sayılar olup, $e > a$ dır. Bir sembolü a faktörü kadar sıkıştırmak, diğer sembolü bir büyüme faktörü ile genişletmeyi gerektirir. Sıkıştırılan sembolün kullanım sıklığından fazla ise toplam sıkıştırma faktörü $SF > 1$ olur.

$$\sum_{i=1}^n 2^{-L_i} = 2^{-L_1} + 2^{-L_2} + \sum_{i=3}^n 2^{-\log_2 n} \quad (2.6)$$

$$2^{-\log_2 n + a} + 2^{-\log_2 n - e} + \sum_{i=3}^n 2^{-\log_2 n} = 2^{\frac{2^a}{n} + \frac{2^{-e}}{n}} + 1 - \frac{2}{n} \quad (2.7)$$

Kraft-MacMillan eşitsizliğine göre,

$$\frac{2^a}{n} + \frac{2^{-e}}{n} + 1 - \frac{2}{n} \leq 1; \quad \frac{2^a}{n} + \frac{2^{-e}}{n} - \frac{2}{n} \leq 0 \text{ veya } 2^{-e} \leq 2 - 2^a$$

$$-e \leq \log_2(2 - 2^a) \text{ veya } e \geq \log_2(2 - 2^a)$$

olarak ifade edilir. Yukarıdaki eşitsizlikte $a \leq 1$ olmalı (aksi halde $2 - 2^a$ negatif olur) fakat a değeri $(0,1]$ aralığında bir değere sahip olmalı ve bu aralıkta $a > a$ şartını sağlamalıdır (Diri, 1999).

3. KARAKTER TİPLİ VERİ ŞIKIŞTIRMA TEKNİKLERİ

3.1 Olasılık Tabanlı Teknikler

3.1.1 Huffman Kodlaması

3.1.1.1 Huffman Kodlaması Hakkında Genel Bilgi

Olasılık tabalı sıkıştırma tekniklerinden Huffman kodunda, sıkıştırılmak istenen mesajın içerisindeki sembollerin frekansının bulunması işlemi ve bu frekans dağılımının temel alınarak tekrar kodlanması ile sıkıştırma işlemi yapılmaktadır.

Her sembol 8-bit uzunluğa sahiptir. Mesaj içerisinde yüksek frekansa sahip sembollerin daha az sayıda bit ile ifade edilmesiyle, mesajın sembol başına kullanılan ortalama bit sayısı azaltılmış olur. Böylece mesaj daha az yer kaplayarak sıkıştırılmış olur. Örnek olarak Mors alfabesinde harfler kodlanırken sık tekrar eden harfler olabildiğince az karakterle, daha çok tekrar eden harfler olabildiğince çok karakterle ifade edilir. Örnek olarak 'N' karakteri '- .' iken 'Y' karakteri '- . - -' şeklinde temsil edilir. Bu taraftan incelendiğinde Mors kodlaması aslında bir veri sıkıştırma tekniği gibi düşünülebilir. Çünkü bu alfabenin amacı ile veri sıkıştırmanın amacı aynı doğrultudadır. Metin tabanlı verilerde veri kaybı olmaması gerektiğinden huffman sıkıştırma algoritması bu tür verilerde daha elverişlidir. Kayıpsız sıkıştırma algoritması olması bu nedenle önemlidir.

3.1.1.2 Huffman Kodlaması

David Albert Huffman tarafından 1951 yılında geliştirilen kayıpsız sıkıştırma algoritmasıdır. Huffman yöntemi bir derecede Shanno - Fano'ya benzer bir yöntem olmakla beraber daha iyi kodlar üretmektedir (Diri, 1999; Shanmugasundaram and Lourdusamy, 2011).

Huffman kodlama algoritmaları, semboller için kaynak alfabenin olasılık dağılımını kullanır. Olasılık dağılımını hesaplamak için kaynağın tüm karakterlerinin frekans dağılımı hesaplanır. Olasılıklara göre, kod kelimeleri atanır. Daha yüksek olasılıklar için daha kısa kod sözcükleri ve daha küçük olasılıklar için daha uzun kod sözcükleri atanır (Krajci et al., 2015).

Bir metin dosyasındaki karakterlerin kullanılma sıklığına frekans denir. Kısaca, Huffman algoritması bir verideki frekansı yüksek olan karakterleri daha

kısa uzunluklu kodlarla, frekansı düşük olan karakterleri uzun uzunluklu kodlarla temsil etmektedir. Bunun içinde veri içerisindeki her bir karakterin ne sıklıkla geçtiği, frekans tablosu adı verilen bir tablo içerisinde tutulur. Sıkıştırma işlemine başlamadan önce veri üzerinde gezerek bu frekans tablosunu oluşturmak gerekir (Schiller, 2012; Arora and Shukla, 2014).

Huffman algoritması verinin karakter sayısına ve içerdiği karakterlerin frekansına bağlı olarak %10 ile %90 arasında değişen bir veri sıkıştırma oranına sahiptir. Huffman algoritmasındaki karakterlerin kodlanmasında kullanılan bit sayısı değişken uzunluktadır. Yani ASCII'deki gibi sabit uzunluklu kod değildir (Krajci et al., 2015; Schiller, 2012; Arora and Shukla, 2014).

Bir örnekle açıklarsak eğer: Bilgisayar sisteminde her karakter bellekte 1 byte yani 8 bit ile yer kaplamaktadır. Yani 8 karakterden oluşan bir karakter dizisi için bellekte kullanılan yer 8 byte yani $8 \times 8 = 64$ bittir. Örneğin “vvvvvaap” bu şekildeki bir veri için 5 tane v, 2 tane a ve bir tane p harfi kullanılsın. Verimizdeki her bir karakteri 8 bit ile kodlarsak toplamda verimiz için bellekte 64 bitlik bir yer kaplayacaktır. Ama daha sık kullanılan karakteri yani “v” yi “0” ile “a” yi “10” ve son olarakta “p” yi “11” ile kodlarsak 5 tane “v” 5 bit, 2 tane “a” 4 bit ve 1 tane “p” 2 bit toplamda 11 bitlik bir yer kaplayacaktır. Bu şekilde de yaklaşık $(11/64) \times 100 = 17,1875$ oranında bir sıkıştırma elde etmiş oluruz.

3.1.1.3 Huffman Ağacının Oluşturulması

Bir Huffman ağacını oluştururken aşağıdaki adımlar takip edilir. Bu örnekte aşağıdaki karakter frekans tablosu kullanılacaktır.

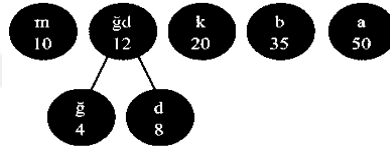
Karakter	Karakter Frekansı
A	50
B	35
K	20
M	10
D	8
Ğ	4

Bu tablodan şu sonuç çıkarılır ki o da şu şekildedir: Verilen döküman dosyasında “b” karakteri 35defa, “m” karakteri 10 defa, “ğ” karakteri 4 defa geçmektedir. Buradaki amaç ise her bir karaktere denk gelen bit dizilerini bulmak.

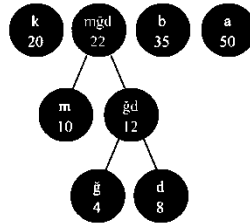
1. İlk başta “Huffman Ağacı”ndaki en son dalları oluşturacak bütün karakterler frekanslarına göre küçükten büyüğe doğru sıralanır.



2. En azfrekansa sahip olan iki karakterin frekansları toplanarak yeni bir düğüm meydana getirilir ve oluşturulan bu yeni düğüm yine küçükten büyüğe olacak şekilde olması gereken yere yerleştirilir. Örnek olarak yukarıdaki şekilde “d” ve “ğ” karakterleri toplanarak “12” frekansında yeni bir “ğd” düğümü oluşturuldu. “12” frekanslı bir karakter şekilde “m” ve “k” karakterleri arasındaki frekans aralığına denk geldiği için o aralığa yerleştirilir. “ğ” ve “d” düğümleri ise yeni oluşturulan düğümün dallarında şekildeki gibi kalır. Yeni ağacımızda aşağıdaki şekilde oluşmaya başlar.

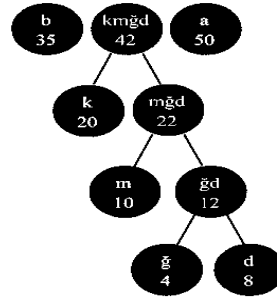


3. Bu adımda da aynı işlem yinelenerek en küçük frekanslı iki karakter düğümü toplanır ve yeni bir düğüm elde edilir. Bu yeni düğümün frekansı 22 olacağı için “k” ve “b” karakterlerinin frekansları arasında yer alacağından bu aralığa yerleştirilecektir. Oluşmaya devam eden ağacımız aşağıdaki şekilde olacaktır.

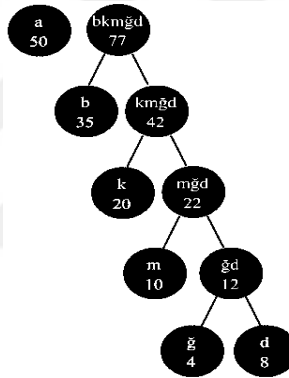


4. Benzer işlemler tekrarlanarak yeni bir düğüm oluşturulmaya devam edilir. Bu yeni düğümün frekansı 42 olacağı için “b” ve “a” düğümleri arasına yerleşecektir. Yeni dizimiz aşağıdaki şekilde olacaktır. Dikkat ederseniz her dalın en ucunda sembollerimiz bulunmaktadır. Dalların ucundaki düğümlere

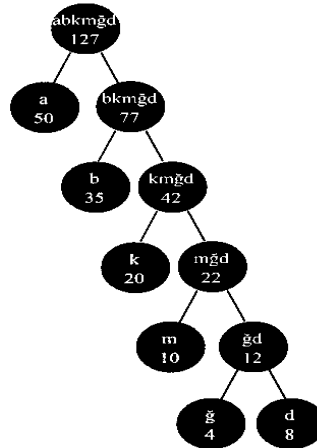
özel olarak yaprak(leaf)denilmektedir. Sona yaklaştıkça Bilgisayarbilimlerinde önemli bir veri yapısı olan ağaç (Tree) veri yapısına yaklaştığımızı görüyoruz.



5. Yine aynı diğer adımlarda olduğu gibi işlemlere devam eldir. Yeni bir düğüm oluşturulur. Bu yeni düğümün frekansı 77 olacağı için “a” düğümünden sonra yerleşecektir. Yeni dizimiz aşağıdaki şekilde olacaktır. Dikkat ederseniz her bir düğümün frekansı o düğümün sağ ve sol düğümlerinin frekanslarının toplamına eşit olmaktadır. Aynı durum düğüm sembolleri içinde geçerlidir.

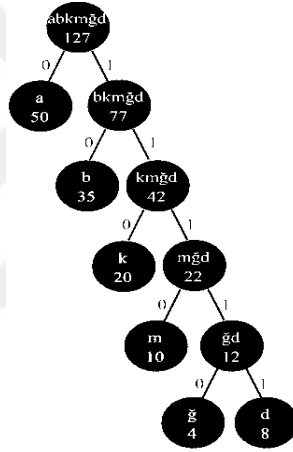


6. İşlemler tek bir düğüm kaldığında durur ve son kalan düğüm Huffman ağacının kök düğümü olarak alınır. Son düğümdeki frekans 127 olmuştur. Böylece Huffman ağacının şekli aşağıdaki gibi olacaktır.



Huffman ağacının şekli her zaman bu şekilde çıkmak zorunda değildir. Bizim seçtiğimiz frekanslara göre değişmektedir yani her doküman için farklı bir ağaç modeli gelebilir.

7. Huffman ağacı son şeklini aldığına göre her bir karakterin yeni kodunu tek tek elde edebiliriz. Karakter kodlarını oluştururken Huffman ağacının en üst kısmından yani tepesindeki kök düğümden başlanarak kök düğümlerinin sağ ve sol düğümlerindeki her bir dala sırasıyla “0” ve “1” kodları yazılır. Sırası farklı da olabilir. Bu tamamen keyfi seçime bağlıdır. Ancak ilk sıra nasılsa diğerleri de aynı olmalıdır. Bu durumda “a” karakterinin düğüme gelen dal “0”, “bkmğd” karakterinin düğüme gelen dal “1” olur. Bu işlem ağaçtaki tüm dalların sonuna kadar devam ettirilir. Dalların kodlarla yazılmış hali aşağıdaki verilmiştir.



8. Şimdi de her bir karakterin bit dizilerinin kodlarını bulalım. Her bir karakter dalların ucunda bulunan yere kadar tek tek kodlar birleştirilerek karakterlerin kodları belirlenmiş olur. Örneğin “k” karakterinin kodu “110” şeklinde olacaktır. “d” karakterinin kodu “11111” şeklinde oluşacaktır. Bu şekilde bütün karakterlerin kodlarını tek tek oluştururuz. Karakterlerin kodları aşağıdaki tabloda verilmiştir.

Sıkıştırılmış veri oluşturulurken ASCII kodları yerine oluşturduğumuz Huffman kodları yazılacaktır. Huffman kodlarının hiçbirisi diğerinin ön eki biçiminde değildir bu da aslında tek çözülebilir bir kod oluşturduğumuzu gösterir. Örnekteki frekans tablosuna göre ne kadarlık bir sıkıştırma sahip olacağını bulalım. Sıkıştırma olmadan önceki bit sayısı her bir karakter 8 bit uzunluğunda olduğundan toplam karakter sayısı $(50+35+20+10+8+4) = 127$ ile 8 in çarpımı bize bellekte ne kadar yer kaplayacağını ya da kaç bitlik yer

muhafaza edeceğini bulmuş oluruz. O da $127 \times 8 = 1016$ bitlik yer gerekmektedir.

Frekans	Karakter	Kodlanmış Bit Sayısı	Huffman Kodu
50	a	1	0
35	b	2	10
20	k	3	110
10	m	4	1110
8	d	5	11111
4	ğ	5	11110

Huffman algoritmasını kullanırsak eğer: 50 tane “a” karakteri için 50 bitlik, 35 tane “b” karakteri için 70 bitlik, 20 tane “k” karakteri için 60 bitlik, 4 tane “ğ” karakteri için 20 bitlik yere gereksinimimiz vardır. Sonuç olarak gerekli olan bit sayısı;

$$1 \times 50 + 2 \times 35 + 3 \times 20 + 4 \times 10 + 5 \times 8 + 5 \times 4 = 50 + 70 + 60 + 40 + 40 + 20 = 280$$

bitlik yer gerekecektir.

Sonuç olarak, 1016 bitlik bellek gereksinimini 280 bite düşürdük. Yani yaklaşık olarak %28 gibi bir sıkıştırma oranı elde etmiş olduk. Gerçek bir ortamda ek olarak karakter frekanslarını da ekleyeceğimizden sıkıştırma oranı %28 den biraz daha fazla olacaktır. Bu genelde sıkıştırılan verinin yanında çok az bir miktar olduğundan ihmal edilebilir.

3.1.1.4 Huffman Kodunun Çözülmesi

Örnekte verilen frekans tablosuna sahip bir metin içerisindeki “aabkdğmma” veri kümesinin sıkıştırılmış hali her karakter ile karakterin kodu yer değiştirilerek aşağıdaki gibi elde edilir.

aa b k d ğ m m a

0 0 10 110 11111 11110 1110 1110 0 → 00101101111111110111011100

Eğer elimizde kodlamanın yapıldığı frekans tablosu ve sıkıştırılmış veri varsa kodlama işleminin tersi yapılarak orijinal veri elde edilir. Şöyle ki; sıkıştırılmış verinin ilk bitine bakılır. Eğer bakılan bit tablodaki bir koda denk geliyorsa, o kodun eşiti olan kod karakteri yerine yazılır, eğer bakılan bit bir kod karakterine eşit değilse sonraki bite bakılarak birlikte ele alınır ve yeni dizinin bir kod karakterine eşit olup olmadığına bakılır. Bu işlem aynı şekilde veri dizisi bitinceye kadar devam edilir. Bu şekilde huffman kodunu çözmüş oluruz. Huffman kodları tek bir şekilde çözülebilir olduğundan kesinlikle farklı bir veri dizisi oluşmaz. Bu da aslında huffman algoritmasının kayıpsız sıkıştırmasının bir sonucudur.

3.1.2 Aritmetik Kodlama

Aritmetik kodlamanın temel kavramı Elias tarafından 1960'lı yılların başında geliştirildi ve daha çok Pasco, Rissanen ve Langdon tarafından geliştirildi. Aritmetik kodlama, Huffman ve Shannon Fano algoritmalarından farklı olarak her giriş simgesini bir kod sözcüğüyle değiştirme fikrini tamamen atlar. Bunun yerine, girdi simgelerinin bir akışını çıktı olarak tek bir kayan noktalı sayı ile değiştirir (Schiller, 2012; Weinberger et al., 1999).

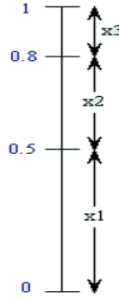
Aritmetik kodlama, ikili kaynaklar gibi küçük alfabe içeren kaynaklarla uğraşırken özellikle çarpık olasılıkları olan alfabeler için yararlıdır. Ayrıca, çeşitli nedenlerle, kayıpsız sıkıştırmanın modelleme ve kodlama özelliklerinin ayrı tutulması gerektiğinde çok yararlı bir yaklaşımdır (Weinberger et al., 1999).

Zaman karmaşıklığı Huffman'dan daha fazla olmasına rağmen, tam sayı uygulaması kötü değildir (Kodituwakku and Amarasinghe, 2010).

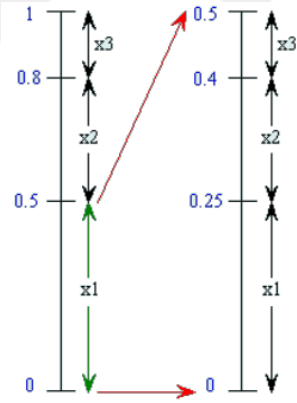
Aritmetik kodlamanın ana amacı, her potansiyel simgeye bir aralık atamaktır. Sonra bu aralığa ondalık sayı atanır. Algoritma, 0.0 ve 1.0 aralığıyla başlar. Alfabadeki her girdi sembolü okunduktan sonra aralık, giriş sembolünün olasılığına oranla daha küçük bir aralık olarak alt bölümlere ayrılmıştır. Bu alt-aralık yeni aralık haline gelir ve girdi alfabesinden gelen sembollerin olasılığına göre bölümlere ayrılır. Bu, her giriş sembolü için tekrarlanır. Sonunda, son aralıktaki herhangi bir kayan noktalı sayı girdi verisini benzersiz şekilde belirler (Choudhary et al., 2015).

Algoritmanın açıklaması şöyledir:

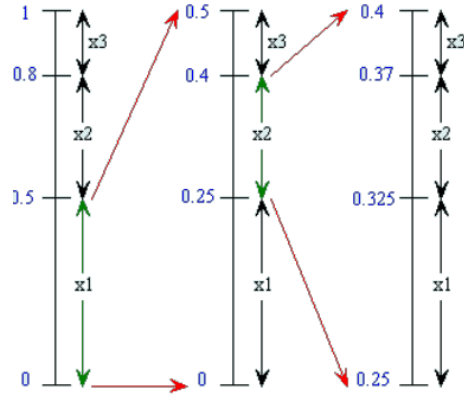
- Alfabedeki her simge için $(0,1)$ eksenini üzerinde örtüşmeyen aralıklar atayın. Her aralığın uzunluğu, karşılık gelen sembolün olasılığı ile orantılı olmalıdır. Bir örnek olarak, x_1, x_2, x_3 olmak üzere 3 sembolümüz varsa 0.5, 0.3 ve 0.2 olasılıklarıyla aralıklar: $x_1 (0,0.5)$, $x_2 (0.5,0.8)$, $x_3 (0.8, 1)$ olabilir. Bu aralık ataması aşağıdaki şekilde gösterilmiştir:



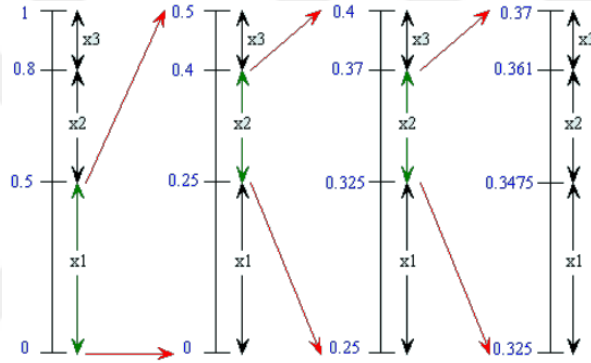
- Kaynağımızın bir sembolünün x_1 olduğunu varsayalım. Sadece 0 ile 0.5 arasında bir sayı üretmek suretiyle bu aralıklarla nasıl kodlayabiliriz? Mesajımız $x_1 x_2$ gibi iki ardışık sembolden oluşuyor olsaydı ne olurdu? Daha sonra Aritmetik kodlayıcı x_1 aralığını $(0,0.5)$ her sembol için aynı oransal aralık uzunluklarıyla alt aralıklarla bölüştürür:



- Bu nedenle, $x_1 x_2$ dizisini temsil eden kod, $(0.25, 0.4]$ aralığındaki herhangi bir sayıdır. Bu kodu, $(0,1]$ eksenindeki aralıkları daha ince bölünmüş olarak düşünebilirsiniz. $x_1 x_2 x_2$ dizisi için kod nedir? Cevap: son x_2 aralığını tekrar böl ve son uygun aralığı bul:



- Dolayısıyla üç sembolden sonra kodlayıcı 0.325 ile 0.37 arasında bir sayı üretmelidir. Çıktının aralığının (0.325, 0.37) olduğunu söyleyebiliriz. Önceki örnekte, bir simge daha yayımlayalım ve $x_1 x_2 x_2 x_3$ dizisini kodlayalım:



- Görünüşe göre, $x_1 x_2 x_2 x_3$ dizisi için aralık (0.361, 0.37).

Buradan çıkarılacak önemli bir sonuç vardır: daha uzun dizileri kodlarken, karşılık gelen aralığı temsil eden sayılar giderek daha kesin hale gelir. Daha kesin rakamlar temsil etmek için daha uzun bit akışları gerekir (Howard and Vitter, 1992).

3.1.3 Shannon-Fano

1948’li yıllarda Claude E. Shannon ve Robert M. Fano ayrı ayrı hafızasız bir kaynaktan etkili bir tanımlama için iki farklı kaynak kodlama algoritması önerdiler. Ne yazık ki, farklı olmasına rağmen, her iki düzen de Shannon-Fano kodlamasıyla aynı ad altında bilinmeye başlamıştı. Shannon-Fano kodlaması, bir dizi sembol ve bunların olasılıklarına dayanan bir ön kod oluşturmaya yönelik bir tekniktir. Kısacası, Huffman Kodlama algoritmasının bir başka versiyonudur. Tek fark, kod kelimesini yaratmaktır. Diğer tüm işlemler, Huffman kodlama algoritmasına ile aynıdır.

