T.C.
ADNAN MENDERES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
MATEMATİK ANABİLİM DALI
2016-YL-034

BELİRTEÇ SEÇİMİNİN HUFFMAN KODLAMASI ÜZERİNE ETKİSİ

Onur DİNCEL

Tez Danışmanı:
Yrd. Doç. Dr. Korhan GÜNEL

AYDIN-2016

T.C.
ADNAN MENDERES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜNE
AYDIN

Matematik Anabilim Dalı Yüksek Lisans Programı öğrencisi Onur DİNCEL tarafından hazırlanan *Belirteç Seçiminin Huffman Kodlaması Üzerine Etkisi* başlıklı tez, 28.06.2016 tarihinde yapılan savunma sonucunda aşağıda isimleri bulunan jüri üyelerince kabul edilmiştir.

	Ünvanı, Adı Soyadı	Kurumu	İmzası
Başkan :	Doç. Dr. Ali FİLİZ	Adnan Menderes Üniv	
Üye :	Yrd. Doç. Dr. Korhan GÜNEL	Adnan Menderes Üniv	
Üye :	Yrd. Doç. Dr. Refet POLAT	Yaşar Üniversitesi	

Jüri üyeleri tarafından kabul edilen bu Yüksek Lisans tezi, Enstitü Yönetim Kurulunun ……Sayılı kararıyla ……. tarihinde onaylanmıştır.

Prof. Dr. Aydın ÜNAY

Enstitü Müdürü

T.C.
ADNAN MENDERES ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜNE

Bu tezde sunulan tüm bilgi ve sonuçların, bilimsel yöntemlerle yürütülen gerçek deney ve gözlemler çerçevesinde tarafımdan elde edildiğini, çalışmada bana ait olmayan tüm veri, düşünce, sonuç ve bilgilere bilimsel etik kuralların gereği olarak eksiksiz şekilde uygun atıf yaptığımı ve kaynak göstererek belirttiğimi beyan ederim.

…./…../2016

Onur DİNCEL

ÖZET

BELİRTEÇ SEÇİMİNİN HUFFMAN KODLAMASI ÜZERİNE ETKİSİ

Onur DİNCEL

Yüksek Lisans Tezi, Matematik Anabilim Dalı
Tez Danışmanı: Yrd. Doç. Dr. Korhan GÜNEL
2016, 49 sayfa

Bu çalışmada, belirteç seçiminin istatistiksel veri sıkıştırma yöntemlerinden biri olan Huffman sıkıştırma algoritması üzerine etkisi ve verimliliği araştırılmıştır. Bu amaçla Huffman ağacı üretebilmek için düzgün deyimler kullanılarak tanımlanan farklı türdeki belirteçlerin sıkıştırmada sağladığı kazanç hesaplanmış ve sıkıştırma performansları karşılaştırılmıştır.

Çalışma beş ana bölümden oluşmaktadır. Giriş bölümünde, veri sıkıştırma tanımından ve veri sıkıştırma yöntemlerinin sınıflandırılmasından bahsedilmiştir. İkinci bölümde veri sıkıştırma yöntemlerinden olan istatistiksel veri sıkıştırma incelenmiş ve bilgi teorisi kavramları açıklanmıştır.

Çalışmanın üçüncü bölümünde, kullanılan belirteç türlerini açıklama adına *n*-gram, Türkçe heceleme algoritması ve düzgün deyim kavramlarından söz edilmiştir. Dördüncü bölümde ise *n*-gram, hece ve düzgün deyimlerin yanı sıra bunların birlikte kullanımları ile yaratılan belirteçler ile Huffman ağaçları oluşturulmuş ve sıkıştırma işlemleri gerçekleştirilmiştir. Sıkıştırma işlemi yedi farklı doküman üzerinde test edilmiştir ve her bir dokümanın kullanılan tüm belirteç türlerine ait sonuçları elde edilmiştir. Çalışmanın son bölümünde elde edilen sonuçlar tartışılmıştır.

Anahtar Kelimeler: Veri sıkıştırma, Huffman kodlaması, *n*-gram, Düzgün deyimler.

ABSTRACT

EFFECT OF TOKEN SELECTION ON HUFFMAN CODING

Onur DİNCEL

M.Sc. Thesis, Department of Mathematics
Supervisor: Asst. Prof. Korhan GÜNEL
2016, 49 pages

In this study, the effect and efficiency of token selection is investigated on the Huffman compression algorithm, one of the statistical data compression methods. To this end, compression gains for different types of tokens identified using regular expressions to produce Huffman tree is calculated and compression performance is compared.

The study consists of five main chapters. In the introductory chapter, it is mentioned that the definition of data compression and classification of the data compression methods. In the second chapter, statistical data compression, one of the data compression methods is examined and basic concepts in information theory are explained.

In the third chapter of the study, to describe used token type, it is introduced *n*-gram, Turkish syllabification algorithm and regular expression concept. Also in the fourth chapter, as well as n-gram, syllable and regular expression, Huffman trees with tokens created with collocation of their is generated and compression processing is performed. Compression processing is tested on seven different documents and the results of each document that is used for all tokens type is obtained. In the last chapter of the study, the results obtained is discussed.