Shannon-Fano Algoritması:

Shannon-Fano ağacı, etkili bir kod tablosunu tanımlamak üzere tasarlanmış bir spesifikasyona göre oluşturulmuştur. Gerçek algoritma basitçe şöyledir:

1. Boş kodlarla başlayın.
2. Tüm semboller için frekans istatistiklerini hesaplayın.
3. Kümedeki sembolleri frekansa göre sıralayın, en sık ortaya çıkan semboller solda ve en az ortaya çıkan semboller daima sağda olacak şekilde sıralayın.
4. Listeyi iki parçaya bölün, soldaki toplam frekans sayısı, sağdaki toplama mümkün olduğunca yakın olsun.
5. Listenin sol kısmına ikili sayı 0, sağdaki parçaya 1 rakamı atanır. Böylece, birinci bölümdeki sembollerin kodlarının tümü 0 ile başlasın ve ikinci bölümdeki kodların hepsi 1 ile başlar.
6. Adım 3 ve 4'ü her iki yarımın her birine, alt bölümlere ayrılmış gruplara uygulayın ve her sembol ağaca karşılık gelen bir kod yaprağı oluncaya kadar kodlara bitler ekleyin.

3.1.4 BWCA (Burrows-Wheeler Sıkıştırma Algoritması)

Burrows-Wheeler Algoritması, Michael Burrows ve David Wheeler tarafından 1994 yılında “A Blok Sıralama Kayıpsız Veri Sıkıştırma Algoritması” araştırma raporunda yayınlandı. Bu araştırma raporu 1983 yılı David Wheeler tarafından yayımlanmamış bir esere dayanmaktadır. Burrows-Wheeler Algoritması veri sıkıştırma işlemi için kullanılacaktır. Algoritma birkaç aşamadan oluşur ve bu aşamalar birbiri ardına yapılır. Her aşamadaki giriş ve çıkış keyfi büyüklükteki bloklardır, ancak boyutun tüm sıkıştırma işleminin başında belirtilmesi gerekir. Algoritmanın kod çözme işlemiyle, algoritma tarafından sıkıştırılan veriler orijinal verisine geri döndürülebilir. Bu arada, algoritmanın sayısız farklı versiyonu vardır.

Burrows-Wheeler sıkıştırma algoritması art arda uygulanan üç farklı algoritmik bileşenden oluşur:

1. Burrows-Wheeler dönüşümü: Normal bir İngilizce metin dosyası verildiğinde, aynı harf serisinin birçok kez birbirlerine yakın olduğu bir metin dosyasına dönüştürün.
2. Taşınacak ön kodlama: Aynı harf serisinin birçok kez birbiri ardına ortaya çıktığı bir metin dosyası verildiğinde, küçük tamsayıların büyük harflerden daha sık görüldüğü bir metin dosyasına dönüştürün.
3. Huffman kodlaması: Bazı sembollerin diğerlerinden daha sık görüldüğü bir metin dosyası verildiğinde, sık kullanılan harfleri kısa kod sözcükleri ve az kullanılan harfleri uzun kod sözcükleri ile kodlayarak sıkıştırın.

Mesajı sıkıştıran tek adım son adımdır. İlk iki adım, özellikle bazı sembollerin diğerlerinden daha sık görüldüğü bir mesajla sonuçlandığı için etkilidir. Bir mesajın kodunu çözmek için, işlemleri ters sırada uygulayın: önce Huffman kod çözme işlemini, ardından taşınacak kod çözme işlemini ve son olarak ters Burrows-Wheeler dönüşümünü uygulayın. Burrows-Wheeler dönüşümünün amacı, bir iletiyi sıkıştırmak değil, daha çok sıkıştırmaya daha uygun bir biçimde dönüştürmektir. Dönüşüm, özgün girdiyi kurtarmanın mümkün olduğu şekilde girdideki karakterleri yeniden düzenleyerek tekrarlanan karakterler içeren çok sayıda küme oluşturur. Aşağıda “abracadabra” metin iletisi için algoritmamızın nasıl çalıştığını görelim.

i	Original suffixes	Sorted suffixes	t	next
0	abracadabra	aabracadabr		2
1	bracadabraa	abraabracad		5
*2	racadabraab	abracadabra		6
3	acadabraabr	acadabraabr		7
4	cadabraabra	adabraabrac		8
5	adabraabrac	braabracada		9
6	dabraabraca	bracadabraa		10

7	abraabracad	cadabraabra	4
8	braabracada	dabraabraca	1
9	raabracadab	raabracadab	0
10	aabracadabr	racadabraab	3

Burrows Wheeler dönüşümü t[] dizininin orijinal dizinin görüldüğü son eklenen satıra geleceği, listenin eklenmiş halinin son sütunudur, 2 rdarcaaaabb

Burada sırayla 4 a ve ardışık 2 b - dosyanın sıkıştırılmasını kolaylaştırır.

Burrows-Wheeler dönüşümü ters çevirme: Şimdi Burrows Wheeler dönüşümü geri almayı ve orijinal mesajı nasıl kurtaracağımızı açıklıyoruz. Birinci ekte yukarıdaki i numaralı satırı eklerseniz, bir next[i], j + 1'in son ekinin (j ekini, sağa bir karakter kaydırdı) bulunduğu satırları kaydeder. next[i] dizisini bilmek, aşağıdaki C kodunda olduğu gibi çözmeyi kolaylaştırır:

```
int x = 2;
```

```
int next[11] = { 2, 5, 6, 7, 8, 9, 10, 4, 1, 0, 3 };
```

```
char t[11] = "rdarcaaaabb";
```

```
for (i = 0; i < 11; i++)
```

```
putchar(t[x = next[x]]);
```

Şaşırtıcı bir şekilde, Burrows-Wheeler dönüşümü içerdiği bilgiler next[] ve dolayısıyla orijinal mesajı yeniden oluşturmak için yeterlidir. İlk olarak, orijinal mesajdaki tüm karakterleri yanlış sırada değiştirilmiş olsa bile bilebiliriz. Bu, karakterleri sıralayarak son eklenen listedeki ilk sütunu yeniden oluşturmamızı sağlar. c yalnızca mesajda bir kez olduğundan ve ekler döngüsel çevreleme kullanılarak oluşturulduğundan, next [7] = 4 olacağı sonucuna varabiliriz. Benzer şekilde, d yalnızca bir kez oluşur, dolayısıyla next [8] = 1'i çıkarabiliriz.

i	SortedSuffixes t	next
0	a ?????????r	
1	a ?????????d	
*2	a ?????????a	
3	a ?????????r	
4	a ?????????c	
5	b ?????????a	
6	b ?????????a	
7	c ?????????a	4
8	d ?????????a	1
9	r ?????????b	
10	r ?????????b	

Bununla birlikte, r iki kez görünür, $next[9] = 0$ ve $next[10] = 3$ olsun, ya da $next[9] = 3$ ve $next[10] = 0$ olsun, belirsiz gibi görünebilir. İşte belirsizliği gidermek için anahtar kural:

Satır i ve j aynı harfle başlıyor ise ve $i < j$ ise, $next[i] < next[j]$ dir.

Bu kural $next[9] = 0$ ve $next[10] = 3$ olduğunu ima eder. Ancak neden bu kural geçerlidir? Satırlarda, satır 9 sözlüğü satır 10'dan daha az olacak şekilde sıralanır. Böylece satır 9'daki dokuz bilinmeyen karakter, satır 10'daki dokuz bilinmeyen karakterden daha az olmalıdır (her ikisi de r ile başlar). Ayrıca, r ile biten iki satır arasında 0 satırının 3 satırından küçük olduğunu da biliyoruz. Ancak satır 9 ve 10'daki dokuz bilinmeyen karakter tam olarak 0 ve 3 satırlarındaki ilk dokuz karakterdir. Böylece, $next[9] = 0$ ve $next[10] = 3$ veya bu, son eklerin sıralandığı gerçeği ile çelişecektir (Princeton University, 2018).

3.2 Dinamik Sözlük Tabanlı Teknikler

3.2.1 Lempel - ziv algoritmaları

Lempel-Ziv algoritmaları, daha önce görülen dizelerden oluşan bir sözlük oluşturarak sıkıştırma işlemi yapar. Her karakterin olasılığını tahmin etmek için sözlüğü kullanan ve her bir karakteri bağlamına göre ayrı ayrı kodlayan PPM'den farklı olarak, Lempel-Ziv algoritmaları farklı uzunluklarda karakter gruplarını kodlar. Ayrıca özgün algoritmalarla olasılıkları kullanmazlar. Gzip gibi daha yeni varyantlardan bazıları olasılıklardan bir miktar avantaj elde eder.

Algoritmanın iki ana değişkeni, Ziv ve Lempel tarafından 1977 ve 1978'de iki ayrı makalede anlatılmıştır ve genellikle LZ77 ve LZ78 olarak adlandırılmaktadır. Algoritmalar, arama yaptıkları zamana ve eşleşmeleri nasıl bulduklarına göre farklılık gösterir. LZ77 algoritması sürgülü pencere fikrine dayanır. Algoritma yalnızca geçerli konumdan sabit aralıklarla uzaktaki bir penceredeki eşleşmeleri arar. Gzip, ZIP ve V.42bis (standart bir modem protokolü) hepsi LZ77 üzerine kurulmuştur. LZ78 algoritması, sözlüğe dizeleri eklemek için daha tutucu bir yaklaşımı temel alır. Unix sıkıştırılmış ve Gif biçiminin her ikisi de LZ78'e dayanır (Zeeh, 2003).

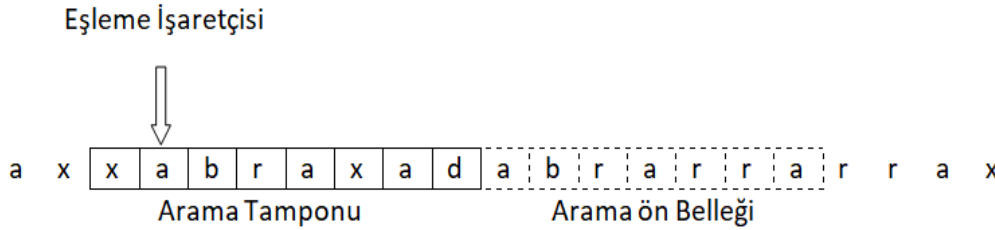
3.2.2 LZ – 77

Abraham Lempel ve Jakob Ziv tarafından 1977-78 yılları arasında yapılan çalışmanın ürünü olduğu için LZ77 adı verilmiştir. Sınırsız veri akışının (yani verinin gerçek boyutu bilinmeden) kayıpsız bir sıkıştırma yaparak daha az yer kaplayacak şekilde kodlanmasını sağlar (Choudhary et al., 2015; Diri, 1999). LZ77 sıkıştırma algoritması en sık kullanılan sıkıştırma algoritması olup, temelinde PkZip ve birkaç algoritma bulunmaktadır. LZ77, bir metin dosyasındaki sözcüklerin ve cümlelerin tekrar edilmesinin muhtemel olduğunu iddia eder. Yineleme olduğunda, işaretçiyi eşleştirecek karakter sayısına göre izleyerek, önceki bir gösterime işaretçi olarak kodlanabilirler. Kaynak hakkında önceden bilgi sahibi olmayı gerektirmeyen ve kaynağın karakteristikleri hakkında hiçbir varsayım yapılmasına ihtiyaç duyulmayan çok basit bir tekniktir (Goemans, 2015).

Verinin ilk defa verimli şekilde aktarımının yolunu açması, günümüz internetinin yaygınlaşmasındaki ivmeyi beraberinde getirmiştir (Choudhary et al., 2015; Diri, 1999).

Günümüzde kullandığımız birçok sıkıştırma-kodlama algoritmasının temelinde yer almaktadır: zip, bzip, gzip, WinRAR, LZMA, LZO, LZ4, JPEG, PNG, GIF (Choudhary et al., 2015; Diri, 1999).

LZ77 algoritması ve versiyonları imleçle birlikte hareket eden kayan bir pencere kullanır (Blelloch, 2001). LZ77 yaklaşımında sözlük, önceden kodlanmış dizinin bir parçası olarak çalışır. Encoder, iki bölümden oluşan sürmeli pencere servisine basarak giriş sırasını inceler: Arama arabelleği (search buffer) ve ön-görüntü (lookahead buffer) arabelleği. Bir arama arabelleği, en son kodlanmış dizinin bir bölümünü içerir ve bir ön-görüntü arabelleği, dizinin kodlanacağı sonraki bölümünü içerir. Şekil 2.1’de, arama ön belleği yedi sembol içerirken, arama tamponu sekiz sembol içermektedir. Uygulamada, tamponların boyutları daha büyüktür. Bununla birlikte, açıklama amacıyla, arabellek boyutu küçük tutulur. Sekansı göz önüne getiren arabellekte kodlamak için, kodlayıcı, bir arama imlecini, ön ilerleme arabelleğindeki ilk simgeye eşleşen bir durum bulana kadar arama arabelleğine geri getirir.



Şekil 3.1: LZ77 uygulamasını kullanarak kodlama

İşaretçinin ön görüntü arabelleğinden uzaklığı ofset olarak adlandırılır. Ardından, kodlayıcı işaretçinin bulunduğu konumdaki simgeyi takip eden sembollerin, ön tampondaki ardışık sembollerle eşleşip uyuşmadığını görmek için inceler. Arama tamponundaki ardışık sembollerle eşleşen ardışık sembollerin sayısı, birinci sembolle başlayarak, ileriye dönük tamponda eşleşme uzunluğu olarak adlandırılır. Kodlayıcı, arama tamponunu en uzun eşleşme için arar. En uzun eşleşme bulunduktan sonra, kodlayıcı onu üçlü (o, l, c) ile kodlar; burada o, ofsettir, l, eşleşme uzunluğudur ve c, arama sırasındaki sembole karşılık gelen kod kelimesidir. Eşleşmeyi takip eden ön tampon örneğin, Şekil 3.1’de işaretçi en uzun eşleşmenin başlangıcını gösteriyor. Bu durumda ofset 7, eşlemenin uzunluğu 4’tür. Üçüncü unsurun üçlü halinde gönderilmesinin nedeni, ön tampondaki sembol eşleşmesinin arama tamponunda bulunamadığı duruma dikkat etmektir. Bu durumda, uzaklık ve eşleşme uzunluğu değerleri 0’a ayarlanır ve üçlü ögenin üçüncü ögesi simgenin kendisi için olan koddur. Arama arabelleğinin boyutu S,

pencerenin boyutu (arama ve ön okuma arabellekleri) W 'dir. Kaynak alfabe boyutu A 'dır ve üçlü kodlamanın sabit uzunlukta kullanılması gereken bit sayısıdır. Kodları, $\lceil \log_2 S \rceil + \lceil \log_2 W \rceil + \lceil \log_2 A \rceil$ dir. İkinci terimin $\lceil \log_2 W \rceil$ değil $\lceil \log_2 S \rceil$ olduğuna dikkat edilmelidir. Bunun sebebi eşleşme uzunluğunun aslında arama arabelleğinin uzunluğunu aşabilmesidir. Bunun bir örnek üzerinde nasıl olacağını görelim:

Aşağıdaki örnekte, kodlama işlemi sırasında karşılaşılabilecek üç farklı olasılığa bakacağız:

1. Pencerede kodlanacak bir sonraki karakter için eşleşme yoktur.
2. Bir eşleşme var.
3. Eşleşen dizge ön ilerleme arabelleğinin içinde uzanır.

Örnek:

Kodlanacak dizin,

..... cabracadabrarrarrad

Pencerenin uzunluğu 13, ön ilerleme tamponunun boyutu 6 ve şu anki durum şu şekilde:

<i>cabraca</i>	<i>dabrar</i>
----------------	---------------

Ön tamponda “dabrar” ile birlikte d için bir eşleşme bulmak amacıyla pencerenin kodlanmış bölümüne geri dönüyoruz. Gördüğümüz gibi eşleşme yok, bu nedenle üçlü $\langle 0, 0, C(d) \rangle$ iletiyoruz. Üçlünün ilk iki elemanı, arama tamponunda d ile eşleşme olmadığını gösterirken $C(d)$, karakter d için olan koddur. Bu, tek bir karakteri kodlamak için çok zahmetli bir yoldur. Bu nedenle, ilerleyen bölümlerde bu konuya değinilecektir.

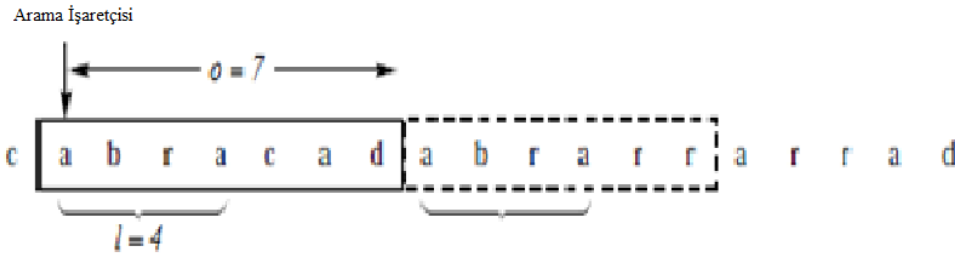
Kodlama işlemine devam edecek olursak, tek bir karakteri şifrelediğimiz için pencereyi bir karakterle hareket ettiriyoruz. Şimdi tamponun içeriği aşağıdaki gibidir:

abracad	abrarr
---------	--------

Önceden tamponda abrarr ile geçerli yerden geriye dönüp baktığımızda, a ile ikisinin ofsetinde bir eşleşme buluyoruz. Bu eşleşmenin uzunluğu bir daha geriye dönüp baktığımızda, a için, dörtte bir ofsette başka bir eşleşme vardır ve yine eşleşmenin uzunluğu 1'dir. Pencerede daha da geriye dönüp baktığımızda, a için yedide bir ofsete sahiptik. Bununla birlikte, bu sefer eşleşme uzunluğu 4'dür. Bu nedenle abra'yı üçlü (7, 4, C (r)) ile kodlarız ve pencereyi beş karakter ileri doğru hareket ettiririz. Pencere şimdi aşağıdaki karakterleri içermektedir:

adabrar	rarrad
---------	--------

Şimdi ön tampon, rarrad dizisini içerir. Pencereye geri dönüp baktığımızda r için bir ofset ve bir eşleşme uzunluğunda bir eşleşme buluruz ve üçte bir ofsette ikinci bir eşleşme var olur. Üç yerine bir beş eşleşme uzunluğunu kullanabileceğimiz ortaya çıktı.

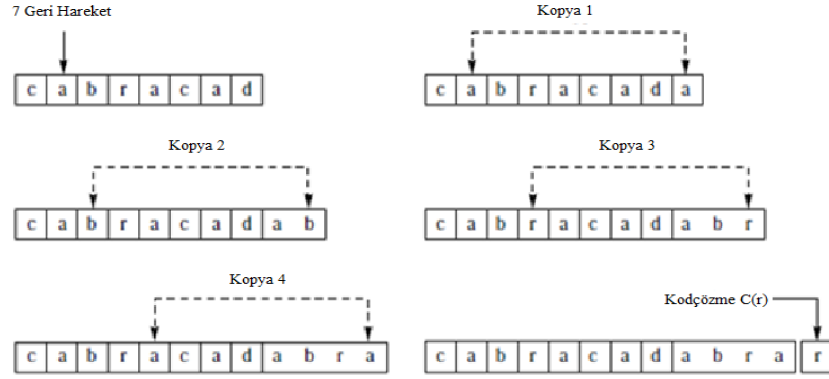


Şekil 3.2: Kodlama süreci

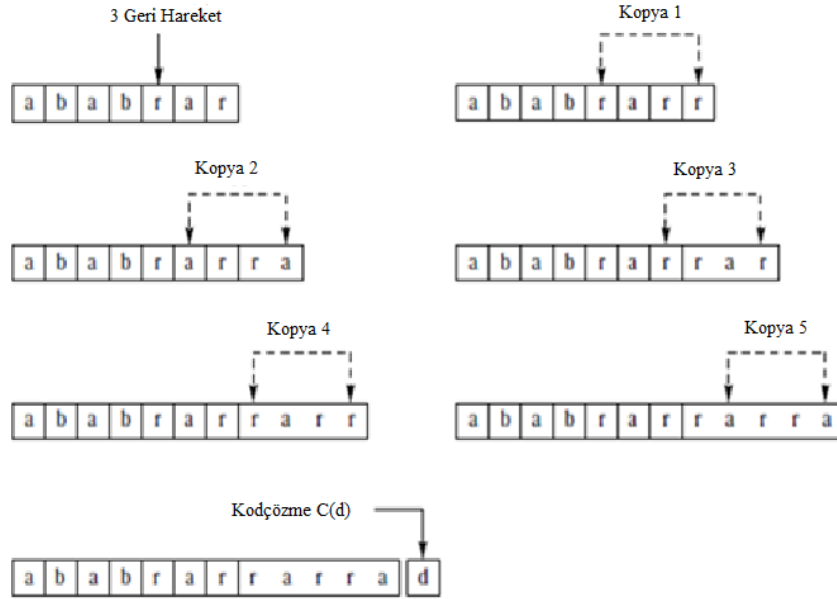
Bu diziyi çözdüğümüzde nedeni netleşecektir. Kod çözme işleminin nasıl çalıştığını görmek için, dizi cabraca'yı kodladığımızı varsayalım ve üçlü (0, 0, C (d)), (7, 4, C (r)) ve (3, 5, C(d)). İlk üçlüyü çözmek kolaydır; Önceden çözülmüş dizgenin içinde bir eşleşme yoktu ve bir sonraki simge d oldu. Kod çözülmüş tel hali şimdi cabracad. Sonraki üçlüdeki ilk öge decode, kopya işaretçisini yedi karaktere geri taşımasını ve o noktadan dört karakter kopyalamasını söyler. Kod çözme işlemi, Şekil 3.2'de gösterildiği gibi çalışır.

Son olarak, (3, 5, C (d)) üçlünün nasıl çözüldüğünü görelim. Üç karakter geri taşıyoruz ve kopyalamaya başlıyoruz. Kopyaladığımız ilk üç karakter rar. Kopyalayıcı işaretçisi, Şekil 3.2'de gösterildiği gibi yeniden kopyalanan r karakterini kopyalamak için bir kez daha hareket eder. Benzer şekilde, bir sonraki karakteri kopyalayacağız. Tek üç karakter kopyalamaya başladık, ancak beş karakteri çözdük. Eşleşmenin yalnızca arama arabelleğinde başlatılması gerektiğine

dikkat edilmelidir. Ön görünüş arabelleğine kadar uzanabilir. Aslında, ileriye dönük ön bellek tamponundaki son karakter d yerine r, daha sonra birkaç tekrar tekrar rar izledi, tekrarlanan rar'ların tüm dizisi tek bir üçlü ile kodlanmış olabilirdi (Sayood, 2005).



Şekil 3.3: (7, 4, C(r)) Üçlüsünün kod çözümü



Şekil 3.4: (7, 4, C(r)) üçlüsünün kod çözümü

Gördüğümüz gibi, LZ77 şeması kaynak hakkında önceden bilgi sahibi olmayı gerektirmeyen ve kaynağın özellikleri hakkında herhangi bir varsayım gerektirmeyen çok basit bir uyarlamalı şemadır. Bu algoritmanın yazarları, asimptotik olarak, bu algoritmanın performansının, kaynak istatistikleriyle ilgili tam bilgiye sahip bir şema kullanarak elde edilebilecek en iyi yola yaklaştığını

gösterdi. Bu asimptotik olarak doğru olabileceği gibi, pratikte burada açıklandığı gibi LZ77 algoritmasının performansını arttırmanın birkaç yolu vardır. Ayrıca, dizinin son bölümlerini kullanarak, burada kullanılan türler varsayımı vardır - yani, bu modeller birbirini “tekrar” yinelemektedir. LZ-78’de yazarlar bu “varsayımı” kaldırdılar ve tamamen farklı bir uyarlamalı sözlük tabanlı şema buldular (Sayood, 2005).

3.2.3 LZ – 78

Lempel ve Ziv’in LZ78 diye adlandırdığı algoritma, 1978 tarihli bir makalede sundukları sözlük tabanlı bir sıkıştırma algoritmasıdır (LZ77 algoritması ile aynı genel fikri temel almıştır, ancak uygulama ayrıntılarında oldukça farklıdır.) LZ78 algoritması, metinde görülen “cümleler” olarak adlandıracağımız bir alt dize sözlüğü oluşturarak çalışır. LZ78 algoritması anında sözlüğünü oluşturur, yalnızca verileri bir kez geçirir. Bu, kodlamaya başlamadan önce tüm belgeyi almak zorunda olmadığınız anlamına gelir. Algoritma diziyi farklı cümlelere ayırır. Yani açgözlü bir algoritmadır (Goemans, 2015; Weinberger et al., 1999).

Algoritma tarafından çıktılanan kod sözcükleri, <i, c> iki öğesinden oluşur; burada ‘i’, en uzun eşleşen sözlük girişine ve ilk eşleşen simgeye atıfta bulunan bir dizindir. Depolama iletim için kod sözcüğünün çıktısına ek olarak, algoritma da dizine ve simge çifti sözlüğe ekler. Sözlükte henüz bulunmayan bir sembol olduğunda, kod sözcüğü dizin değeri 0’dır ve sözlükte de eklenir. Algoritma, bu yöntemle bir sözlük oluşturur. LZ78 için algoritma aşağıda verilmiştir (Schiller, 2012):

w:=NIL;

while(there is input){

K:=next symbol from input;

If(wK exists in the dictionary){

w:=wK;

}else{

Output(index(w),K);

Add wK to the dictionary;

w:=NIL}

}}

Örneğin; Elimizde AABABBBABAABABBBABBABB dizesi olsun.

Daha önce görmediğimiz en kısa cümle ile başlarız. Bu her zaman tek bir harf olacak, bu durumda A:

A|ABABBBABAABABBBABBABB

Artık görmediğimiz bir sonraki cümleyi alıyoruz. Daha önce bir A gördük, bu yüzden AB'yi alıyoruz:

A|AB|ABBBABAABABBBABBABB

Görmediğimiz bir sonraki cümle AB'yi daha önce gördüğümüz gibi ABB'dir. Devam edersek bundan sonra da elde edeceğiz:

A|AB|ABB|BABAABABBBABBABB

ve dizenin kalan kısmının ayrıştırıldığını kontrol edebilirsiniz.

A|AB|ABB|B|ABA|ABAB|BB|ABBA|BB

Harfler bittiğinden sonunda son ifade tekrarlanan bir deyimdir. Şimdi, bunu nasıl kodlayacağız? Gördüğümüz her cümle için onu sözlükte saklarız. Bir dahaki sefer bunu göndermek istediğimizde, tüm cümleyi değil, sadece bu cümlelerin sayısını göndermiyoruz. Aşağıdaki tabloyu düşünün

1	2	3		4	5	6	7	8	9
A	AB	ABB		B	ABA	ABAB	BB	ABBA	BB
A	1B	2B	0B	2A	5B	4B	3A		7

İlk satır, ikonların sayılarını verir (ikili olarak düşünmeniz gerekir), ikinci satır cümleleri, üçüncü satır ise kodlamalarını verir. Yani ABAB'yi altıncı cümleden kodladığımızda onu 5B olarak kodlarız (Kodlamayı tamamlamak için ikisini de ikili olarak kodlarız.) Beşinci ifade ABAB olduğu için bu ABAB ile eşlenir ve biz ona B ekleriz. Burada, boş küme; 0'ıncı cümle olarak kabul edilir ve 0 ile kodlanır. Algoritmanın son adımı bu dizeyi ikili kodlamaktır.

01110100101001011100101100111

Nasıl çalıştığını görmek için bölücü ve virgöl (LZ78'in gerçek çıktısında bulunmayan) ekleyelim ve bunu daha anlaşılır kılalım.

,0|1,1|10,1|00,1|010,0|101,1|100,1|011,0|0111

Önceki dizinin üçüncü satırını alıp, tüm sayıları virgülden önce ve ikili olarak harfleri (virgülden sonra) ifade ettik. A'yı 0'a ve B'yi 1'e eşlediğimize dikkat edin. Daha büyük bir alfabeniz olsaydı, harfleri birden fazla kodlardınız. Aynı zamanda 2'yi (ikinci ifadeye başvurarak) 10 ve 010'a eşlediğimizi de unutmayın. Bir cümleye yapılan bir referans, k bitlerini (yani, $2^{k-1} + 1$ sözlük elemanından başlayarak) cümleleri kodlamak için k bitleri kullandık, böylece virgölün önünde kullanılan bit sayısı arttı. Bu, kod çözme algoritmasının virgöl ve bölücüyü nereye koyacağını bilmesini sağlar. Aksi halde alıcı, bir '10' ın ikinci cümlecik için mi yoksa dördüncü cümleyi ifade eden '100' ilk iki biti olup olmadığını anlayamaz (Goemans, 2015).

Bu durumda, sıkıştırma algoritmasının aslında daha uzun sürdüğünü fark edebilirsiniz. Bu, iki nedenden biri için geçerli olabilir. Ya bu orijinal dizi çok fazla sıkıştırılamayacak kadar rastgele ya da Lempel-Ziv'in asimptotik ortamı göze çarpmaya başlaması için çok kısa idi.

Çözmek için kod çözücünün aynı sözlüğü oluşturması gerekir. Bunu yapmak için, aldığı ikili diziyi alıp bölücü ve virgöl ekler. İlk bölücü biraz sonra gelir, bir sonraki bölme 2 bit sonra gelir. Sonraki iki bit 3 bit sonra gelir. Daha sonra 22 bit uzunluğunda 4 bit, 23 uzunluk 5 bit, 24 uzunluk 6 bit ve genel olarak 2^k uzunluk $k + 2$ bit elde ediyoruz. İfadeler sözlüğü verilir. Örneğin, yukarıdaki dize için sözlük aşağıda verilmiştir.

$\emptyset$	0
A	1
AB	2
ABB	3
B	4
ABA	5
ABAB	6
BB	7
ABBA	8

Boş ifade $\emptyset$ sözlükte her zaman 0 ile kodlanır. Kelimeleri kodladığımızda, sözlükte r cümlesi varsa, (boş ifade $\emptyset$ dahil olmak üzere) cümle numarasını kodlamak için $\lceil \log_2 r \rceil$ bitlerini kullandığımızı hatırlayın. (hatırlatma $\lceil x \rceil$, x 'den büyük olan en küçük tam sayıdır.) Bu, kod çözücünün her bir ifade içinde kaç bit olduğunu tam olarak bilmesini sağlar. Yukarıdaki örnekte, ilk kez AB'yi (cümle 2) kodladık, onu 10 olarak kodladık ve ikinci kez de 010 olarak kodladık. Bunun nedeni, ilk defa, sözlükte üç cümle ve ikinci kez vardı (Goemans, 2015).

Kod çözücü, kodlayıcının yaptığı sözcüğü oluşturmak için aynı algoritmayı kullanır. Bu, kod çözücünün ve kodlayıcının aynı sözlüğü oluşturmasını sağlar. Kod çözücü, cümlecik ibaresinin ne olduğunu anlamaya çalışırken 0 ve 1 cümleciklerini bilir ve bu tam olarak sözlüğü yeniden yapılandırmak için gereken bilgidir (Goemans, 2015).

LZ78 algoritması desenleri yakalama ve süresiz olarak tutma kabiliyetine sahiptir, ancak ciddi dezavantajı da vardır. Sözlük sonsuza dek bağlı olmadan büyümeye devam ediyor. Sözlük boyutunu sınırlamak için çeşitli yöntemler vardır, en kolay girdileri eklemeyi durdurmanın ve statik bir sözlük kodlayıcıya benzer şekilde devam etmenin veya belirli bir sayıda girdikten sonra sıfırdan başlayarak sözlüğün atılmasının sağlanmasıdır. LZ78 tarafından yapılan kodlama, LZ77 ile karşılaştırıldığında hızlıdır ve sözlük tabanlı sıkıştırmanın ana avantajıdır. LZ78 algoritmasının koruduğu önemli özelliği, şifre çözmenin kodlamadan daha hızlı olmasıdır. LZ78'deki kod çözme, sıkıştırma sürecine kıyasla daha hızlıdır (Shanmugasundaram and Lourdasamy, 2011).

3.2.4 LZSS

James Storer ve Thomas Szymanski sıklıkla karşılaşılan Lempel Ziv algoritmasının bir başka versiyonundan sorumludur. Bu versiyon, sıklıkla görülen çeşitli çiftler, üçlüler ve diğer kombinasyonlarla dolu bir sözlük tutmaya çalışmaz. Sadece dosyanın “hareketli penceresi”ni tutar ve bunu bir sözlük olarak kullanır. Sözlükteki “girdilere” değinmek yerine, bu hareket eden penceredeki konuma atıfta bulunur.

Algoritma oldukça basittir. Dosyanın son n baytı pencere olarak korunur. Algoritma, sıkıştırılacak sonraki m baytın başka bir penceresini tutar. Her turda, m baytın en iyi eşleşmesi için n bayt penceresinin tamamına bakar. Bulabiliyorsa, harfler yerine ofseti ve uzunluğu iletir.

Açıkça belirtmek gerekirse, bu algoritmanın en çok zaman harcayan kısmı son n bayt penceresinde arama yapmaktır. n değeri ne kadar büyük olursa, algoritmanın dosyayı sıkıştırmak için ihtiyaç duyduğu zaman artacaktır. n 'nin boyutu kod çözme süresini etkilemez. Bazı veri yapıları bu işleme yardımcı olabilir, ancak bunlar şu an tartışmanın ötesindedir.

Bir programcının çözmesi gereken en büyük sorunlardan biri, bu sıkıştırma algoritması tarafından üretilen simgelerin nasıl temsil edilebileceğidir. Basit bir çözüm, bir değeri kaçış kodu olarak belirlemektir. İyi bir seçim 255 olabilir, çünkü çok sık görünmez. Bir kaçış kodu, sonraki değerlerin ham veriler yerine bir ofset ve bir uzunluğu temsil edeceğini sinyalleştirir. Kaçan kod dosyanın bir parçasıysa ve iletilmesi gerekiyorsa ne olur ? Bu durumda, kodu, kodun kendisi anlamına gelen özel bir ofset ve uzunluk çifti izleyebilir.

Ofsetin ve uzunluğun depolanması için çeşitli olası yaklaşımlar vardır. Basit bir sistem, ofset için bir bayt ve uzunluk için bir bayt ayırır. Bununla çalışmak daha kolay olabilir, ancak boşa olabilir. Çoğu durumda örtüşme nispeten küçük olacaktır. Başka bir çözüm, k biti bloke etmek ve uzunluğu temsil etmek için 4 veya 5'i ve ofseti temsil etmek için $k - 4$ veya $k - 5$ 'i atamaktır. Bu pencere boyutuna bağlıdır.

3.2.5 LWZ

Orijinal LZ78 algoritmasının geliştirilmiş versiyonu belki de en ünlü modifikasyonudur ve bazen yanlışlıkla Lempel Ziv algoritması olarak da anılır. Terry Welch tarafından 1984 yılında yayınlanan LZSS ilkesini, bir sonraki

eşleşmeyen simgeyi LZ78 algoritmasına açıkça göndermeyerek, temel olarak uygular. LZW algoritması dosya sıkıştırma konusunda en popüler algoritmalarından biridir. LZW algoritması açgözlü bir yaklaşım gibidir ve metni alt dizelere böler (Weinberger et al., 1999).

LZW'nin en iyi bilinen iki uygulaması: GIF ve V.42 bis'tir.

Kodlama

LZW yönteminde sıkıştırma işlemi boyunca “sözlük” adı verilen bir tablo tutulur. LZW uyarlanabilir bir tekniktir. Başlangıçta sözlük tablosunun ilk 256 elemanı (0-255) ASCII tablodaki karakterler için ayrılmıştır. Veriler giriş dosyasından byte byte okunup I stringine eklenirken tabloda bu string aranır. Her bir byte eklendikçe I'nın son hali tabloda bakılarak ilerlenir. Herhangi bir noktada I tabloda varken ona eklenen x isimli byte ile birlikte Ix tabloda olmayabilir. Bu durumda I çıkış dosyası yazılır, Ix tabloya eklenir, giriş dosyasından yeni okunacak y isimli byte x'e eklenir ve I=xy tabloda aranır.

Çıktı kodları, girdi verisinden daha az bit sayısına sahiptir. Bu teknik hem grafik görüntüleri hem de sayısallaştırılmış ses için yararlıdır (Sayood, 2005).

Kod Çözme

LZW kod çözme algoritmasında, sıkıştırma algoritmasından çıkan kod çıktı akışını alması ve bunları girdi akışını tam olarak yeniden oluşturmak için kullanması gerekir.

Kod çözme algoritmasında, sözlükte arama yapılacak ve karakteri çıkarılacaktır (Sayood, 2005).

Örnek: LZW algoritması - kodlama

Aşağıda verilen ifadeyi LWZ algoritması ile kodlayalım.
wabbabwabbabwabbabwabbabwoobwoobwoo

Kaynak için alfabenin $\{b, a, b, o, w\}$ olduğunu varsayarsak, LZW sözlüğü başlangıçta Şekil 2.5'e benzemektedir.

İndis	Giriş
1	$\emptyset$
2	a
3	b
4	o
5	w

Şekil 3.5. LZW sözlük başlangıcı

Kodlayıcı önce w harfini görür. Bu “desen” başlangıç sözlüğündedir, bu yüzden bir sonraki harfi bitiştirerek deseni oluşturuyoruz. Bu desen (wa) sözlükte yok, bu yüzden w 'yi sözlük dizini 5 ile kodlamaktadır. Desen wa 'yı sözlüğün altıncı ögesi olarak sözlüğe ekleyin ve a harfinden başlayarak yeni bir desene başlayın. Bir sözlükte olduğu gibi ab kalıbını oluşturmak için bir sonraki b ögesini bitıştirdik. Bu desen sözlükte yok, bu yüzden a 'yı sözlük dizini değeri 2 ile kodlamaktayız, desen ab 'yi sözlüğün yedinci elemanı olarak sözlüğe ekliyoruz ve b harfiyle yeni bir desen oluşturmaya başlayalım. İkinci $wabba$ 'da w harfini bulana kadar iki harfli kalıplar oluşturarak bu şekilde devam ediyoruz. Bu noktada encoder'ın çıktısı tamamen ilk sözlükten gelen indislerden oluşur: 5 2 3 3 2 1. Bu noktadaki sözlük Şekil 2.6'ye benzemektedir. (Sözlüğün 12. girişi halen yapım aşamasındadır.) Dizideki bir sonraki simge a 'dır. Bunu w 'ye bitiştirirken deseni elde ediyoruz. Bu model zaten sözlükte (madde 6) mevcut, bu nedenle sonraki b sembolünü okuyoruz. Bunu wa ile birleştirirsek, şekil örüntüsünü elde ederiz. Bu desen sözlükte mevcut değil, bu nedenle sözlüğün 12. girişi olarak ekliyoruz ve b simgesiyle yeni bir desen başlıyoruz. Ayrıca, indeks değeri 6 olan wa 'yı kodlarız. Bir dizi iki harfli girişten sonra şimdi üç harfli bir girişe sahip olduğumuza dikkat edin. Kodlama ilerledikçe, girişlerin uzunluğu artmaya devam ediyor. Sözlükteki daha uzun girdiler, sözlüğün dizideki yapıyı daha yakaladığını gösterir. Kodlama işleminin sonunda bulunan sözlük Tablo 2.7'de gösterilmektedir. 12. ila 19. girişlerin hepsinin ya üç ya da dört harf uzunluğunda olduğuna dikkat edin. Daha sonra ilk kez desen $\emptyset w$ ile karşılaşırız ve daha sonra artan uzunlukta girdilere geri dönüyoruz (Sayood, 2005).

İndis	İçerik
1	$\emptyset$
2	a
3	b
4	o
5	w
6	wa
7	ab
8	bb
9	ba
10	$a\emptyset$
11	$\emptyset w$
12	$w...$

Şekil 3.6. İlk 12 giriş için LZW sözlüğünün inşası

İndis	İçerik	İndis	İçerik
1	<i>Ø</i>	14	<i>aØw</i>
2	<i>a</i>	15	<i>wabbb</i>
3	<i>b</i>	16	<i>baØb</i>
4	<i>o</i>	17	<i>Øwa</i>
5	<i>w</i>	18	<i>abb</i>
6	<i>wa</i>	19	<i>baØw</i>
7	<i>ab</i>	20	<i>wo</i>
8	<i>bb</i>	21	<i>oo</i>
9	<i>ba</i>	22	<i>aØ</i>
10	<i>aØ</i>	23	<i>Øwo</i>
11	<i>Øw</i>	24	<i>oØb</i>
12	<i>wab</i>	25	<i>Øwoo</i>
13	<i>bba</i>		

Şekil 3.7. Kodlama için LZW sözlüğü

Kodlayıcı çıkış sırası 5 2 3 3 2 1 6 8 10 12 9 11 7 16 5 4 4 11 21 23 4.

Örnek: LZW algoritması - kod çözme

Bu örnekte, önceki örnekteki kodlama çıktısını alıp LZW algoritmasını kullanarak çözeceğiz. Önceki örnekte kodlayıcı çıkış sırası aşağıdaki gibidir:

5 2 3 3 2 1 6 8 10 12 9 11 7 16 5 4 4 11 21 23 4

Bu, kodlama giriş sırası haline gelir. Kod çözücü, kodlayıcıyla aynı başlangıç sözlüğünden başlar. İndeks değeri 5, *w* harfine karşılık gelir, bu nedenle *w*'yi dizimizin ilk ögesi olarak çözeriz. Aynı zamanda, kodlamanın sözlük yapım prosedürünü taklit etmek için, sözlüğün bir sonraki ögesinin yapımına başlanır. *w* harfiyle başlıyoruz. Bu model sözlükte bulunmaktadır, bu nedenle sözlüğe eklemeyiz ve şifre çözme işlemine devam ediyoruz. Bir sonraki kod çözücü girişi, *a* harfine karşılık gelen dizin olan 2'dir. Bir *a* kodunu çözdük ve kalıbı oluşturmak için şu anki kalıbımızı birleştirdik. Bu sözlükte mevcut olmadığı için, sözlüğün altıncı ögesi olarak eklenir ve *a* harfinden başlayarak yeni bir desen başlatırız. Sonraki 4 giriş 3 3 2 1, *bbab* harflerine karşılık gelir ve *ab*, *bb*, *ba* ve *aØ* sözlük girişlerini üretir. Sözlük şu an Şekil 2.8'a benziyor ve burada 11. giriş inşaa aşamasındadır (Sayood, 2005).

İndis	İçerik
1	$\emptyset$
2	a
3	b
4	σ
5	w
6	wa
7	ab
8	bb
9	ba
10	$a\emptyset$
11	$\emptyset. .$

Şekil 3.8: 11 giriş için LZW sözlüğü kod çözüm inşası

Önce w 'yi var olan desene, yani $\emptyset$ 'ye birleştirdik ve desen $\emptyset w$ 'yi oluşturduk. Sözlükte $\emptyset w$ yoksa 11. giriş olur. Yeni desen artık w harfi ile başlıyor. Daha önce, a kodunu çözdük ve şimdi, modeli elde etmek için w 'ye bağladık. Bu desen sözlükte bulunur, bu nedenle bir sonraki girdiyi 8 olarak çözeriz. Bu, sözlükteki bb girişine karşılık gelir. İlk b 'yi çözdük ve kalıbı elde etmek için desene wa bağladık. Bu desen sözlükte mevcut değil, bu yüzden sözlüğün 12. girişi olarak ekleyin ve b harfiyle yeni bir desen başlatın. İkinci b 'nin kodunu çözerek yeni desene birleştirerek bb kalıbını elde ediyoruz. Bu model sözlükte mevcut, bu yüzden encoder çıktı dizisindeki bir sonraki unsurun kodunu çözüyoruz. Bu şekilde devam edersek, tüm diziyi çözebiliriz. Kod çözücü tarafından oluşturulmakta olan sözlüğün, kodlayıcı tarafından oluşturulan sözlüğün özdeşi olması dikkat edilmesi gereken bir noktadır (Sayood, 2005).

3.2.6 LWZR1

LZRW1 algoritması, 1990-1992 arasında Ross N. Williams tarafından oluşturulan yüksek hızlı “LZRW” metin sıkıştırma algoritmalarının ilkidir. Kök adı “LZRW”, “Lempel-Ziv Ross Williams” anlamına gelir. Zaten Welch tarafından alınmış olan “LZW” ismini kullanamamış, bu yüzden “LZRW” kullanmış ve yarattığı algoritmaları numaralandırmıştır. LZ77 ile benzer şekilde çalışmaktadır. Fakat LZ77'ye göre, sıkıştırma oranından biraz taviz verip sıkıştırma ve açma işlemlerini daha hızlandıran bir tablo kullanmaktadır. LZRW2 ve LZRW3 algoritmaları diğer LZRW1 algoritmasından farklı olarak daha çok sıkıştırma yapmakta, fakat bu işlemi gerçekleştirirken daha uzun sürede gerçekleştirmektedir.

3.2.7 DEFLATE

Deflate algoritması Phil Katz tarafından geliştirilmiştir. DEFLATE sıkıştırılmış veri biçimi, ardışık girdi veri bloklarına karşılık gelen bir dizi bloktan

oluşur. Her blok, LZ77 algoritması ve Huffman kodlamasının bir kombinasyonu kullanılarak sıkıştırılır. LZ77 algoritması, tekrarlanan alt dizeleri bulur ve bunları geriye doğru referanslarla (göreceli uzaklık uzaklıkları) değiştirir. LZ77 algoritması, aynı veya daha önceki bloklarda meydana gelen, çoğaltılan bir dizgeye, en fazla 32K giriş baytına bir referans kullanabilir.

Verimlilik şu anda mevcut en iyi genel amaçlı sıkıştırma yöntemleriyle karşılaştırılabilir. Veriler, ara bellekte önceden sınırlı bir miktar kullanıldığında, keyfi uzun aralıklarla sunulan bir giriş veri akışı için bile üretilebilir veya tüketilebilir. Biçim, patent kapsamında olmayan bir şekilde kolaylıkla uygulanabilir.

Patentli olmadığı zamanlarda LZOP, Gzip gibi sıkıştırma programlarında ve PNG görüntü dosyalarında kullanıldığı görülmüştür. Son zamanlarda sık kullanılan sıkıştırma programlarından olan WinZip'te de DEFLATE algoritması kullanılmaktadır (Oswal et al., 2016).

4. GÖRÜNTÜ SIKIŞTIRMA TEKNİKLERİ

Görüntü sıkıştırma, depolama ve iletim sürecinin performansını artırmaya yardımcı olacak, görüntülerin, videoların depolama alanını azaltabilecek bir yöntemdir ve görüntü verisinin gereksiz bilgilerini kaldırmakla ilgilidir.

Görüntü sıkıştırmaya ihtiyaç duymamızın nedenleri:

- Saklama alanını arttırmak.
- Bir görüntünün internet üzerinden gönderilmesi veya web sayfalarından indirilmesi için geçen zamanı azaltmak.
- Görüntü Arşivleme: Uydu Verileri
- Görüntü İletimi: Web Verileri
- Multimedya Uygulamaları: Masaüstü Düzenleme

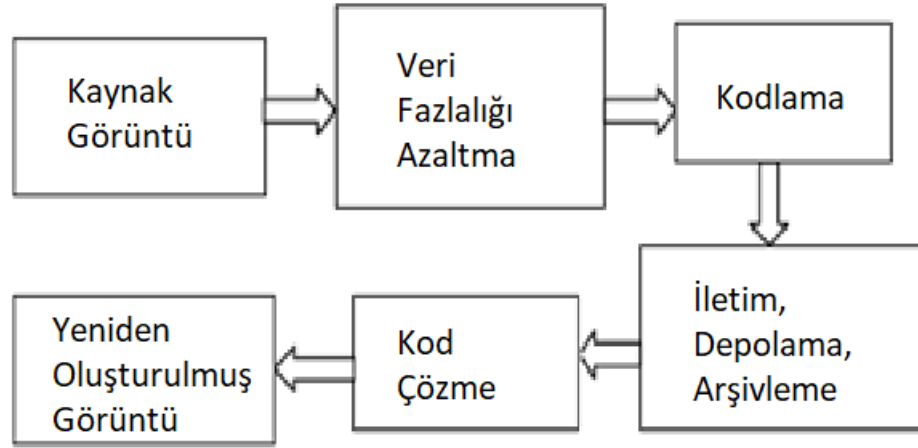
Sıkıştırma işlemi, üç temel veri yedekliliğinden bir veya daha fazlasının kaldırılmasıyla sağlanmaktadır.

(1) Kodlama Yedekliliği: Genellikle sabit bir uzunluğu her bir piksel ile kodlanmış sıkıştırılmamış resim.

(2) İnterpixel Yedekliliği: Bir görüntünün pikselleri arasındaki korelasyonlardan kaynaklanır.

(3) Psiko-görsel Yedekliliği: İnsan görsel sistemi tarafından görmezden gelinen verilere (yani, görsel olarak gerekli olmayan bilgilere) bağlıdır.

Görüntü verilerini sıkıştırma, daha verimli kodlama için artıklıktan yararlanır (Arora and Shukla, 2014).



Şekil 4.1: Görüntü Sıkıştırma Aşamaları

Bir görüntüyü sıkıştırmaya yönelik olağan adımlar şunlardır:

1. Hedef görüntü için Oran (bit mevcut) ve Bozulma (tolere edilebilir hata) parametrelerinin belirtilmesi.
2. Resim verilerinin önemine göre çeşitli sınıflara bölünmesi.
3. Mevcut bit bütçesinin bu sınıflar arasında bölünmesi, bozulmanın minimum olmasıdır.
4. Adım 3’de türetilen bit atama bilgisinin kullanılarak her sınıfın ayrı ayrı nicelenmesi.
5. Her sınıfın bir entropi kodlayıcı kullanılarak ayrı kodlanması ve dosyaya yazılması. Görüntüyü sıkıştırılmış veriden yeniden yapılandırmak, sıkıştırmadan daha hızlı bir işlemdir. İlgili adımlar şunlardır:
6. Bir entropi kod çözücü kullanarak, nicelenmiş verinin dosyadan okunması. (Adım 5’ tersi).
7. Verilerin ters çevrilmesi. (Adım 4’ün tersi).
8. Görüntünün yeniden oluşturulması. (Adım 2’nin tersi).

Görüntü sıkıştırma kayıplı veya kayıpsız olabilir. Kayıpsız sıkıştırma, sıkıştırılmış verilerin orijinal verilerin tam bir kopyası olacağı sıkıştırma işlemini

içerir. Ancak, kayıplı sıkıştırma tekniklerinde, görüntüdeki daha ince ayrıntıların bir kısmı, biraz daha fazla bant genişliği veya depolama alanı kazanmak adına feda edilebilir (Arora and Shukla, 2014).

4.1 Kayıpsız Görüntü Sıkıştırma

Kayıpsız sıkıştırma, arşivleme amacıyla ve genellikle tıbbi görüntüleme, teknik çizimler, küçük resim vb. için tercih edilir. Kayıpsız görüntü sıkıştırma yöntemleri şunlardır:

- Run Length Encoding - PCX’de varsayılan yöntem olarak ve BMP, TGA, TIFF’de mümkün olan bir yöntem olarak kullanılır.
- Entropi kodlaması
- GIF ve TIFF’de kullanılan LZW gibi uyarlanabilir sözlük algoritmaları
- Aritmetik Kodlama
- Alan Kodlaması
- Çok Çözünürlüklü Kodlama
- Kayıpsız Tahmin Edici Kodlama
- JPEG - LS
- JBIG

4.1.1 RLE (Run-length coding)

Her türlü veride kullanılabilen bir algoritmadır ama daha çok görüntülerin sıkıştırılmasında kullanılır. BMP, PCX ve TIFF gibi bazı görüntü formatları, RLE ile sıkıştırıla bilmektedir.

RLE, interpixel yedeklilik üzerinde çalışan bir entropi kodlama sıkıştırma tekniğidir. Bu sıkıştırma, öncelikle resimdeki verilerin bulunmasını sağlar. Veriyi uzunluk, değer çifti ile değiştirir, burada “değer” tekrarlanan değerdir ve “uzunluk”tekrarların sayısıdır. İkili görüntülerde veya Siyah-Beyaz görüntülerde

piksel değeri 0 veya 1'dir. Burada siyah piksel için 0, beyaz piksel için 1 değerleri bulunur. Bu teknik, sıradan gri tonlamalı görüntülerde uzun vadeli bir değer oluşumunun nadir olması nedeniyle iki seviyeli görüntülerin sıkıştırılmasında çok başarılıdır. Bunun bir çözümü, gri tonlamalı görüntüyü bit düzlemlerine ayrıştırmak ve her bit düzlemini ayrı ayrı sıkıştırmaktır. Bu sıkıştırma algoritması, çizimler, logolar ve küçük animasyon dosyaları için uygundur (Ranjeet and Reddy, 2005).

Örneğin, siyah arka planda, 'I' ikili görüntüsünün siyahtaki örneklerini göz önünde bulundurun. Bu hafızada sıfırlar ve birler olarak saklanır. Beyaz için 0 ve siyah için 1 kullanılır.

000000 {0,6}

011110 {0,1},{1,4},{0,1}

001100 {0,2},{1,2},{0,2}

001100 {0,2},{1,2},{0,2}

001100 {0,2},{1,2},{0,2}

011110 {0,1},{1,4},{0,1}

000000 {0,6}

Toplam resimdeki orijinal sembol sayısı: 42

Sıkıştırılmış Görüntüdeki Toplam Sembol Sayısı: 34

Ancak bu kodlama şeması, görüntüler için uygun olmayacaktır (Ranjeet and Reddy, 2005).

4.1.2 Entropi Kodlaması

Entropi, belirli bir miktarda bilgiyi iletmek için gerekli minimum veri kümesini temsil eder. Entropi kodlamasında, artıklık olarak adlandırılan bir özellik, dosyanın boyutunu azaltmak için kullanılır. Bir dosyada genellikle birçok yinelenen karakter bulunur; bu tekrarlanan simgeler aşağıya kaydedilir ve bunları her pikselde tekrarlamak yerine, bu piksellerin konumları kaydedilir ve hepsinin aynı sembolü

bulundurduğu kaydedilir. Dolayısıyla hiçbir bilgi kaybı olmaz(Ranjeet and Reddy, 2005).

4.1.3 LZW Kodlaması

Bu teknikte, görüntüdeki çeşitli desenler bir sözlükte kaydedilir ve ardından orijinal görüntü desendeki kelimeyi sözlükte bağlayarak kodlanır. Bir desen oluşturulduktan sonra, herhangi bir desene rastlandığında, algoritma bu örüntünün sözlükte mevcut olup olmadığını kontrol eder. Evet ise, sözlükteki o desene bir bağlantı verilir. Yoksa o desen sözlüğe eklenir ve daha sonra orijinal dosyaya bağlanır. Dolayısıyla bu algoritma “sözlük tabanlı sıkıştırma algoritması” olarak bilinir. Bu kodlama şeması, çok sayıda yinelenen desen içeren ve daha az sayıdaki tekrarlanan desenli görüntüler için uygun olmayan görüntüler için idealdir, çünkü sözlük, dosyayı daha büyük hale getiren büyük miktarda bellek alanı kaplar (Ranjeet and Reddy, 2005)

Örneğin “entropy kodlaması” metninde “en, op, ng, co, tr, to, di” kalıplarının varlığını göz önünde bulunduralım

1	2	3	4	5	6	7
en	Op	Ng	Co	Tr	to	di

Şekil 4.2: Desen sözlüğü

Kodlanmış metin: “152y 1473”. Bir desenin boyutu iki olarak alınır. Metinleri gruplarken, gruplandırılmayan tek bir öge bulunduğu zaman o öge olduğu gibi yazılır.

Bu tekniğin avantajı, sözlüğün gönderilmesine gerek olmaması, alıcıda otomatik olarak yeniden oluşturulmasıdır (Ranjeet and Reddy, 2005).

4.1.4 Aritmetik Kodlama

Aritmetik kodlama, bir mesajı gerçek sayı hattında 0 ile 1 arasında belirli aralıklarla temsil eden kayıpsız kodlama için sıkıştırma tekniğidir. Temel olarak, 0 ile 1 arasındaki aralıkları mesajın sembollerinin olasılıklarına karşılık gelen daha küçük aralıklara böler. Ardından, ilk sembol daha küçük aralıklarla bölünen bir aralığı seçer. Bir sonraki giriş simgesi bu aralıklardan birini seçer, işlem tekrarlanır. Sonuç olarak, seçilen aralık her sembolle daraltılır ve sonunda, bir mesajı temsil etmek için son aralıktaki herhangi bir sayı kullanılabilir. Sabit bir hassas aritmetik

birimle her sembolü kodlamak için gereken işlem sayısı, dizgenin uzunluğundan bağımsız olarak aynıdır. Böylece, bir dizgeyi kodlamak için yapılan işlemlerin sayısı, uzunluğu ile doğrusal olarak büyür. Çıktı kodundaki her bit aralıktaki giriş kodunun hassasiyetini hassaslaştırır. Aritmetik kodlama, oluşum olasılığına göre sembolleri kodlamak için en verimli yöntemdir ve PPM, JPEG / MPEG (isteğe bağlı olarak), Gzip vb. kullanılır (Ranjeet and Reddy, 2005).

4.1.5 Alan Kodlaması

Alan kodlaması, ikili görüntülerde çalışan bir entropi kodlama şemasıdır. Bu, tüm görüntünün aynı boyuttaki bazı bloklara ayrılması şeklinde çalışır; sonuçtaki blok üç puntodan oluşabilir, yalnızca beyaz piksellere sahip bloklar, yalnızca siyah piksellere sahip bloklar veya hem siyah hem de beyaz piksellere sahip bloklar. Daha sonra sık sık oluşan blok, 1 bitlik bir kod sözcüğü ile atanır ve diğerlerine 2 bitlik kod sözcükleri atanır. Böylece blokları 1-bit veya 2-bit kod sözcükleri ile değiştirerek dosya boyutu azaltılır ve böylece sıkıştırma sağlanır (Ranjeet and Reddy, 2005).

4.1.6 Çok Çözünürlüklü Kodlama

Orijinal görüntünün düşük çözünürlüklü bir sürümüyle başlar ve ardışık olarak daha yüksek çözünürlükler üretmek için piksel değerlerini enterpolasyon yapar. Enterpolasyon değerleri ile gerçek değerler arasındaki hatalar, ilk düşük çözünürlüklü görüntü ile birlikte saklanır. Hem düşük çözünürlüklü görüntü hem de hata değerleri, orijinal görüntüden daha az bit ile depolanabileceğinden, sıkıştırma gerçekleştirilir. Laplacian Pyramid⁴⁹ Burt ve Adelson tarafından geliştirilen başka çözünürlüklü görüntü sıkıştırma yöntemidir. Ortalanmış örnekleme ile orijinal resmin daha düşük çözünürlüklü sürümlerini art arda oluşturur, böylece piksel sayısı her ölçeğinde iki faktör azalır. En düşük çözünürlüklü görüntü ile birlikte ardışık çözünürlük sürümleri arasındaki farklar, orijinal görüntüyü mükemmel bir şekilde yeniden oluşturmak için saklanır ve kullanılır. Ancak, veri değerlerinin sayısı orijinal görüntü boyutunun $4/3$ oranında arttığı için yüksek bir sıkıştırma oranı elde edemez. Genel olarak görüntü, çok çözünürlüklü kodlamada tersine çevrilerek farklı çözünürlük alt-görüntülerinin bir grubuna dönüştürülür. Genellikle, resmin entropisini düşürür. Çoklu çözünürlük yöntemlerinin ağaç yapısını istismar ederek daha fazla sıkıştırma elde etmek için bazı ağaç temsilleri kullanılabilir (Ranjeet and Reddy, 2005).

4.1.7 Kayıpsız Tahmin Edici Kodlama

Kayıpsız öngörülebilir kodlama, komşu piksellerin değerlerini kullanarak her piksel değerini öngörür. Bu nedenle, her piksel orijinal değeri yerine bir tahmin hatası ile kodlanır. Genellikle, hatalar orijinal değere kıyasla çok daha küçüktür, böylece onları saklamak için daha az bit gerekir.

DPCM (diferansiyel puls kod modülasyonu), kayıpsız görüntü sıkıştırma yöntemine dayanan bir tahmin kodlamadır. Aynı zamanda kayıpsız JPEG sıkıştırmasının temelini oluşturur. Kayıpsız öngörü kodlamanın bir varyasyonu, görüntüyü bloklara bölebilen ve yüksek tahmin performansı elde etmek için her bloğun bağımsız olarak tahmin katsayılarını hesaplayan uyarlamalı tahmindir. Yüksek performanslı bir hibrid kodlama algoritması elde etmek için diğer yöntemlerle de birleştirilebilir (Ranjeet and Reddy, 2005).

4.1.8 JPEG – LS

JPEG-LS yeni bir algoritma olmasa da yine de çok etkili ve kullanışlı bir sıkıştırma standardıdır. Bu algoritma, düşük karmaşıklık ve kayıpsız kod çözme üzerine odaklanan Hewlett-Packard'ın LOCO-I şemasından 1998 yılında sürekli tonlu, gri tonlamalı veya renkli dijital hareketsiz görüntüleri sıkıştırmak için görüntüleri kayıpsız veya az kayıplı sıkıştırabilen bir algoritma olarak uyarlandı. LOCO-I algoritması, aritmetik kodlamaya dayanan en son teknoloji şemalarda elde edilenlere benzer veya daha üstün sıkıştırma oranları elde eder. JPEG-LS algoritmasının sıkıştırma oranları, birçok yüksek karmaşıklıkta kayıpsız algoritmalarla karşılaştırılabilir. Buna karşılık, bazı algoritmalar bazı görüntülerle diğerlerinden daha iyi sonuç verir. Sürekli bir görüntü genel olarak JPEG-LS'de en iyi sıkıştırılrsa da diğer herhangi bir görüntü türüyle rekabetçi bir algoritma halini alır.

JPEG-LS, kayıpsız görüntü sıkıştırması için çok özel bir formattır. İnsan gözü son derece küçük kusurları ayırt edebiliyor olsa da, beynimiz bunun çoğunu filtreler ve böylece önemli verilere odaklanmamızı sağlar. Bu, hareketsiz görüntüler ve çok çerçeveli görüntüler (video) gibi analog verilerin normal koşullar altında beynimizin filtreleyebileceği bazı bilgilerin atıldığı formatlara sıkıştırılmasını mümkün kılar. Görüntüyü tanınmayan hale getiren bir noktaya veri kaybetmeden orijinal boyutunun görüntüyü sıkıştırma karmaşıklığı JPEG2000 ortaya çıkıncaya kadar elde edilemedi.

Ticari uygulamalarda bu format çok az kullanılır, çünkü tüm kaliteyi korurken bir görüntünün sıkıştırılması gereken çok uygulama yoktur, ancak NASA için idealdir.

Algoritmanın çalışmasının temel blok diyagramı Şekil 3.4’de gösterilmekte ve aşağıdaki bölümlerde ayrıntılı olarak açıklanmaktadır.

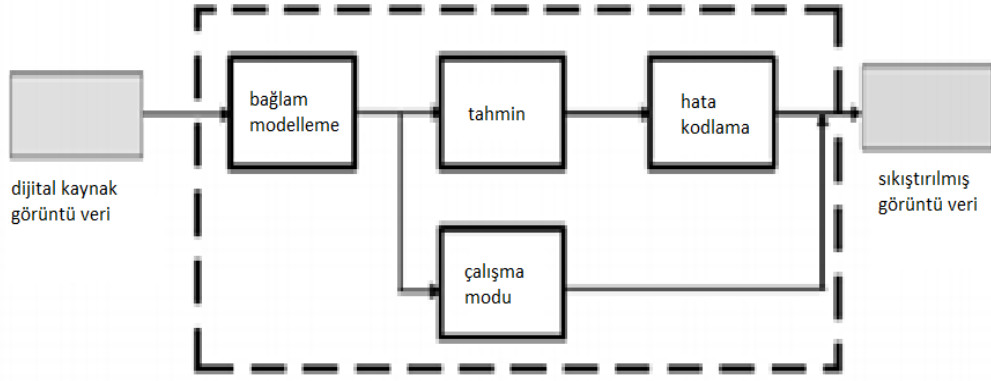
JPEG-LS’yi tanımlarken bahsedilen kayıpsız ve neredeyse kayıpsız kavramlarına bakalım.

- Kayıpsız: bilgi kaybı olmaz
- Neredeyse kayıpsız: yeniden yapılandırılmış görüntü bileşenlerindeki her örnek değeri, orijinal görüntüdeki ilgili değerden önceden belirlenmiş (küçük) bir miktara kadar farklı olduğu garanti edilir.

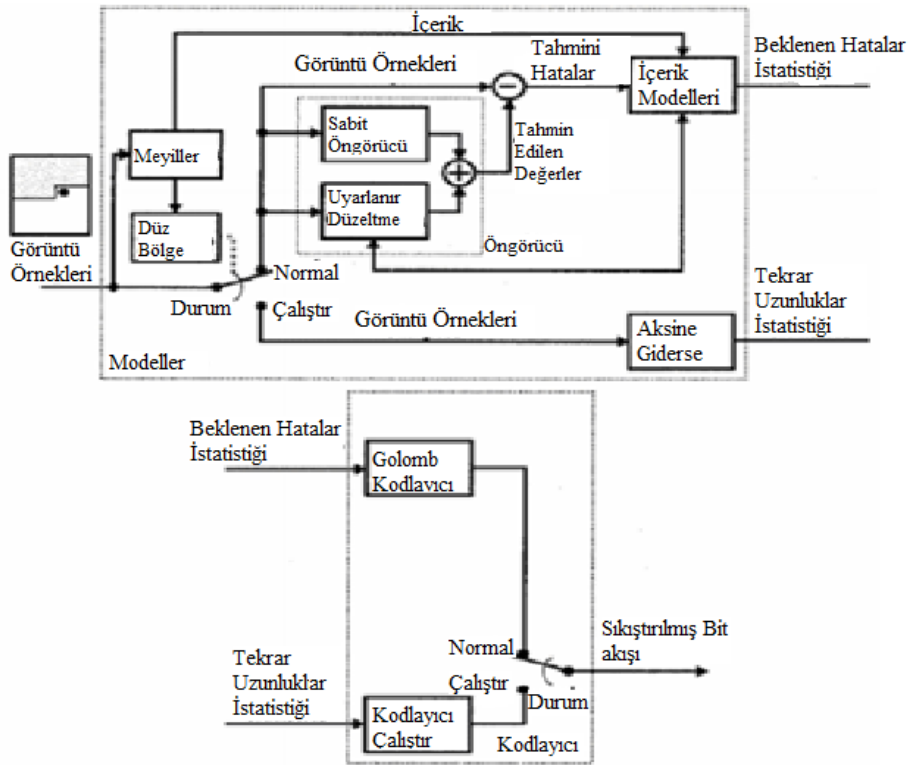
JPEG-LS modelleme şeması bir endüktif çıkarım problemi olarak tanımlanabilir. Algoritmanın başlangıç kısmı, her pikselin komşu pikseller yardımıyla büyüklük açısından ne olacağına dair bir tahmin oluşturulması üzerine tasarlanmıştır. Görüntü verisinin ardışık taranmasından sonra komşu piksel verileri bir olasılık dağılımının oluşturulması için kullanılır. Bu, kod çözücüyü çevreleyen piksellerden belirli bir renk büyüklüğüne dayalı bir piksel oluşturmaya olanak tanır. Bu ilk adım birkaç küçük parçaya bölünür.

Öncelikle, ön tanımlama aşaması, çevreleyen piksellerden olasılığa dayanarak bir pikselin hangi aralıkta olması gerektiğini belirler. İkinci aşama olarak, pikselin içeriği belirlenir ve kaydedilir. “Bağlam” örneği, bir kenar oluştuğunda, büyüklüğün azalması ya da artan büyüklük olurdu. Tabii ki, bu veri taramanın yönüne bağlıdır ve daha sonradan anlatılacaktır. Üçüncü olarak, daha basit bir şekilde modelin hatası olarak bilinen tahmini kalıntı kaydedilir.

Yalnızca hemen komşu pikseller bu ilk bileşenle ilgili olduğundan, algoritmanın çalışması için bir kerede yalnızca birkaç satırın belleğe taranması gerekir. Bu, bellek gereksinimlerini önemli ölçüde azaltacaktır (Weinberger et al., 1999).



Şekil 4.3. Kayıpsız kodlayıcı basitleştirilmiş diyagram



Şekil 4.4. JPEG-LS Algoritmasının blok diyagramı

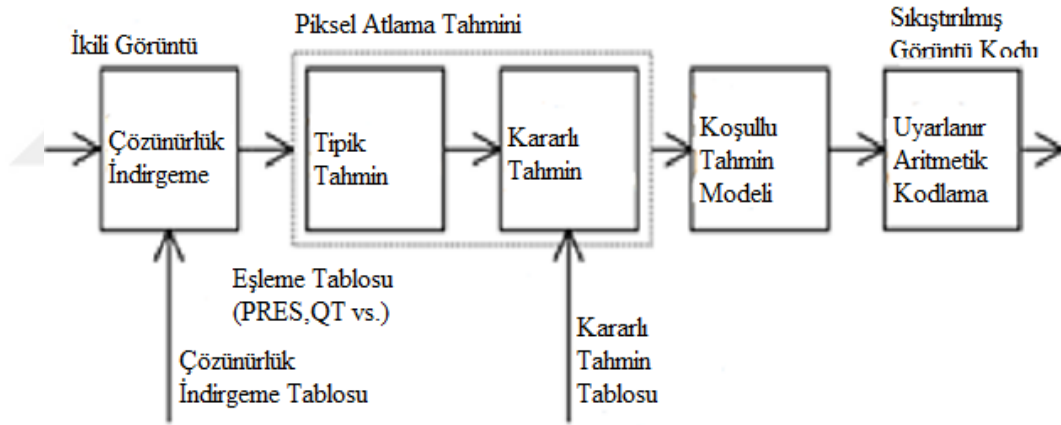
4.1.9 JBIG

JBIG, iki seviyeli (iki renkli) görüntü verilerini sıkıştırmak için kullanılan bir yöntemdir. Kısaltma JBIG, Uluslararası Standartlar Organizasyonu (ISO) içinde yer alan bir standart komitesi olan İki Boyutlu Görüntü Uzmanları Topluluğu ortak grubunun temsilcisidir. Geliştirdikleri sıkıştırma standardı bu komitenin ismini taşımaktadır.

1988’de ISO ve CCITT, ISO / IEC JTC1 / SC29 / WG9 grubuna ve CCITT SGVIII alt grubuna katılarak JBIG’yi kurdu ve iki seviyeli verilerin sıkıştırılmasında standart ve kayıpsız bir yöntem geliştirdi. 1993’de iki seviyeli veri kodlamasının JBIG yöntemini tanımlayan standart tamamlandı. 1993’den sonra faks iletiminde yaygın olarak kullanılmıştır.

JBIG, aritmetik kodlama tabanlı bir tekniktir.

Aşamalı bir JBIG ikili görüntü sıkıştırma kodlayıcısının bir blok diyagramı Şekil 3.5’de verilmektedir. Algoritma, orijinal görüntüyü, çözünürlük olarak indirgenmiş görüntülere dönüştürerek başlar. Her bir görüntü, en düşük çözünürlüklü görüntü ile başlayarak kodlanır. Anlaşılabacağı üzere, piksellerin tipik ve kararlı tahminleri mümkün olduğunda kullanılır, böylece bu tür pikseller aritmetik kodlama aşamasında atlanabilir. Bu şekilde arzu edilen sıkıştırma hızı ile birlikte daha iyi sıkıştırma elde edilir. Boşlanmamış pikseller aritmetik olarak kodlanır.



Şekil 4.5. JBIG sıkıştırma kodlamasının blok diyagramı

İkili görüntülerin hiyerarşik çözünürlüğü azaltma (RR), tek bir görüntüyü kademeli olarak küçük resimlerden oluşan bir piramit haline dönüştürür. JBIG standardına uygun olarak, çözünürlüklerin simetrik olarak iki faktörle azaltıldığını varsayabiliriz. Çözünürlük azaltma piramidinin her katmanı, önceki yüksek çözünürlüklü katmandan ve muhtemelen şu anki katmandan gelen bilgiler kullanılarak oluşturulmuştur. Bir sonraki düşük çözünürlüklü katmandaki her piksel, yüksek çözünürlüklü katmandaki dört pikselin yerini alır. Genellikle, çözünürlük azaltma piramidin her ardışık katmanını oluşturmak için aynı yöntem kullanılır.

JBIG’de önerilen PRES çözünürlüğü azaltma yöntemi QT yönteminden daha karmaşıktır. Amacı, yüksek kaliteli, yani görsel çekici, daha düşük çözünürlüklü görüntüler üretmektir. Çözünürlük azaltılmış piksel değerini belirlemek için PRES yöntemi, aynı katmandan üç piksel ve bir önceki yüksek çözünürlük katmanından dokuz piksel kullanır. Bunun aksine, dörtlü küçültme yöntemi yalnızca dört yüksek çözünürlüklü piksel kullanır.

Şekil 3.6, 256x256 piksellik bir resim üzerinde PRES ve QT yöntemlerinden elde edilen sonuçları göstermektedir. Her ikisinde de çözünürlük düşüşleri varsayılmıştır. Bu resimde ve Şekil 3.7’de çözünürlüğün azaltılmış görüntüleri, çözünürlüklerini temsil eden piksellerle gösterilmektedir. QT düşürmede, ardışık düşük çözünürlüklü görüntüler hızla daha siyah olur ve sonuç olarak PRES azalmasına kıyasla daha hızlı bir kalite kaybına neden olur. QT RR görüntülerinin toplam okunaksızlaşması, PRES RR görüntülerininkinden daha çabuk bir çözünürlük azalmasıyla gerçekleşir. Aritmetik olarak kodlanan piksel sayısı bakımından ölçülen kodlama süresi, daha önce yapıldığı gibi çözünürlük azaltılmış görüntülerin kalitesinden ziyade tasarım objektifimizdir.

Piksel tahmini, aynı görüntünün birden çok gösterimi olan hiyerarşik çözünürlük azaltma katmanlarının yüksek düzeyde ilişkili ve bazen de gereksiz bilgi içerdiğini ortaya çıkarmaktadır. Bu tahmin, atlanarak atıl şekilde kodlanmış piksel sayısını azaltarak sıkıştırma hızını arttırmak için kullanılabilir.

dpi	PRES	% Black	QT	% Black
200		11.18		11.18
100		11.06		13.85
50		11.67		18.43
25		13.09		25.49
12.5		16.02		41.02
6.25		18.75		57.81

Şekil 4.6. PRES veya QT yöntemlerini kullanarak çözünürlüğü azaltılmış görüntüler

D5 PRES TPD	D5 PRES DP	D5 PRES TPD/DP	D5 QT TPD	D5 QT DP	D5 QT TPD/DP

Şekil 4.7. DP, TPD veya TPD / DP tahminini kullanarak PRES ve QT azaltımı için aritmetik olarak kodlanmış pikseller. RR önceki şekildeki ile aynıdır.

4.2 Kayıplı Görüntü Sıkıştırma

Kayıplı görüntü sıkıştırma, bit hızında önemli bir azalma sağlamak için küçük (bazen fark edilmeyen) veri kaybının kabul edilebilir olduğu uygulamalardaki fotoğraflar için kullanılır. Özellikle doğal görüntüler için uygundur.

- Dönüştürme Kodlaması
- Ayrık Kosinüs Dönüşümü (DCT)
- Ayrık Dalgacık Dönüşümü (DWT)
- Fraktal Sıkıştırma
- Blok Kesme Kodlaması
- Altbant Kodlama
- Vektör Niceleme

4.2.1 Dönüştürme Kodlaması

Burada matematiksel dönüşüm görüntü sinyali üzerine uygulanır, bir sinyali alt uzaylara böler ve bazı fazlalık verileri atar. Kalan bilgiler çeşitli yöntemlerle sıkıştırılabilir. Çıktı kodu çözüldüğünde, sonuç, veri sıkıştırma şeması olarak adlandırılan orijinal girdiyle özdeş olmayabilir. Görüntü sıkıştırmada aşağıda bahsedilen dönüşüm şeması vardır.

Bu kaynak kodlama (dönüşüm) şeması, orijinal görüntüdeki piksellerin frekans alan katsayılarına dönüştürüleceği şekilde çalışır. Bu frekans alanı katsayıları, dönüşüm katsayıları olarak bilinir. Bu katsayıların belirli faydalı özellikleri vardır. Örneğin: orijinal verideki azami enerjinin, dönüştürme katsayılarının azında yoğunlaştığı için, enerji sıkıştırma özelliği vardır. Bu katsayılar tek başına seçilir, geriye kalan sıkıştırma elde etmek için silinir. Seçilen katsayılara daha ileri işlem uygulanmaktadır. CT kodlaması, dönüşüm kodlamayı gerçekleştirmek için en yaygın yaklaşım olmuştur. JPEG görüntü sıkıştırma standardında da benimsenmiştir (Ranjeet and Reddy, 2005).

4.2.2 Ayırık Kosinüs Dönüşümü (DCT)

DCT, enerji kompaksiyonunun mükemmel özelliklerinden dolayı görüntü sıkıştırma alanındaki en yeni bilgi dönüşümüdür (Ranjeet and Reddy, 2005; Arora and Shukla, 2014). Dönüştürülecek görüntü, her bloğun n pikselden oluşmuş kare bloklarına bölünür ve her blok, n DCT katsayısına dönüştürülür.

Ayrık kosinüs dönüşümü, $u = 0, 1, I., n-1$ için:

$$F(u) = \frac{2c(u)}{n} \sum_{j=0}^{n-1} f(j) \cos \frac{(2j+1)u\pi}{2 \times n} \quad (3.1)$$

Ters dönüşüm için aşağıdaki tek boyutlu IDCT iki kez uygulanır.

$$f(j) = \sum_{u=0}^{n-1} c(u) F(u) \cos \frac{(2j+1)u\pi}{2 \times n} \quad (3.2)$$

Burada $j=0, 1, \dots, n$ dir.

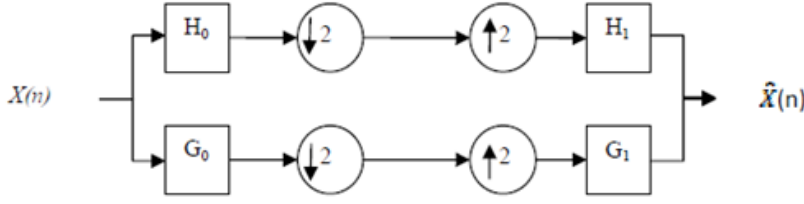
Görüntüye DCT uygulandıktan sonra görüntü için DCT katsayıları elde edilir. Tamsayı olmayan DCT katsayıları, tamsayılara nicelendirilir. Genel olarak, çoğu DCT katsayılarının değeri sıfırdır veya neredeyse sıfırdır. Bu, bazı bilgi kayıpları olduğu anlamına gelir ve yalnızca katsayı kuantumlanması sürecinde oluşur (Krajci et al., 2015). Son on yılda DCT'ye dayanan JPEG görüntü sıkıştırma standardı getirilmiştir (Ranjeet and Reddy, 2005).

4.2.3 Ayırık Dalgacık Dönüşümü (DWT)

JPEG / DCT tabanlı sıkıştırma, düşük bit hızlarında yeniden yapılandırılmış görüntüde blokaj dezavantajına sahiptir (Gomathi and Lotus, 2014). Dalgacık dönüşümü, insan görsel özellikleriyle uyumlu olduğu için görüntü ve video uygulamalarında popüler hale gelmiştir (Sayıt, 2005). Dalgacık kodlama tekniklerinde, bloke edici etki ve bozulma olmadığı için öznel olarak hoş giden görüntü ile sonuçlanır ve JPEG2000 görüntü sıkıştırması için yeni bir standart getirir (Kodituwakku and Amarasinghe, 2010; Weinberger et al., 1999).

Dalgacık tabanlı görüntü kodlamada dalgacık seçimi çok önemlidir ve kodlama performansını ve yeniden yapılandırılmış görüntü kalitesini belirler (White Paper, 2008; Mesut, 2006; Oswal et al., 2016). Bir görüntünün dalgacıklara bölünmesi, bir görüntünün ayrıntılı bölümüne (dalgacık fonksiyonu X) karşılık

gelen yüksek frekansları ve birinin düşük frekansları veya pürüzsüz kısımlarını (ölçekleme fonksiyonu Y) temsil eden bir çift dalga biçimini içerir.



Şekil 4.8. Band dalgacık analizi ve sentezi filtresi

Burada, $X(n)$: girdi görüntüsü ve $\widehat{X}(n)$, yeniden oluşturulmuş görüntü,

H_0 ve H_1 : görüntünün detay katsayıları,

G_0 ve G_1 : görüntünün yaklaşık katsayılarıdır.

Bir x sinyalinin DWT'si bir dizi filtreden geçirilerek hesaplanır. İlk örnek, impulse yanıtı g ile düşük geçişli bir filtreden geçerek iki konvolüsyon oluşturur:

$$y(n) = (x \times g)[n] = \sum_{k=-\infty}^{\infty} x[k]g[n - k] \quad (3.3)$$

Sinyal aynı zamanda bir yüksek geçişli filtre h kullanılarak ayrıştırılır. Çıktı, detay katsayılarına (yüksek geçişli filtreden) ve yaklaşık katsayılarına (Düşük geçişli filtreden) ayrıştırılır. Dolayısıyla:

$$y_{low}[n] = \sum_{k=-\infty}^{\infty} x[k]g[2n - k] \quad (3.4)$$

ve

$$y_{high}[n] = \sum_{k=-\infty}^{\infty} x[k]h[2n + 1 - k] \quad (3.5)$$

Görüntü sıkıştırma için dönüşüm adımları aşağıdaki şekilde açıklanabilir:

Görüntü sıkıştırma için, giriş görüntüsü dönüştürülür ve daha sonra eşik değeri belirlenir ve Entropi kodlayıcı tarafından kodlanır. $F(x, y)$, bir görüntünün gri seviyelerini temsil eden bir $N \times N$ matrisi olsun.

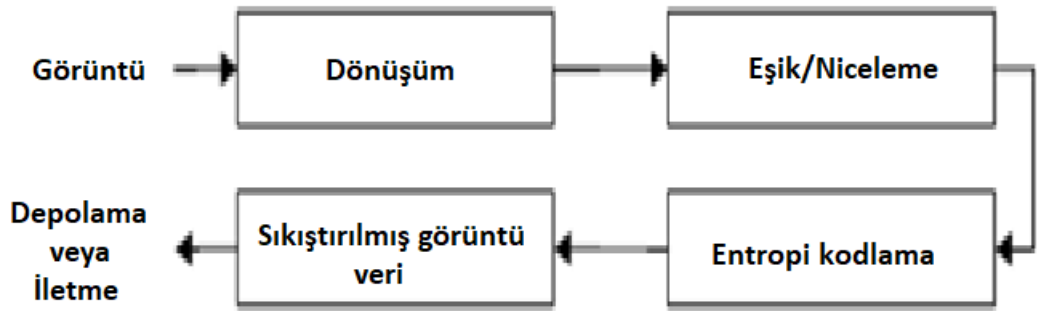
(A) Resim $f(x, y)$ örtüşmeyen $M \times M$ bloklarına bölünür.

(B) Dönüştürme her görüntünün bloğuna uygulanır (DWT için resim bir blok olarak değerlendirilir).

(C) Belirli bir eşik altındaki dönüşüm katsayıları sıfıra ayarlanır. Eşik, tüm blok katsayıları boyunca minimum ve maksimum katsayı değerlerinin bir yüzdesini alır.

(D) Bir entropi kodlama tekniği uygulanır.

(E) Sıkıştırılmış görüntüyü elde etmek için yeniden oluşturulmuş görüntü $F(x, y)$ özgün görüntüyle karşılaştırılır.



Şekil 4.9. Görüntü sıkıştırma için blok diyagramı

Zirve sinyali-gürültü oranı [PSNR] ve sıkıştırma oranı [CR] hesaplanır. Tepe sinyali-gürültü oranı (PSNR), gri ölçekli görüntü için,

$$\text{PSNR[dB]} = 10 \log \left(\frac{x_{\max}^2}{\text{MSE}} \right), X_{\max} = 255$$

ve

$$\text{MSE} = \frac{1}{N \times N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} [f(x, y) - F(x, y)]^2$$

Sıkıştırma oranı (CR) aşağıdaki şekilde tanımlanır:

$$\text{CR} = \frac{\text{orijinal görüntü boyutu}}{\text{sıkıştırılmış görüntü boyutu}}$$

4.2.4 Fraktal Kodlama

Fraktal teknik, görüntüleri kodlamak / çözmek için kullanılır, bir kasılma afin dönüşümü kümesinden oluşan yerel yinelemeli bir fonksiyon sistemi için kolaj teoremi ve sabit nokta teoremine dayanır. Bu kaynak kodlama tekniği, bitmap'leri kodlamakta kullanışlıdır. Fraktal kodlama, gerçek dünya girdi verisinin birçok fraktal içermesi prensibiyle çalışır. Resimlerdeki benzer desenlerin kendiliğinden tekrarlanması fraktallar olarak bilinir. Fraktalların haritalanması tam eşlemeleri gerektirmez, ancak yaklaşık eşlemeler temel alınarak yapılır. Dolayısıyla fraktalların yerini alması, daha az bellek alanı kaplar ve fraktalların yaklaşık eşleşmesinden dolayı elde edilen sıkıştırma kayıplı olur. Bu, diğer kayıplı sıkıştırma tekniklerine kıyasla daha yüksek sıkıştırma oranlarında bile iyi bir sıkıştırma oranı verir; çünkü diğer teknikler, boş veriyi yok etmeyi amaçlarken, fraktal kodlama veriyi atmayı hedeflemez. Fraktal kodlama, sıkıştırmadan sonra kod çözmede daha fazla zamana ihtiyaç duyan asimetric bir tekniktir çünkü sıkıştırma işlemi milyonlarca yineleme kullanarak görüntüdeki çeşitli fraktalları bulması gerekir ve kod çözme işleminde sadece fraktalları bitmap'lere çevirmelidir. Sıkıştırma süresini ve çıktı dosyası boyutunu etkileyen parametreler, giriş görüntüsünün, sıkıştırma oranının ve çıktı görüntüsünün kalitesinin çözünürlüğüdür. Fraktal imge sunumu matematiksel olarak Tekrarlanan İşlev Sistemi (IFS) olarak tanımlanmaktadır. R^2 'nin bir alt kümesi olarak bir ikili görüntü R_b düşünelim. Şimdi görüntünün IFS'sini bulmak istiyoruz. IFS, $\{f_1, f_2, \dots, f_N\}$ eşlemeleri kümesi olarak $f_i : R^2 \rightarrow R^2, 1 \leq i \leq n$ olarak tanımlanabilir. IFS, bir 2D set S 'yi H operatörünün sabit noktası olarak tanımlar. H operatörü Hutchinson operatörünü temsil etmektedir.

$$H(Z) = \bigcup_{j=1}^N f_j(Z), Z \subset R^2 \rightarrow D$$

S , $H(S) = S$ 'yi karşılayan benzersiz kümedir. Bu, $S = f_1(S) \cup f_2(S) \cup \dots \cup f_N(S)$ olduğunu gösterir. Dolayısıyla IFS'nin S 'nin fraktal gösterimi olduğu açıktır. S , Sabit nokta yineleme (FPI) kullanılarak geri kazanılabilir. FPI, boş olmayan başlangıç kümesi Z_0 için, yineleme $Z_{k-1} = H(Z_k)$, S değerini verir. Gri tonlamalı görüntülerde, aynı adımlar izlenir ancak görüntü R , R^3 'ün altkümesi olarak düşünülür.

4.2.5 Blok Kesme Kodlaması

Bu kaynak kodlama tekniğinde, görüntü örtüşmeyen piksel bloklarına bölünür. Eşik değeri her blok için belirlenir. Burada eşik değeri bloktaki değerlerin ortalamasıdır ve $\bar{r}$ olarak adlandırılır.

$$\bar{r} = \frac{1}{N} \sum_{i=1}^N r_i \rightarrow (E)$$

r_i , piksellerin gri değeridir. N , bloğun toplam piksel sayısıdır.

$\bar{r}$ 'den büyük değerler 1 ile değiştirilir, $\bar{r}$ 'den küçük değerler 0 ile değiştirilir ve $\bar{r}$ ile aynı değere sahip olan pikseller, bunu uygulayan sıkıştırma algoritması gereksinimine bağlı olarak 0 ya da 1 ile değiştirilir. Bu işlem sonucunda bir kısmı çıktı elde edilir. Kesit olarak ayrılmış görüntüye sahip olunur. Her segment sıfırlar ve birler içermektedir. Bu blokların yeniden yapılandırma değeri tespit edilmiştir. Yeniden yapılandırma, iki değer, k ve l kullanılarak yapılır.

$$k = \bar{r} - \sigma \sqrt{\frac{p}{N-p}} \rightarrow (F)$$

ve

$$l = \bar{r} - \sigma \sqrt{\frac{N-p}{p}} \rightarrow (G)$$

q değerini 1 olarak alan piksel sayısıdır ve σ , formül kullanılarak hesaplanan standart sapma olup, aşağıdaki şekilde hesaplanır:

$$\sigma = \left\{ \frac{1}{m} \sum_{i=1}^m x_i - \bar{x}^2 \right\}^{\frac{1}{2}} \rightarrow (H)$$

4.2.6 Altbant Kodlama

Bu kaynak kodlama tekniğinde, giriş frekans bandı, dijital filtre bankası kullanılarak farklı alt-bantlara ayrılmıştır. Bir giriş $R(n) = (r_1, r_2, \dots, r_n)$ olsun. Bu giriş $Rx(n) = (rx_1, rx_2, \dots, rx_n)$ ve $Ry(n) = (ry_1, ry_2, \dots, ry_n)$ şeklinde ayrılır.

$$rx_i = \frac{r_i + r_{i-1}}{2}, ry_i = \frac{r_i - r_{i-1}}{2} \rightarrow (I)$$

tüm $2 \leq i \leq n$ için $rx_1 = r_1$ ve $ry_1 = 0$ vardır. N frekans bantlarının sayısıdır

Yine r_i , formül kullanılarak toplanabilir:

$$r_i = rx_i + ry_i \rightarrow (J)$$

Bu dijital filtreleme, düşük ve yüksek frekansların ayrılması üzerine kuruludur. Böylece bu filtreler düşük geçiren ve yüksek geçiren filtreler gibi davranır. Daha fazla ayrılma için alt bantların her biri düşük geçiren ve yüksek geçiren filtrelerle uygulanır. Bu sıkıştırma tekniği, psikolojik olarak gereksiz bilgiyi kaldırmak için çalışır ve dolayısıyla yüksek frekanslı verilerin bir kısmı kaldırılır. Ardından, alt bantların her biri ayrı ayrı nicelenir. Son adım olarak, bantların kodlanması gerçekleştirilir. Alt bantların her biri için uygun olan kodlama tekniği diğerleri tarafından kesintiye uğramadan uygulanır.

4.2.7 Vektör Niceleme Sıkıştırma

LBG-VQ algoritması olarak adlandırılan sabit-sabit uzunlukta bir algoritmadır. Temel ilkeleri, büyük veri noktaları kümesini (vektörler) kendilerine en yakın sayıda noktaya bölen gruplara bölmektir. Böylece VQ'ya "Blok Niceleme" veya "Model Eşleme Niceleme" denir. Çok boyutlu bir vektör uzayındaki değerleri, alt boyutun ayrı bir alt uzayından sonlu bir değer kümesine kodlayarak çalışır. Daha düşük bir alan vektörü daha az depolama alanı gerektirir, bu nedenle veri sıkıştırılır. Vektör nicelemenin yoğunluk uydurma özelliğinden dolayı, sıkıştırılmış verinin yoğunluğu ile ters orantılı olan hataları vardır. Dönüşüm genellikle bir Codebook tarafından yapılır. Vektör niceleme, ses ve görüntü sıkıştırma, ses tanıma gibi birçok uygulamada kullanılır.

4.2.8 Çeyrek Ağaç Ayrımı

Görüntü sıkıştırma yöntemi karakterlerinin Çeyrek ağaç ayrımı, basitlik ve hızlı hesaplama yönünden avantajlıdır, ancak sıkıştırma oranı çok yüksek değildir. Fraktal kodlama ve Çeyrek ağaç ayrıştırma yöntemlerinin her ikisi de görüntü ayrıştırmasına dayanır. Fraktal kodlama yeniden yapılandırma kalitesinin ana noktası olan interaktif fonksiyonu çıkartmak için zorlanabilir ve ayrıca büyük hesaplamalara ihtiyaç duyabilir. Çeyrek ağaç ayrıştırma görüntüsü sıkıştırma

yönteminin hesaplanması nispeten basittir ve hızlı hesaplanmaktadır. Dahası, görüntü verisine dayanan imge ile ilgilenir.

4.2.9 JPEG

JPEG standardı, ISO / IEC 10918, günümüzün en geniş yaygın resim sıkıştırma formatıdır. JPEG hareketsiz görüntü sıkıştırma standardı, hemen hemen tüm platformlarda uygulanmış olması nedeniyle oldukça başarılı olmuştur. Bu standart, uygulanması oldukça kolaydır, hesaplama açısından karmaşık değildir. Oldukça yüksek sıkıştırma oranı ile yüksek resim kalitesini seçme veya makul bir düşük resim kalitesi pahasına çok yüksek bir sıkıştırma oranı elde etme esnekliği sunar. Önemli görsel eserler olmaksızın 10: 1 ila 15: 1 sıkıştırma oranlarını alır. Bu standart, 8x8 piksel veri bloklarının ayrık kosinüs dönüşümünün nicelenmiş katsayılarının entropi kodlamasına dayanmaktadır.

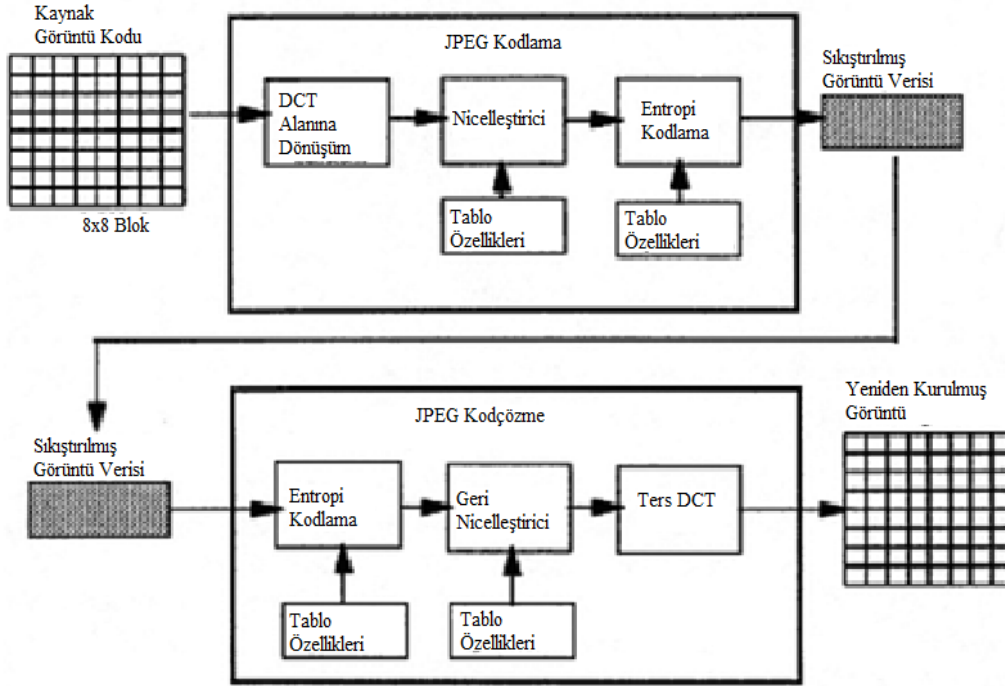
Tekniğin düşük karmaşıklığına bağlı olarak kameralar ve görüntüleyiciler gibi sistemler ucuz yapılabilir. Yapıtlar Şekil 4.11’de görüldüğü gibi “blokaj” gösterir. Bloklama, sıkıştırma oranı çok yüksek olduğunda ortaya çıkar. Normal kullanımda, JPEG sıkıştırılmış bir resim orijinal sıkıştırılmamış resimde hiçbir görsel fark göstermez. JPEG görüntü sıkıştırma, bir dizi gelişmiş teknik içerir. Gerçek görüntü sıkıştırmasını yapan ana şey, Ayrık Kosinüs Dönüşümü (DCT) ve ardından gereksiz bilgileri kaldıran bir nicelemedir (“görünmez” parçalar) (White Paper, 2008).



Şekil 4.10: Soldaki orijinal resim ve sağdaki JPEG ile sıkıştırılmış resim

Şekil 3.10, hem JPEG sıkıştırma hem de kod çözme algoritmalarının blok diyagramını göstermektedir. Tek bir çerçeve 8x8 bloklara bölünür ve her biri bağımsız olarak işlenir. Her blok, DCT katsayılarının 8x8 bloğuyla sonuçlanan

DCT alanına dönüştürülür. Bu katsayılar daha sonra sabitlerle tamsayı bölünmesiyle nicelenir. Her bir DCT katsayısı için niceleyici sabit, minimal görsel eserler üretmek üzere seçilirken, katsayıların temsilsel entropisini azami ölçüde azaltmaktadır. Kuantize edilmiş katsayılar daha sonra sıkıştırılmış bir veri akışına entropi kodlanır. Nicelenmiş katsayıların azaltılmış entropisi, verinin daha yüksek sıkıştırma oranına yansır.



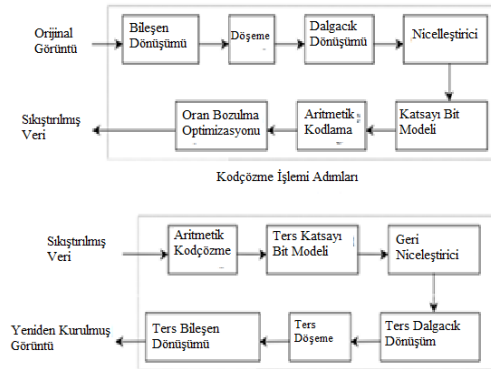
Şekil 4.11. JPEG kodlama ve kod çözme algoritması

4.2.10 JPEG2000

1996'da Joint Photographic Expert Group tekrar bir araya geldiler ve orijinal JPEG sıkıştırma algoritmasının kötüye gidişini düzeltmek amacıyla ve az bit oranlarında daha iyi kalitede görüntüler elde etmek amacıyla tasarlamak için yeni bir algoritmaya ihtiyaç olduğuna karar verdi. JPEG2000 fikri, 1996'da Ortak Fotoğraf Ekspert Grubu toplantılarından biri sırasında ortaya çıkmıştır. JPEG-2000'in amacı "farklı tonlarda" (sürekli ton, metin, karikatür, tıbbi vb.) her türlü hareketsiz görüntü (iki düzeyli, gri tonlamalı, renkli, çok bileşenli) farklı görüntüleme modelleri (istemci/sunucu, gerçek zamanlı aktarım, görüntü arşivi arşivleme, sınırlı tampon ve bant genişliği kaynakları vb.) ve tercihen birleştirilmiş bir sistem için yeni bir görüntü sıkıştırma sistemi geliştirmektir (Lui, 2001).

JPEG2000, JPEG ile aynı isme sahip olmakla birlikte çok farklıdır. Örneğin JPEG “kayıplı” bir format iken JPEG2000 hem kayıpsız hem de kayıplı olarak kabul edilir. JPEG komisyon tarafından alınan kararların tam olarak anlaşılabilmesi için, kayıpsız araçların ne olduğunu anlamının zorunluluğu vardır.

Ayrıca JPEG2000, diğer kayıpsız algoritmaların yapısı üzerine kurulmuştur. 1997’de JPEG WG1 sözleşmesi sırasında, mevcut olan herkese belirli sayıda numune almak ve bunları farklı oranlarda sıkıştırmak için bir meydan okuma yapıldı. Belirtilen oranlar, piksel başına 0.0625 - 2.0 bit aralığında (bpp) idi. Bir sonraki toplantıda, tüm örnekler görüntüleri bastırarak ve daha sonra görüntü kalitesini ve geliştirilen kalite analiz algoritmaları ile karşılaştırılarak analiz edildi. JPEG2000 algoritmasının temelini oluşturan en üst aday, çalıştırılan her test için birinci dereceden olduğu ortaya çıkan dalgacık / kafes kodlu niceleme (WTCQ) olduğu ortaya çıktı. Günü kazanan WTCQ algoritması, JPEG2000 standartları olarak en iyi uygulanacak özelliklerin bulunması için daha ileri testlere tabi tutuldu. WTCQ’nun ve dolayısıyla JPEG2000’nin temeli, ayrık dalgacık dönüşümüdür ve JPEG en iyi bilinen sıkıştırma teknolojilerinin kullanılarak oluşturulan bir kodlama sistemidir (Kodituwakku and Amarasinghe, 2010). Bu ayrık dalgacık dönüşümü, bir Fourier serisidir ve anlaşılması kolay değildir.



Şekil 4.12. JPEG-2000 kodlama ve kod çözme yapısı

JPEG-2000 kodlayıcı ve kod çözücüsünün basitleştirilmiş yapıları Şekil 3.12’de gösterilmektedir. Çok bileşenli bir görüntüye sahip olduğumuzu varsayalım. Encoder’ın temel işlem adımları: bileşen dönüşümü, döşeme, dalgacık dönüşümü, niceleme, katsayı biti modelleme, aritmetik kodlama ve hız bozulması optimizasyonu. Kod çözücünün rolü, hız bozulması optimizasyon adımı haricinde kodlayıcı tarafından gerçekleştirilen adımları tersine çevirmektir.

JPEG2000 Özellikleri

JPEG2000, orijinal algoritmaya göre 1996'da kurulduğu günden beri yaratılmış birçok avantaja sahiptir. Bu avantajlar:

Kaliteye, çözünürlüğe, bileşene veya mekansal yöreye göre aşamalı iletim

Kaybedilen ve kayıpsız sıkıştırma (kusursuz kod çözme doğal olarak tüm ilerleme tipleri ile elde edilebilir)

Bit akışına rastgele (mekansal) erişim

Kaydırma ve yakınlaştırma (yalnızca sıkıştırılmış verilerin bir alt kümesinin sıkıştırmasıyla)

Sıkıştırılmış alan adı işleme (örneğin, kırpmanın döndürülmesi)

Sıkıştırma ile ilgi bölgesi kodlama

Sınırlı bellek uygulaması

4.2.11 Kayıplı ve Kayıpsız Karşılaştırması

Günümüzde kullanılan algoritmanın en yaygın biçimi, bol miktarda kullanımı ve kolay olması nedeniyle “kayıplı” JPEG algoritmasıdır. Kayıpsız bir algoritmayı kullanmanın en büyük avantajı, kullanıcıların bitmap ve tiff gibi sıkıştırılmamış biçimler arasında herhangi bir veri atılmadan biçime ve tekrar dönüştürebilmesidir. Bir “kayıplı” algoritma her sıkıştırma ve kod çözme sırasında kalitesini kaybetmeyi sürdürür. TIF ve JPEG2000 arasındaki bir karşılaştırma Şekil 3.13’de gösterilmektedir.



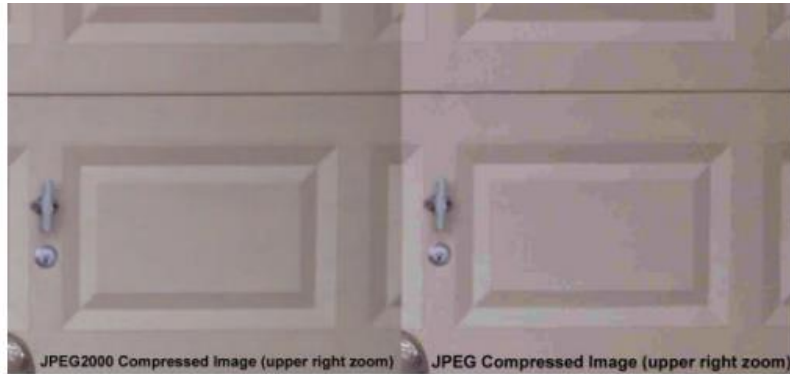
Şekil 4.13. JPEG2000 ve TIFF arasındaki depolama boyutunun karşılaştırılması

Şekil 3.13’de görüldüğü gibi, “kayıplı” modda JPEG2000’ye sıkıştırıldığında dosya boyutu önemli ölçüde azaltılmıştır. Bu noktayı daha da vurgulamak için Şekil 3.14’de JPEG ve JPEG2000’in aynı sıkıştırma oranına yakın dosya boyutları arasında bir karşılaştırma görülebilir.



Şekil 4.14. JPEG ve JPEG2000 dosya boyutu karşılaştırması

Şekil 3.15’de, JPEG2000 formatının orijinal görüntünün daha fazla kalmasını ve JPEG2000’nin neden görüntü sıkıştırma için uygulanacak popüler bir algoritma haline geldiğini görüyoruz. Tüm JPEG2000 görüntüleri Stardust Image Encoder 2003’ün ücretsiz demo sürümünü kullanarak yapıldı. Maalesef bu görüntüler JPEG2000’in “kayıplı” modunu uygulamaktadır. Kayıpsız JPEG2000 modunda sıkıştırılmış tüm görüntülerin boyutu önemli ölçüde azalmaz.



Şekil 4.15. JPEG ve JPEG2000 resim kalitesi karşılaştırması

5. VERİ SIKIŞTIRMADA YENİ BİR TEKNİK

5.1 Yeni Bir Sayı Sistemi Hakkında Genel Bilgi

Bugünün bilgisayarlarında veri 0'lar ve 1'ler şeklinde kodlanmaktadır. 0 ve 1'lerin desenleri sayısal değerleri, alfabedeki karakterleri, sesleri vb. göstermektedir. Bilgisayarlar bütün işlemleri 0 ve 1'ler üzerinde Boolean işlemleri yaparak gerçekleştirmektedir (Hamacher et al., 2002). Bilgisayarlarda 0 ve 1'lerin desenlerinin gösterimini basitleştirmek, onlar üzerinde işlemleri kolaylaştırmak için farklı (sekizlik, onaltılık, vb.) gösterimler kullanılır ve bunun için de uygun sayı sistemlerinden yararlanılır. Bu çalışmada önce bilinen sayı sistemlerine değinilmiş, sonra ise ikilik sayı sistemini temel alan yeni bir sayı sistemi önerilmiştir. Bu sayı sistemi kullanılarak ikilik sistemde verilmiş verilerin tekrar ikilik sistemde kodlanması için yeni bir yöntem önerilmiştir. Önerilen yöntem veri şifreleme ve veri sıkıştırma işlemleri için kullanılabilir (Salomon, 2007).

5.1.1 Sayı Sistemleri

Sayı sistemleri genelde konumsal (pozisyonel) ve konumsal olmayan (pozisyonel olmayan sistemler) olmak üzere iki gruba ayrılır (Knuth, 1998).

5.1.1.1 Konumsal Olmayan Sayı Sistemleri

İlk sayı sistemleri konumsal olmayan sayı sistemleridir: Eski Mısır ve Çin sayı sistemleri konumsal olmayan sayı sistemlerine örnektir. Bu tür sayı sistemlerinde sayılar nesnelerle gösterilmekte ve gruplaştırmaya dayalı olarak ifade edilmektedir. Bu sistemlerin bir diğer özelliği ise sıfır barındırmamalarıdır. Konumsal olmayan sayı sistemlerine en yaygın örnek sıfır ve basamak kavramı olmayan ve günümüzde çeşitli amaçlar için kullanılmakta olan Roma rakamlarıdır.

Roma rakamları: Romalılar sayıları belirtmek için 7 ayrı harfi (I – bir, V – beş, X – on, L – elli, C – 100, D – 500, M - 1000) rakam olarak kullanmışlardır. Daha büyük sayılar ise belirli kurallar kullanılarak toplama ve çıkarma ile elde edilebilmektedir.

5.1.1.2 Konumsal Sayı Sistemleri

Hindu-Arap sistemi konuma dayalı ilk sistemlerdendir. Bu sistemde her sembolün konumu ve şekli önem taşımaktadır. Günümüzde Hindu-Arap sistemini kullanmaktayız. Konumsal sistemlerde genellikle tüm sayı sistemleri aynı ilkeye göre oluşturulmaktadır. Sayı sisteminin tabanı olan p sayısı belirlenir ve her hangi N sayısı sayının $(0, p - 1)$ aralığındaki üslerinin, değişen a_i katsayılarıyla (basamak) çarpımıyla elde edilir:

$$N = a_k p^k + a_{k-1} p^{k-1} + \dots + a_2 p^2 + a_1 p^1 + a_0$$

Sonra ise kısaltılmış olarak katsayıları ile p tabanında ifade edilir.

$$N \equiv (a_k a_{k-1} \dots a_1 a_0)_p$$

Günümüzde kullandığımız sayı sistemi onun kuvvetleri biçiminde ifadelendirilmesine dayandığından (yani $p = 10$) bu sisteme onluk sistem denilmektedir. Bu sistemde sayının basamakları 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 rakamları olabilmektedir. Günümüzdeki elektronik teknolojisi ile kolaylıkla entegre olabildiği için bilgisayarlarda ikilik sayı sistemi (yani $p = 2$) kullanılır. Bu sistemde sadece 0 ve 1 rakamlarından oluşan ikilik sayı sistemini kullanılmaktadır. Yani bilgisayarlar için “evet - hayır” veya “açık - kapalı” ifadeleri geçerlidir. İkilik sistemden başka bilgisayarlarla sekizlik (oktal) ve onaltılık (hexadecimal) sayı sistemleri de yaygın olarak kullanılmaktadır. Sekizlik sistemde taban $p = 8$ ve sayının basamakları 0, 1, 2, 3, 4, 5, 6, 7 rakamları olabilmektedir. Onaltılık sistemde ise taban $p = 16$ ve sayının basamakları 0 ile 15 arasında değerler alabilmektedir. Burada 10, 11, 12, 13, 14, 15 değerlerinin temsil edebilmek için sırasıyla A, B, C, D, E ve F harfleri kullanılmaktadır.

Dengeli sayı sistemleri: İkilik sayı sistemi, sayısal bilgisayarlarda kullanılan en geleneksel ve kolay gerçekleştirilebilen bir sayı sistemi olduğu düşünülmektedir. Fakat üçlü sayı sisteminin en ekonomik sayı sistemi olduğu gösterilmiştir (Brusentsov et al., 2015). Hatta 60’lı yıllarda Moskova Devlet Üniversitesinde üçlü sayı sisteminde çalışan SETUN bilgisayarları üretilmiştir (Brusentsov, 1960). Burada kullanılan üçlü sistemde taban $p = 3$ basamak rakamları olarak ise 0, 1, 2

yerine $(-1, 0, 1)$ kullanılmaktadır (Brusentsov, 1978). Yani burada basamak rakamları simetriktr (Shannon, 1950). Bu yüzden, negatif sayıları göstermek için sayının önünde “-” işaretini kullanmaya gerek kalmamaktadır (Brusentsov, 1965). Başka bir dengeli sayı sistemi ise Negapozisyonlu sayı sistemidir. Bu sistemlerde taban negatif sayı olarak seçilmektedir:

$p = -2$ ise Nega ikilik, $p = -4$ ise Nega dörtlük sistem denilmektedir. Nega ikilik sistemde basamak değerleri 0 ve 1, nega-dörtlük sistemde ise basamaklar $-a_i$, 0, 1, 2, 3 değerlerini alır. Bu sistemlerde de negatif sayıları göstermek için “-” işaretine gerek yoktur. Dengeli sistemlere başka bir örnek ise sanal birim i 'yi içeren sayı sistemleridir. Bu sistemlere örnek olarak, $p = 2i$, $p = a + bi$, $p = r(\cos a + i \sin a)$ sistemlerini gösterebiliriz. $p = 2i$ sistemi için basamak kümesi olarak $A = 0, 1, 2, 3$ alınır. $p = 4i$ sistemi için ise basamak kümesi olarak $A = 0, 1, 2, 3, 5, 6, 7$ alınır.

Karışık sayı sistemleri: Yukarıda ele aldığımız sayı sistemlerinde bir taban belirlenmekte ve verilen sayının her basamağındaki rakam tabanın uygun kuvvetleri ile çarpılarak toplanmaktadır. Karışık sayı sistemlerinde ise kuvvetlerin yerine farklı fonksiyonlarla tanımlanmış değerler kullanılmaktadır. Karışık sistemlerden en yaygını faktoriyel sayı sistemleridir. Faktoriyel sayı sistemlerinde belirlenmiş bir sayıyı genel olarak aşağıdaki şekilde yazabiliriz:

$$(a_n a_{n-1} a_{n-2} \dots a_2 a_1) = a_n.n! + a_{n-1}.(n-1)! + a_{n-2}.(n-2)! + \dots + a_2.2! + a_1.1!$$

Karışık sayı sistemlerine başka bir örnek ise Fibonacci sayılarına dayanan sayı sistemidir. Günümüzde yukarıda söylediklerimizden başka kullanılan diğer bir sayı sistemi de “kalanlar sınıfı” sayı sistemidir. Bu sayı sistemi modüler aritmetiğe dayanmaktadır.

5.1.2 Yeni Sayı Sistemi

Aşağıda önerilen N^k sistemi ($k = 1, 2, 3, \dots$) taban olarak konumsal sayı sistemini (ikilik sayı sistemi) kullanılmasına rağmen, genel olarak konumsal olmayan sayı sistemidir. Bu sistemde de sıfır değeri yoktur. Tabanda ise farklı basamaklı ikilik sayılar kullanılabilir ve burada “k” tabanı göstermektedir. İkilik sistemde bir basamaklı sayılar iki tanedir: 0 ve 1. Önerilen N^1 sisteminde bir basamaklı yalnız bir değer vardır: 1. Burada “0”, “1+” anlamına gelmektedir. Örneğin:

$$(1)_N^1 = (1)_{10}$$

$$(0)_N^1 = (1+)_{10}$$

$$(01)_N^1 = 1 + 1 = 2_{10}$$

$$(00)_N^1 = 1 + 1 + 0 = (2+)_{10}$$

$$(001)_N^1 = 1 + 1 + 1 = 3_{10}$$

$$(000)_N^1 = 1 + 1 + 1 + 0 = (3+)_{10}$$

$$(0001)_N^1 = 1 + 1 + 1 + 1 = 4_{10}$$

$$(0000)_N^1 = 1 + 1 + 1 + 1 + 0 = (4+)_{10}$$

Bu kural ile bir basamaklı tabanla sonsuz çoklukta sayı elde edilebilmektedir.
İkilik sistemde iki basamak ile dört sayı gösterilebilir:

$$(00)_2 = 0_{10}$$

$$(01)_2 = 1_{10}$$

$$(10)_2 = 2_{10}$$

$$(11)_2 = 3_{10}$$

Bu sayılar kullanılarak N^2 sisteminde sonsuza kadar sayılar gösterilebilir:

$$(01)_N^2 = 1_{10}$$

$$(10)_N^2 = 2_{10}$$

$$(11)_N^2 = 3_{10}$$

$$(00)_N^2 = (3+)_{10}$$

$$(0001)_N^2 = 3 + 1 = 4_{10}$$

$$(0010)_N^2 = 3 + 2 = 5_{10}$$

$$(0011)_N^2 = 3 + 3 = 6_{10}$$

$$(0000)_N^2 = 3 + 3 = (6+)_{10}$$

$$(0001)_N^2 = 3 + 1 = 4_{10}$$

$$(000001)_N^2 = 6 + 1 = 7_{10}$$

$$(000010)_N^2 = 6 + 2 = 8_{10}$$

$$(000011)_N^2 = 6 + 3 = 9_{10}$$

$$(000000)_N^2 = 6 + 3 = (9+)_{10}$$

İkilik sistemde 3 basamakla 8 sayı gösterilebilir:

$$(000)_2 = 0_{10}$$

$$(001)_2 = 1_{10}$$

$$(010)_2 = 2_{10}$$

$$(011)_2 = 3_{10}$$

$$(100)_2 = 4_{10}$$

$$(101)_2 = 5_{10}$$

$$(110)_2 = 6_{10}$$

$$(111)_2 = 7_{10}$$

Aynı şekilde bu sayıları kullanarak N^3 sisteminde de sonsuza kadar sayılar gösterilebilir:

$$(001)_N^3 = 1_{10}$$

$$(010)_N^3 = 2_{10}$$

.....

$$(111)_N^3 = 7_{10}$$

$$(000)_N^3 = (7+)_{10}$$

$$(000001)_N^3 = 7 + 1 = 8_{10}$$

$$(000111)_N^3 = 7 + 7 = 14_{10}$$

$$(000000)_N^3 = 7 + 7+ = (14+)_{10}$$

$$(00000000)_N^3 = 14 + 1 = 15_{10}$$

.....

$$(000000111)_N^3 = 14 + 7 = 21_{10}$$

Böylece N^k sistemindeki sayılar için genel formül aşağıdaki şekilde yazılabilir:

$$\underbrace{\left(\underbrace{0 \dots 0}_k \underbrace{0 \dots 0}_k \dots \underbrace{0 \dots 0}_k \underbrace{a_{k-1} \dots a_0}_k \right)}_m^k = (m-1) \cdot (2^k - 1) +$$

$$+ (a_{k-1} 2^{k-1} + a_{k-2} 2^{k-2} + \dots + a_1 \cdot 2 + a_0)$$

Önerilen sayı sistemi aşağıdaki kodlama yönteminde kullanılmak için tasarlanmıştır.

5.1.3 Yeni Kodlama Yöntemi

Önerilen kodlama yöntemi “0” ve “1” lerden oluşan dizinin yukarıda anlatılan sayı sistemini kullanarak tekrar kodlanmasına dayanmaktadır. Bu yönteme göre

tekrar kodlama için ilk önce verilmiş dizinin ilk elemanı yazılır ve sonra bu elemanın ardışıklık sayısı yazılır ve bu kural ile dizinin sonuna kadar devam ettirilir. Örneğin, varsayalım ki, “0” ve “1” lerden oluşan aşağıdaki dizi verilmiştir ve 2 basamaklı N^2 sistemini kullanalım.

000 111 00 1 0000 11 00000 111111
 3 3 2 1 4 2 5 6

Bu dizini önerilen yöntemle tekrar kodlayalım. İlk eleman “0” olduğu için yeni kodda ilk önce “0” ve sonra burada ardışık “0”ların sayısı 3 olduğu için 11 yazılacak ve böyle devam ederek bütün dizini kodlayalım.

0 11 11 10 01 0001 10 0010 0011
 3 3 2 1 4 2 5 6

5.1.4 Yeni Kodlama Yönteminin Uygulama Alanları

Bu yöntemi kullanarak ardışık “0” ve “1” olan dizileri sıkıştırabiliriz. N^1 ile hiçbir sıkıştırma olmuyor, dizinin eleman sayısı ilk elemanı göstermek için kullanılan elemana göre 1 artar.

N^2 sistemi kullanıldığında ise 1 elemanlı tekrarlar için 1 artım olur. 2 elemanlı tekrarlar için eleman sayısı değişmez. 3 elemanlı tekrarlar için eleman sayısında 1 azalma olur. 4 elemanlı tekrar için eleman sayısı değişmez. 5 elemanlı tekrarlar için eleman sayısında 1, 6 elemanlı tekrarlar için ise 2 azalma olur. 7 elemanlı tekrarlar için eleman sayısında 1, 8 elemanlı tekrarlar için 2, 9 elemanlı tekrarlar için ise 3 azalma olur.

$10 \leq n \leq 12$ elemanlı tekrarlar için eleman sayısında $(n-8)$ azalma olacak. N^2 için azalmaları genel olarak

$$n - 2 \cdot \left\lceil \frac{n}{3} \right\rceil$$

formülü ile belirleyebiliriz. Herhangi bir k için ise N^k yeni oluşturulacak gösterimin uzunluğu aşağıdaki şekilde gösterilebilir:

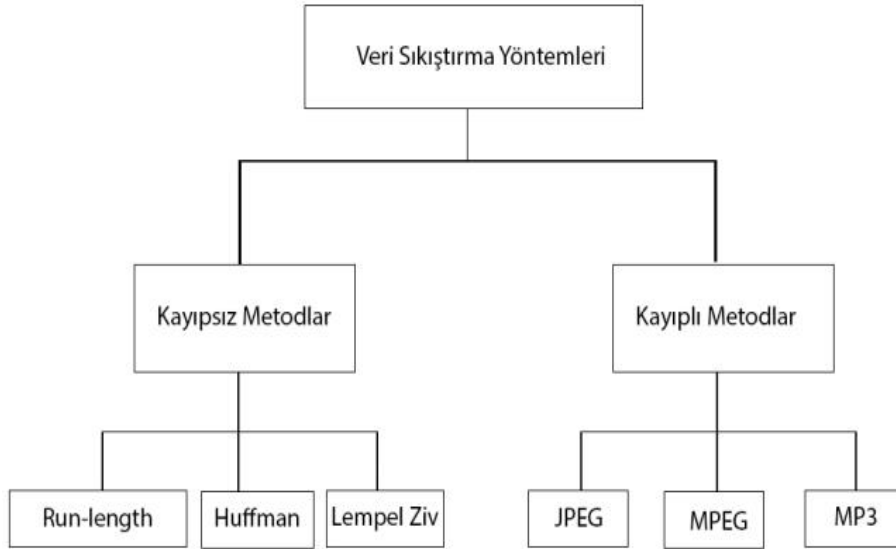
$$k \cdot \left\lceil \frac{n}{(2k-1)} \right\rceil$$

5.2 İkili Görüntülerin Kayıpsız Sıkıştırılması İçin Yeni Bir Yöntem

Görüntü sıkıştırma teknikleri görüntülerin saklanması için gerekli depolama alanının azaltılmasında ve ağlar üzerindeki transferlerinin hızlandırılması için yaygın olarak kullanılmaktadır. Görüntü sıkıştırma teknikleri genel olarak Şekil 4.3’de gösterildiği gibi kayıplı ve kayıpsız olmak üzere iki sınıfa ayrılır. Kayıpsız sıkıştırma görüntünün eski haline dönüştürülmesinden sonra orijinal görüntü ile tamamıyla aynı olması anlamına gelir. Yani sıkıştırma sonrası görüntünün boyutunda azalma olmakla birlikte görüntünün tüm özellikleri korunur ve kalitesinde herhangi bir kayıp gerçekleşmez. Kayıplı sıkıştırmada ise görüntünün sıkıştırılmasından sonra eski haline dönüştürülmesi görüntünün bazı özelliklerinde değişiklik göstermekle birlikte kalitesinde de düşme görülebilir. Kayıpsız sıkıştırma görüntü kalitesini koruduğundan yüksek bir sıkıştırma oranı elde edebilmek kayıplı sıkıştırma yöntemlerine göre daha zordur. Sıkıştırma oranı herhangi bir sıkıştırma algoritmasının performansını gösteren parametre olarak kullanılır (Sayood, 2012). Bu oran aşağıdaki gibi hesaplanır:

$$\text{Sıkıştırma Oranı} = \frac{\text{Sıkıştırılmış Verinin Boyutu}}{\text{Gerçek Verinin Boyutu}}$$

İkili görüntülerde sıkıştırma işlemi görüntüyü oluşturan bitlerin sayısının azaltılmasını amaçlar. İkili görüntülerin sıkıştırılması önemlidir çünkü günümüzde özellikle görüntü olarak saklanan ve işlenen metin belgesi, parmak izi ve haritalar gibi geniş bir uygulama alanı vardır. İkili görüntülerin yer aldığı veri tabanlarında daha fazla alan kazanmak ve veri transferlerinin hızlanması için ikili görüntülerin etkin bir şekilde sıkıştırılması önemlidir. İkili görüntülerin sıkıştırılması için geliştirilen yöntemler, genel olarak görüntülerin geometrik yapıları gibi karakteristik özelliklerini kullanırlar. Bunun yanı, sıra ikili görüntüyü oluşturan verilerin daha az sayıda bit ile ifade edilebilmesini sağlayan kodlama yöntemleri geliştirilmiştir.



Şekil 5.1. Sıkıştırma metodlarının ağaç görünümü (Brar and Singh'den, 2013)

Literatürde ikili görüntülerin kayıpsız sıkıştırılmasında bir çok kodlama yöntemi yer almaktadır. Başlıca kodlama yöntemleri koşan-uzunluk kodlama (Jayant and Noll, 1984), Huffman kodlama, Aritmetik kodlama, Çeyrek ağaç kodlama, zincir kod gösterimi ve Lempel-Ziv-Welch kodlamalarıdır (Jindal et al., 2013). Bu yöntemler genellikle farklı sıkıştırma algoritmaları ile birlikte kullanılarak daha iyi sıkıştırma oranları elde edilmiştir. Vang ve arkadaşları sıkıştırma aşamasında çeyrek ağaç kodlama ve koşan uzunluk kodlamanın birlikte kullanıldığı QSBBIC isimli bir ikili resim sıkıştırma yöntemi önermiştir (Wang et al., 2005). Talu ve Türkoğlu ikili görüntüleri sıkıştırmada kenar izleme algoritması ile birlikte LZW ve Huffman kodlama yöntemlerinin birlikte kullanıldığı hibrit bir yöntem önermiştir (Talu and Türkoğlu, 2011). Zahir ve Naqvi ikili resimdeki siyah bölgeleri en az sayıda dikdörtgensel bölgelere ayırıp bölgelerin koordinatlarını kodlamaya dayanan sıkıştırma yöntemi önermiştir (Zahir and Naqvi, 2005).

İkili resimlerin sıkıştırılması konusunda yapılan çalışmalara bakıldığında görüntülerin karakteristik özelliklerini dikkate almadan direk olarak sıkıştırma yapan yöntemler olduğu gibi görüntülerin karakteristik özelliklerini kullanarak sıkıştırma yapan algoritmalarda mevcuttur. Tüm bu çalışmalardaki ortak özellik verilerin kodlanması için literatürde var olan kodlama yöntemlerinin kullanılması veya yeni kodlama yöntemlerinin geliştirilmesidir. İkili görüntülerin sıkıştırılması için geliştirilen yöntemlerden elde edilen sonuçların en çok karşılaştırıldığı kodlama yöntemi koşan-uzunluk yöntemidir.

Bu çalışmada Nuriyev ve arkadaşlarının geliştirdiği kodlama yöntemi (Nuriyev et al., 2016) kullanılarak ikili görüntülerin kodlanmasında yeni bir yöntem geliştirilmiş ve bu yöntemden elde edilen sonuçlar diğer kodlama yöntemleri ile karşılaştırılmıştır.

5.1.5 Önerilen Yöntem

Yöntem ikili resimlerin sıkıştırılmasında 0 ve 1 lerin yoğunluğundan yararlanan kodlama, sıkıştırma ve orjinal resme geri dönüştürme olmak üzere üç aşamadan oluşmaktadır. Önerilen yöntemin kodlama aşamasında NNS (Nuriyev Number System) kullanılmıştır.

5.1.5.1 Kodlama Aşaması

Günümüzde kullanılan sayı sistemi sayıları 10'un kuvvetleri şeklinde ifadelendirilmesine dayanan onluk sayı sistemidir. Dijital ortamda kullanılan sayı sistemleri ise ikilik (binary), sekizlik (oktal) ve onaltılık (hexadecimal) sayı sistemleridir. İkili görüntülerin ifade edilmesinde kullanılan sayı sistemi ikilik gösterimde olup siyah bölgeler “1” ile beyaz bölgeler ise “0” ile ifade edilir. İkilik gösterimde sayıları ifade ederken kullanılan basamak sayısı dijital ortamda bit sayısı anlamına gelmektedir. Örneğin bu gösterimde iki bit ile dört sayı gösterilebilir:

$$(00)_2 = 0_{10}$$

$$(01)_2 = 1_{10}$$

$$(10)_2 = 2_{10}$$

$$(11)_2 = 3_{10}$$

NSS’de kullanılan sayılar “0” ve “1” olmasına rağmen bu sistemde kullanılan taban ne olursa olsun sonsuz tane sayı gösterilebilir. $N^k(k=1, 2, 3, \dots)$ sistemi olarak ifade edilen bu sistemde “0” değeri yoktur ve burada “k” değeri tabanı ifade etmektedir. Örneğin N^1 sisteminde bir basamaklı yalnız bir değer vardır: 1.

Burada “0”, “1+” anlamına gelmektedir.

Örneğin:

$$(1)_N^1 = (1)_{10}$$

$$(0)_N^1 = (1+)_{10}$$

$$(01)_N^1 = (1+1)_{10} = (2)_{10}$$

$$(00)_N^1 = (1+1+)_{10} = (2+)_{10}$$

$$(001)_N^1 = (1+1+1)_{10} = (3)_{10}$$

Bu kural ile bir basamaklı tabanla sonsuzluğa kadar devam edilebilir. Aynı şekilde bu sayılar N^3 sisteminde de sonsuza kadar gösterilebilir :

$$(001)_N^3 = (1)_{10}$$

$$(010)_N^3 = (2)_{10}$$

.....

$$(111)_N^3 = (7)_{10}$$

$$(000)_N^3 = (7+)_{10}$$

$$(000001)_N^3 = (7+1)_{10} = (8)_{10}$$

.....

$$(000000111)_N^3 = (7+7+7)_{10} = (21)_{10}$$

N^k sistemindeki sayılar için onluk tabandaki karşılığını veren formül aşağıdaki gibidir:

$$\underbrace{\underbrace{(0 \dots 0)_k}_{m/k} \underbrace{(0 \dots 0)_k}_{m/k} \dots \underbrace{(0 \dots 0)_k}_{m/k} \underbrace{(a_{k-1} \dots a_0)_k}_{m/k}}_m^N = \left(\frac{m}{k} - 1\right) \cdot (2^k - 1) +$$

$$+ (a_{k-1} 2^{k-1} + a_{k-2} 2^{k-2} + \dots + a_1 \cdot 2 + a_0) \quad (2)$$

burada m : tüm bitlerin sayısı

Örneklerden de görüldüğü gibi kullanılan bu sayı sisteminde seçilen herhangi bir tabanda sonsuz tane sayı ifade edilebilir. Bu durum koşan-uzunluk kodlamada karakterlerin frekans sayılarını ifade etmek için kullanılan ikilik gösterimdeki bit sayısı belirleme sorununu engelleyebilecek bir faktördür. Çünkü bir veride bulunan sayıları kodlarken ikilik gösterimde bu sayıların en büyüğünü ifade edebilecek bit sayısı seçilmesi gerekir. Örneğin “15” sayısını ikilik gösterimde ifade edebilmek için en az “4” bit kullanılması gerekir. Oysa bu sistemde sayı ne kadar büyük olursa olsun seçilen herhangi bir tabanda rahatlıkla tüm sayılar ifade edilebilir. Bu yüzden NSS nin avantajlarından yararlanabilmek için verilerin kodlanma aşamasında NCS (Nuriyev Coding System) kullanılmıştır. Özellikle ikili görüntülerde “0” ve “1” lerin ardışık olarak yoğun bir şekilde bulunması yöntemin avantajını daha da ön plana çıkarmaktadır.

NNC “0” ve “1” lerden oluşan bir dizinin NNS’ye göre tekrar kodlanmasına dayanır. Bu yöntemle göre tekrar kodlama için ilk önce verilmiş dizinin ilk elemanı yazılır ve sonra bu elemanın ardışıklık sayısı yazılır ve bu kural ile dizinin sonuna kadar devam ettirilir.

Örneğin, varsayalım ki, “0” ve “1” lerden oluşan aşağıdaki dizi verilmiş olsun ve 2 basamaklı N^2 sistemini kullanalım.

$$\underbrace{000}_3 \underbrace{111}_3 \underbrace{00}_2 \underbrace{1}_1 \underbrace{0000}_4 \underbrace{11}_2 \underbrace{00000}_5 \underbrace{11111}_6$$

Bu dizi önerilen yöntemle tekrar kodlanırken ilk eleman “0” olduğu için (NNC)’de ilk önce “0” ve sonra burada ardışık “0” ların sayısı 3 olduğu için 11 yazılır ve böyle devam ederek bütün dizi kodlanır. Verilen örneğe göre dizinin kodlanmış şekli aşağıdaki gibi olacaktır.

0 11 11 10 01 0001 10 0010 0011
 3 3 2 1 4 2 5 6

Kodlama aşamasının sözde kodu aşağıdaki gibidir:

```
Function Kodla(Sayi, taban)
  m= ⌊ Sayi / k ⌋
  for i= 1 to m
    for j= 1 to k
      Kod=Kod+ '0'
    End(for)
  End(for)
  Kod=Kod+SayiSistemi(Sayi%taban)
  Return Kod
End(Function)
```

5.1.5.2 Sıkıştırma Aşaması

Önerilen yöntem ikili görüntülerin sıkıştırılmasında kullanılan koşan-uzunluk kodlama yöntemindeki sayı sisteminin değiştirilmiş halidir. Koşan-uzunluk kodlama, kayıpsız bir kodlama tekniğidir ve ardışık özdeş sembollere sahip görüntüler için iyi bir performans sağlar. Koşan-uzunluk yöntemi Faks makineleri için standart bir kodlama yöntemi olmuştur. Bu yöntem aynı değere sahip ardışık bit değerlerinin belli bir doğrultuda sıralanmış olmasından yararlanır. Bu sıranın uzun olması başarılı bit azaltma işlemi sağlar (Jayant and Noll, 1984).

Bu çalışmada koşan-uzunluk yönteminde olduğu gibi ardışık bitlerin frekans değerlerini onluk sistemde ifade edip daha sonra bu değerlerin NCS ile kodlanması yapılmıştır. Yeni oluşturulan bu yönteme NDC (Nuriyev Data Compression) ismi verilmiştir. Buradaki k değeri frekans değerlerinin ortalama değerinin ikilik gösterimde ifade edebilecek minimum bit sayısı olarak seçilir.

Eğer $k=1$ ise sıkıştırma işleminde herhangi bir kazanç olmaz. $k=2$ seçildiğinde 1 elemanlı tekrarlar için fazladan bir bit kullanılır, 2 elemanlı tekrarlar için değişiklik olmaz. 3 elemanlı tekrarlar için 1 bit kazanç olur. 4 elemanlı tekrarlar için bit sayısı değişmez. 5 elemanlı tekrarlar için 1 bit kazanç, 6 elemanlı tekrarlar için ise 2 bit kazanç olur. 7 elemanlı tekrarlar için 1 bit kazanç, 8 elemanlı tekrarlar için 2, 9 elemanlı tekrarlar için 3 bit kazanç olur.

$10 \leq n \leq 12$ elemanlı tekrarlar için $(n-8)$ bit kazanç olur. Genel olarak $k=2$ için kazanç miktarı

$$n - 2\lceil n/3 \rceil$$

formülü ile belirlenir. Herhangi bir k değeri için n elemanlı tekrarda kazanç miktarı

$$n - k\lceil n/(2^k - 1) \rceil$$

formülü ile belirlenir.

Sıkıştırma aşamasının sözde kodu aşağıdaki gibidir.

```

Function GörüntüSıkıştırma(Image,k)
    M[,]=Read(Image)
    sakla= M[ilk]
    Kodlama = Kodlama + sakla
    sayi =0
    While (M matrisinin son hücresi değilse)
        hücre = M [sıradaki]
        If (hücre = sakla)
            sayi = sayi +1
        Else
            sakla = hücre
            Kodlama=Kodlama+Kodla(sayi,taban)
            sayi =0
        End(if)
    End(While)
End(Function)

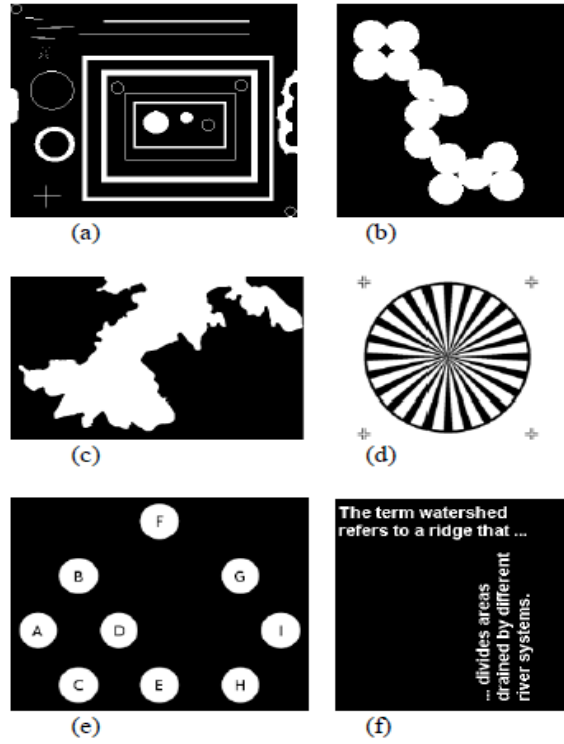
```

5.1.5.3 Dönüştürücü Aşaması

Kodlanmış verinin geri dönüştürülmesi aşamasında öncelikle kodun ilk karakteri belirlenir. İlk karakterden sonra gelen k tabanında ifade edilmiş sayılar onluk tabanındaki karşılığına dönüştürülür. Dönüştürme işlemi tamamlandıktan sonra her frekans değeri kadar “0” veya “1” karakteri yazılır. Burada hangi karakterin yazılacağı ilk karakterden anlaşılmaktadır. Örneğin ilk karakter “0” ise onun yanında gelen frekans değeri kaç defa “0” karakterinin yazılacağını belirler. Daha sonra gelecek olan değer “1” in kaç defa yazılacağını gösterir ve sıralama kaldığı yerden devam eder.

5.3. Hesaplama Denemeleri

İkili görüntülerin kayıpsız sıkıştırılmasında geliştirilen metot ve koşan-uzunluk sıkıştırma yöntemi ile üretilen sonuçlar rastgele alınan Şekil 4.4’deki altı görüntü üzerinde karşılaştırılmıştır. Karşılaştırma sonuçları Tablo 4.1’de gösterilmiştir. Seçilen görüntüler farklı boyutlara ve geometrik şekillere sahiptir. “Dart” resmi gibi çok parçalı resimlerde, iki yönteminde başarısı düşük çıkarken, sürekliliği barındıran şekiller içeren resimlerde ikin yöntemin de başarısı yüksek çıkmıştır. Seçilen görüntülerden elde edilen karşılaştırmalı sonuçlara göre “NDC” nin sıkıştırma oranları koşan-uzunluk metoduna göre oldukça iyi bir üstünlük sağlamıştır.



Şekil 5.2. Test görüntüler (a) “Baloncuklar”; (b) “Daireler”; (c) “Uydu görüntüsü”; (d) “Dart”; (e) “Harfler”; (f) “Metin”

	Boyut	NDC		Koşan-uzunluk	
		Bit Sayısı	Sıkıştırma Oran	Bit Sayısı	Sıkıştırma Oran
Baloncuklar	329*272	32005	0,352	32451	0,36
Daireler	256*256	7414	0,112	8893	0,128
Uydu görüntüsü	1305*802	54955	0,048	72855	0,069
Dart	256*256	90570	1,36	117693	1,795
Harfler	252*200	25350	0,496	29484	0,585
Metin	256*256	18564	0,283	25190	0,384

Tablo 5.3. Karşılaştırma sonuçları (Sadik et al.’den, 2017)

6. SONUÇ

Bu tez çalışmasında veri sıkıştırma teknikleri üzere araştırma yapılmıştır. Tezin 4. Bölümünde yeni bir sıkıştırma tekniği ortaya konulmuş ve koşan uzunluklu sıkıştırmaya göre daha iyi verim elde edildiği görülmüştür. Bu teknikte kullanılan ikili görüntüleri daha verimli sıkıştırabilen yeni bir sayı sistemi ortaya konulmuştur.

Bu çalışmada önerilen yeni sayı sistemi kodlama ve şifrelemede de kullanılabilir. Geliştirilen yeni kodlama sisteminin ise özellikle ardışık tekrarların olduğu sistemlerde yüksek sıkıştırma sağlayabileceği gösterilmiştir. Ardışık sıfırların çok olduğu matrislere örnek olarak grafların matris şeklinde verilmesini gösterebiliriz. Bu sıkıştırma yönteminin bir avantajı da onun tekrar-tekrar aynı dizinin sıkıştırılmış versiyonları üzerinde kullanılabilmesidir.

Bu çalışmada, ikili görüntülerin kayıpsız sıkıştırılması için yeni bir kodlama yöntemi önerilmiştir. Bu alanda en yaygın olarak bilinen koşan-uzunluk kodlama yöntemi ile yeni geliştirilen yöntem karşılaştırılmıştır. Elde edilen karşılaştırmalı sonuçlar yeni yöntemin ikili görüntülerin sıkıştırılmasında başarılı olduğunu göstermiştir. Özellikle uzun ardışık özdeş bitlere sahip görüntülerde yeni metodun etkinliğinin koşan-uzunluk metoduna göre daha iyi olduğu görülmüştür.

KAYNAKLAR DİZİNİ

Arora, K., and Shukla, M., 2014, A comprehensive review of image compression techniques, *International Journal of Computer Science and Information Technologies*, 5(2), 1169-1172p.

Blelloch, G. E., 2001, Introduction to data compression, *Computer Science Department*, Carnegie Mellon University.

Brar, R. S., and Singh, B., March 2013, A survey on different compression techniques and bit reduction algorithm for compression of text data, *International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE)*, 3(2).

Brusentsov, N. P., 1960, Computing machine “Setun” of Moscow State University, *New developments on computer technology*, Kiev, 226– 234p.

Brusentsov, N. P., 1965, A ternary arithmetic machine Moscow State University, *Vesnik*, 2, 39 – 48p.

Brusentsov, N. P., 1978, Threshold realization of threevalued logic on electromagnetic elements, *Computers and problems of cybernetics*, 15, Moscow State University, 3 – 8p.

Brusentsov, N. P., Maslov, S.P., Ramil Alvares, J., and Zhogolev, E.A., January 19, 2015, Development of ternary computers of Moscow State University Russian Virtual Computer Museum, *Retrieved*.

Choudhary, S. M., Patel, A. S., and Parmar, S. J., 2015, Study of LZ77 and LZ78 Data Compression Techniques, *Certif. Int. J. Eng. Sci. Innov. Technol*, 4(3), 45-49p.

Diri, B., 1999, Türkçe’nin biçimbilim yapısına dayalı bir metin sıkıştırma sistemi”, *Fen Bilimleri Enstitüsü Yıldız Teknik Üniversitesi*, İstanbul.

Goemans, M., “Lempel - Ziv Codes”, <http://math.mit.edu/~goemans/18310S15/lempel-ziv-notes.pdf> (2015), (Erişim tarihi: 5 Ocak 2018)

Gomathi, K., Lotus, R., 2014, Digital image compression techniques, *Centre for Information Technology and Engineering*, Manonmaniam Sundaranar University, India.

Hamacher, V.C., Vranesic, Z.G., and Zaky, S.G., 2002, Computer Organization, 5th ed., New York: McGraw-Hill.

Howard, P. G., and Vitter, J. S., 1992, Practical implementations of arithmetic coding, *In Image and text compression*, Springer, Boston, MA, 85-112p.

Jayant, N. S., and Noll, P., 1984, Digital coding of waveforms. Principles and applications to speech and video, *Prentice-Hall*, Englewood Cliffs, N.J.,.

Jindal, V., Verma, A. K., and Bawa, S., 2013, Impact of compression algorithms on data transmission, *International Journal on Advanced Computer Theory and Engineering*, 2(2), 2319 – 2526p.

KAYNAKLAR DİZİNİ (devam)

Knuth, D.E., 1998, The art of computer programming, *Seminumerical algorithms*, vol. 2, 3rd ed., Boston, Addison – Wesley.

Kodituwakku, S. R., and Amarasinghe, U. S., 2010, Comparison of lossless data compression algorithms for text data, *Indian Journal of Computer Science and Engineering*, 1(4), 416-425p.

Krajci, S., Liu, C. F., Mikeš, L., and Moser, S. M., 2015, June, Performance analysis of Fano coding, *In Information Theory (ISIT)*, 2015 IEEE International Symposium on, 1746-1750p.

Lui, C., 2001, A study of the jpeg-2000 image compression standard, 9-14p.

Mesut, A., 2006, Veri Sıkıştırma Yeni Yöntemler, *Trakya Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Ana Bilim Dalı*, Trakya.

Nuriyev, U., Nuriyeva, F., and Sadık, T., 2016, On a new number system, *Proceeding of the International Conference on Computer Science and Engineering*, (UBMK 2016), Namık Kemal University, Tekirdağ, Turkey, 825-827p.

Oswal, S., Singh, A., and Kumari, K., 2016, Deflate compression algorithm, Department of Information Technology, *KJ'S Trinity College Of Engineering and Research*, Pune, India.

Princeton University,

<http://www.cs.princeton.edu/courses/archive/spr03/cs226/assignments/burrows.html> (Erişim tarihi: 10 Ocak 2018)

Ranjeet, K., Reddy, B., 2005, Image compression : an overview, *PDPM Indian Institute of Information Technology Design and Manufacturing Jabalpur*, India.

Sadik, T., Tanir, D., and Nuriyeva, F., 2017, A new method for lossless compression of binary images, *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, Malatya, Turkey, 1-4p. doi:10.1109/IDAP.2017.8090198
<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8090198&isnumber=8090153>

Salomon, D., 2007, Data compression: The complete reference, 4th ed. New York: Springer.

Sayit, M., 2005, İnternet üzerinde hareketli görüntü nakli için veri sıkıştırma ve uyarlama, *Ege Üniversitesi Fen Bilimleri Enstitüsü Uluslararası Bilgisayar Anabilim Dalı*, Bornova.

Sayood, K., 2005, Introduction to data compression, Third Edition, Morgan Kaufmann Publishers.

Sayood, K., 2012, Introduction to Data Compression, Newnes.

Schiller, D., 2012, The Burrows-Wheeler algorithm, *Theoretical Computer Science Department*, RWTH Aachen University, Germany. http://tcs.rwth-aachen.de/lehre/Komprimierung/SS2012/a_usarbeitungen/Burrows-Wheeler.pdf.

KAYNAKLAR DİZİNİ (devam)

Shanmugasundaram, S., Lourdasamy, R., 2011, A comparative study of text compression algorithms, *International Journal of Wisdom Based Computing*.

Shannon, C. E., 1950, A symmetrical notation for numbers, *The American Mathematical Monthly*, 57(2), 90-93p.

Universitetet i Oslo, <https://www.uio.no/studier/emner/matnat/ifi/nedlagte-emner/INF5080/v05/undervisningsmateriale/mkt14e-jpegls.pdf> (Erişim Tarihi: 10.12.2018)

Talu, M. F., and Türkoğlu, İ., 2011, Hybrid lossless compression method for binary images, *IU-Journal of Electrical & Electronics Engineering*, 11(2), 1399-1405p.

Zahir, S., and Naqvi, M., May 2005, A new rectangular partitioning based lossless binary image compression scheme, *In Electrical and Computer Engineering*, Canadian Conference IEEE, 281-285p.

Zeeh, C., 2003, The Lempel Ziv algorithm, *Seminar Famous Algorithms*.

Wang, C. L., Wu, S. C., Chan, Y. K., and Chang, R. F., 2005, Quadtree and statistical model-based lossless binary image compression method, *The Imaging Science Journal*, 53(2), 95-103p.

Weinberger, M., Seroussi, G., and Sapiro, G., 1999, From LOCO-I to the JPEG-LS Standard, *Computer Systems Laboratory HP Laboratories Palo Alto*.

White Paper, 2008, An explanation of video compression techniques, http://www.axis.com/corporate/corp/tech_papers.htm.

ÖZGEÇMİŞ

1986 yılında Azerbaycan'ın Cebrayıl şehrinde doğdum. Anne ve babamın işleri sebebiyle okul hayatıma Azerbaycan'ın başkenti Bakü'de başladım. İlköğretim ve Lise hayatımı tamamladıktan sonra 2007 yılında Azerbaycan Kooperasiya Üniversitesinin Uluslararası Ekonomi ve Gümrük bölümünden mezun oldum. Aynı yıl Azerbaycan Ekonomi Üniversitesinde Uluslararası Ekonomi bölümünde yüksek lisans yapmaya başladım ve 2009 yılında mezun oldum. Evliliğim sebebiyle 2011 yılında İzmir şehrine taşındım. 2013 yılında Ege Üniversitesi Matematik Bölümü Bilgisayar Bilimleri bölümünde doktora eğitimine başladım ve halen doktora eğitimim devam etmektedir. İzmir'de yazılım alanında NETSİS ve AVANDEL firmalarında çalıştım. 2014 yılında iş değişikliği sebebiyle İstanbul'a taşındım. BORUSAN'a bağlı olan BENİM FİLOM MAKİNA A.Ş.'de BİLGİ İŞLEMLERİ YÖNETİCİSİ olarak çalışmaktayım.

SERTİFİKALAR

- Salesforce Service Cloud
- Salesforce ADM-201
- Salesforce DEV-401
- MCPD-Windows Developer
- MCTS-.NET 3.5, Windows Presentation Foundation

YABANCI DİLLER

- İngilizce (TOEFL IBT 100/120)
- Rusça (İleri Seviyede)
- Azerbaycanca (Çok ileri seviyyede)

TEKNİK YETERLİLİK

- **Salesforce** administration and apex/VisualForce/SOQL development
- Mobile application development (Android, IOS, Windows Phone, Phonegap)
- **C#.NET** (rich internet applications, smart client applications)
- Databases Oracle,(MSSQL)**SQL Server 2008**,PLSQL,TSQL
- **WPF**, Silverlight, Expression Blend
- Workflow Foundation
- ASP.NET **jQuery**, BackboneJs, scriptaculous
- **HTML**,CSS,XML,XSLT,**XAML**
- C++
- Delphi
- Java
- Arduino embedded development
- Machine Learning

PROJELER

BENİM FİLOM MAKİNA A.Ş

- İT projelerinin yönetilmesi;
- Uygulanabilir yeni teknolojilerin tekniksel değerlendirilmesi;
- SAP, IOT, Sipariş Tahmini Projesi gibi harici entegrasyon ve teknik altyapı projelerine nezaret edilmesi;

COCA COLA İÇECEK (CCİGO)

- İş süreçlerinin salesforce + native custom telefon uygulamaları ile otomatikleştirilmesi;

CLOUD TEAM

- Siemens Türkiye için destek süreçlerinin uygulanması;
- Dominos için şube denetim uygulamasının geliştirilmesi;
- Ebay Türkiye için (gittigidiyor.com) desteklenmesi ve geliştirilmesi;

AVANDEL

- ABD'nin Maryland eyalet hükümeti, Coca-Cola eForms, Porto Montenegro Marine, NP Marine/Singapur'un kullandığı InspectTHIS! saha denetim uygulama projesinin takım liderliği;
- İtalyan tıbbi ürün şirketi Esaote için hazırlanan fiyat ve fiyat teklif uygulamasının geliştirilmesi (CPQ);
- İşverenlerin attığı tweet'lerin otomatik veya elle görüntülenmesi, bütün konuşma geçmişinin saklanması ve konuyla ilgili işçilere bildirimin gönderilmesi için Salesforce-Twitter uygulamasının geliştirilmesi;

- Cep telefonlarından çekilen resim veya yazılı mesajlarla müşterilerden gelen bilgileri saha çalışanlarına gönderen PhoneGap esaslı hibrit mobil uygulamasının (Döküman Yönetim Sistemi) geliştirilmesi;

NETSİS

- Duru pazarlama için Redcode modülü kullanarakharici entegrasyon yapı sisteminin uyarlanması ve B2B projesinde yer alınması

YAYINLAR

1. Nuriyev U., Nuriyeva F., **Sadik T.**, “On a New Number System”, *Proceeding of the International Conference on Computer Science and Engineering, (UBMK 2016)*, Namık Kemal University, Tekirdağ, Turkey, October 20-23, 2016, pp. 825-827.

(<http://www.ubmk.org/bildiriler-kitabi/>).

2. **Sadik T.**, Tanir D., Nuriyeva F., “A new method for lossless compression of binary images,” 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), Malatya, Turkey, 2017, pp. 1-4. doi:10.1109/IDAP.2017.8090198

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8090198&isnumber=8090153>

3. Tanir D., **Sadik T.**, Nuriyev U., “A mathematical model for density-based clustering methods”, *Proceeding of the International scientific conference on “Theoretical and application problems of Mathematics”*, Azerbaijan, Sumgait State University, May 22-26, 2017, pp. 206-207.

http://sdu.edu.az/userfiles/file/conferences/math_01022016.pdf

EKLER

EK.1: 4.2.2.2 deki Problemin Çözümünde Kullanılan Algoritma

EK.2: Program Kodu



EK.1: 4.2.2.2 deki Problemin Çözümünde Kullanılan Algoritma

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
#define N 4000

int h;
shortint C[N];
staticint j=-1;
void trans(int l){
    switch (l){
        case 7:
            C[j++]=0;
            C[j++]=0;
            C[j++]=1;
            break;
        case 4:
            C[j++]=0;
            C[j++]=0;
            C[j++]=0;
            C[j++]=0;
            C[j++]=1;
            break;
        case 6:
            C[j++]=0;
            C[j++]=1;
            break;
        case 3:
            C[j++]=1;
            C[j++]=0;
            C[j++]=1;
            C[j++]=1;
            C[j++]=1;
            C[j++]=0;
            break;
        case 1:
            C[j++]=1;
            C[j++]=0;
            C[j++]=1;
            C[j++]=1;
            C[j++]=0;
            break;
        case 5:
            C[j++]=0;
            C[j++]=0;
            C[j++]=0;
            C[j++]=1;
            C[j++]=1;
            break;
```

```
case 2:  
    C[j++]=1;  
    C[j++]=1;  
    break;
```

```
case 0:  
    C[j++]=1;  
    C[j++]=0;  
    C[j++]=0;  
    break;
```

```
case 8:  
    C[j++]=0;  
    C[j++]=0;  
    C[j++]=0;  
    C[j++]=1;  
    C[j++]=0;  
    C[j++]=1;  
    break;
```

```
case 12:  
    C[j++]=1;  
    C[j++]=0;  
    C[j++]=1;  
    C[j++]=0;  
    C[j++]=1;  
    C[j++]=1;  
    break;
```

```
case 10:  
    C[j++]=0;  
    C[j++]=0;  
    C[j++]=0;  
    C[j++]=0;  
    C[j++]=0;  
    C[j++]=0;  
    break;
```

```
case 11:  
    C[j++]=0;  
    C[j++]=0;  
    C[j++]=0;  
    C[j++]=0;  
    C[j++]=0;  
    C[j++]=1;  
    break;
```

```
case 9:  
    C[j++]=1;  
    C[j++]=0;  
    C[j++]=1;  
    C[j++]=0;  
    C[j++]=1;  
    C[j++]=0;  
    break;
```

```
case 13:
```

```

        C[j++]=0;
        C[j++]=0;
        C[j++]=0;
        C[j++]=1;
        C[j++]=0;
        C[j++]=0;
        break;
    case 14:
        C[j++]=1;
        C[j++]=0;
        C[j++]=1;
        C[j++]=0;
        C[j++]=0;
        break;
    case 15:
        C[j++]=1;
        C[j++]=0;
        C[j++]=1;
        C[j++]=1;
        C[j++]=1;
        C[j++]=1;
        break;
    }
    h=j;
}
int main(){
    int B[16]={0},t[N];
    int i,k;
    char p[N],s[0];
    scanf("%s",&p);
    for(i=0;i<N;i++){
        s[0]=p[i];
        t[i]=atoi(s);
    }
    for(i=0;i<N;i=i+4){
        k=0;
        k=t[i]*8+4*t[i+1]+t[i+2]*2+t[i+3];
        B[k]++;
        trans(k);
    }
    for(i=0;i<16;i++)
        printf("%d=%d\n",i,B[i]);
    for(i=-1;i<j;i++)
        printf("%d",C[i]);

    printf("\n---%d---",h);}

```

EK.2

```
function[new]=new()
c = 0;
I=imread('blobs.png');

seq=I;
b='';
cstring={};
[m,n] = size(I);
ilk=0;

for j=1:m*n;
    if ilk ==0
        b=seq(j);
        c=c+1;

    ilk=-1;
    else if b==seq(j)
        c=c+1;
    else
        %%cstring(end+1,:)={strcat(int2str(b),'-',int2str(c))};
        cstring(end+1,:)={c};
        c=0;
        b=seq(j);
        c=c+1;
    end
end
newEncode(cstring)

function[new]=newEncode(cstring)

cM = containers.Map();

cM('1') = '0000001';
cM('2') = '0000010';
cM('3') = '0000011';
cM('4') = '0000100';
cM('5') = '0000101';
cM('6') = '0000110';
cM('7') = '0000111';
cM('8') = '0001000';
cM('9') = '0001001';
cM('10') = '0001010';
cM('11') = '0001011';
cM('12') = '0001100';
cM('13') = '0001101';
cM('14') = '0001110';
cM('15') = '0001111';
cM('16') = '0010000';
cM('17') = '0010001';
cM('18') = '0010010';
cM('19') = '0010011';
cM('20') = '0010100';
cM('21') = '0010101';
cM('22') = '0010110';
cM('23') = '0010111';
cM('24') = '0011000';
cM('25') = '0011001';
cM('26') = '0011010';
cM('27') = '0011011';
cM('28') = '0011100';
cM('29') = '0011101';
cM('30') = '0011110';
cM('31') = '0011111';
cM('32') = '0100000';
cM('33') = '0100001';
```

```
cM('34') = '0100010';
cM('35') = '0100011';
cM('36') = '0100100';
cM('37') = '0100101';
cM('38') = '0100110';
cM('39') = '0100111';
cM('40') = '0101000';
cM('41') = '0101001';
cM('42') = '0101010';
cM('43') = '0101011';
cM('44') = '0101100';
cM('45') = '0101101';
cM('46') = '0101110';
cM('47') = '0101111';
cM('48') = '0110000';
cM('49') = '0110001';
cM('50') = '0110010';
cM('51') = '0110011';
cM('52') = '0110100';
cM('53') = '0110101';
cM('54') = '0110110';
cM('55') = '0110111';
cM('56') = '0111000';
cM('57') = '0111001';
cM('58') = '0111010';
cM('59') = '0111011';
cM('60') = '0111100';
cM('61') = '0111101';
cM('62') = '0111110';
cM('63') = '0111111';
cM('64') = '1000000';
cM('65') = '1000001';
cM('66') = '1000010';
cM('67') = '1000011';
cM('68') = '1000100';
cM('69') = '1000101';
cM('70') = '1000110';
cM('71') = '1000111';
cM('72') = '1001000';
cM('73') = '1001001';
cM('74') = '1001010';
cM('75') = '1001011';
cM('76') = '1001100';
cM('77') = '1001101';
cM('78') = '1001110';
cM('79') = '1001111';
cM('80') = '1010000';
cM('81') = '1010001';
cM('82') = '1010010';
cM('83') = '1010011';
cM('84') = '1010100';
cM('85') = '1010101';
cM('86') = '1010110';
cM('87') = '1010111';
cM('88') = '1011000';
cM('89') = '1011001';
cM('90') = '1011010';
cM('91') = '1011011';
cM('92') = '1011100';
cM('93') = '1011101';
cM('94') = '1011110';
cM('95') = '1011111';
cM('96') = '1100000';
cM('97') = '1100001';
cM('98') = '1100010';
cM('99') = '1100011';
cM('100') = '1100100';
cM('101') = '1100101';
```

```

cM('102') = '1100110';
cM('103') = '1100111';
cM('104') = '1101000';
cM('105') = '1101001';
cM('106') = '1101010';
cM('107') = '1101011';
cM('108') = '1101100';
cM('109') = '1101101';
cM('110') = '1101110';
cM('111') = '1101111';
cM('112') = '1110000';
cM('113') = '1110001';
cM('114') = '1110010';
cM('115') = '1110011';
cM('116') = '1110100';
cM('117') = '1110101';
cM('118') = '1110110';
cM('119') = '1110111';
cM('120') = '1111000';
cM('121') = '1111001';
cM('122') = '1111010';
cM('123') = '1111011';
cM('124') = '1111100';
cM('125') = '1111101';
cM('126') = '1111110';
cM('127') = '1111111';

```

```

Rstring={};
say=0;
for j=1:length(cstring);
i=cstring(j);
    k=mod( i{:,1},127);
    r= floor(i{:,1}/127);

    for x=0:r-1
say=say+1;
        Rstring{end+1}= '0000000';
        end
        if k~=0
            Rstring{end+1}= cM(num2str(k)) ;

        end
    end
end
sonuc=length(Rstring)*7

```