Key Words: Data compression, Huffman coding, *n*-gram, Regular expressions.

ÖNSÖZ

Bu çalışmanın hazırlanmasının yanı sıra üniversite eğitimim boyunca değerli bilgisini, deneyimini ve zamanını, sabır ve anlayış ile benimle paylaşan saygıdeğer danışman hocam Yrd. Doç. Dr. Korhan GÜNEL'e ve yardımlarını hiçbir zaman esirgemeyen değerli hocam Yrd. Doç. Dr. Rıfat AŞLIYAN'a teşekkür ederim.

Hayatım boyunca maddi ve manevi her türlü desteği ile yanımda olan aileme de teşekkürü ayrıca bir borç bilirim.

Onur DİNCEL

İÇİNDEKİLER

KISALTMALAR VE SİMGELER DİZİNİ

C_i : i. belirtecin dokümandaki görüntülenme sıklığı

P_i : i. belirtecin dokümandaki görüntülenme olasılığı

H : Doküman entropisi

R : Veri artıklık değeri

l_i : i. belirtecin kod uzunluğu

S : Düzgün deyim kümesi

C : Ünsüz harf

V : Ünlü harf

$\sum$: Alfabe

$\sum^*$: Alfabenin Kleene kapanışı

A^* : A kümesinin Kleene kapanışı

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

EKLER DİZİNİ

1. GİRİŞ

İçinde bulunduğumuz XXI. yüzyıla, günümüzde verilen bir diğer isim de "Bilgi Çağı"dır. Bu bilgi çağının oluşumunda ise "Bilgi Teknolojileri" şüphesiz en büyük paya sahiptir. Bilgi teknolojisi; bilginin üretilmesi, toplanması, biriktirilmesi, işlenmesi, yeniden elde edilmesi, yayılması, korunması ve bunlara yardımcı olan araçlar olarak tanımlanabilir (Akkoyunlu, 1998).

Günümüzde bilgi teknolojilerinin insan hayatının vazgeçilmez bir parçası olması, onun tanımında yer alan özellikleri de doğrudan önemli hale getirmektedir. Bunun sonucunda, bilginin ham hali olarak tanımlanan "veri" için en az maliyetle en fazla depolamanın sağlanabilmesi ve iletişim ağları üzerinden hızlı bir biçimde aktarımının yapılabilmesi önemini artırarak devam ettirmektedir. Bu gereksinimleri gerçekleştirebilme noktasında ise veri sıkıştırma yöntemlerine sıkça başvurulmaktadır.

Veri sıkıştırma, 0 ve 1'den oluşan bitler halinde saklanan verinin çeşitli yöntemlerle orijinal haldeki boyutundan daha az yer kaplayacak şekilde tekrar işlenmesi ve bunu yaparken de işlenen verinin kullanışlılığını devam ettirebilmesi olarak tanımlanabilir. Verinin kullanışlılığı açısından bakıldığında veri sıkıştırma yöntemleri kayıplı ve kayıpsız sıkıştırma yöntemleri olarak iki grupta incelenir.

Kayıplı sıkıştırma, sıkıştırma sonucunda verinin kalitesini çok fazla etkilemeyecek ayrıntıların kaldırılmasıyla yapılan sıkıştırma türüdür. Sıkıştırma işlemi uygulandıktan sonra verinin eski haline getirilmesi mümkün olmadığından bu ismi almıştır. İnsanların görsel ve işitsel olarak algıları belli sınırlar içerisindedir. Bu yüzden de görüntü, video ve ses gibi çoklu ortam verilerinde bu sınırların dışında kalan veriler çıkartılarak sıkıştırma işlemi gerçekleştirilmiş olur. Bu durumda sıkıştırma yapılan verinin oluşturduğu bilgi hala bilgiyi alan tarafından anlamlı haldedir. Günümüzde kullanılan 'mp3' uzantılı ses dosyaları, 'jpg', 'png' uzantılı resim dosyaları ve 'mpg', 'mp4' uzantılı video dosyaları kayıplı sıkıştırma yöntemi uygulamalarına örnek olarak gösterilebilirler.

Kayıpsız sıkıştırma yöntemleri ise sıkıştırılan verinin tekrar açılması durumunda özgün verinin elde edilebilmesinin istendiği zamanlarda kullanılan yöntemlerdir. Örneğin metin verisi içeren dosyalara bu tür veri sıkıştırma yöntemleri

uygulanmalıdır. Veri sıkıştırma aşamasında veri kaybı olması durumunda metin eski haline döndürülemediğinden anlamsız veya yanlış bilgiye ulaşılabilir. Metin dosyaları yanında kaynak kodlar, çalıştırılabilir (executable) dosyalar gibi veri kaybının hataya sebep olabileceği dosyalar da kayıpsız veri sıkıştırmaya uygun olacaktır.

Kayıpsız sıkıştırma yöntemleri de istatistiksel (olasılık tabanlı) yöntemler ve sözlük tabanlı yöntemler olmak üzere ayrıca iki başlıkta incelenir. Sıkıştırma işlemini yapmak için istatistiksel model kullanıldığında bir sembolü kodlarken, semboller bit dizileri içinde kodlanır ve orijinal sembolden daha az bit kullanılmış olur. Sıkıştırmanın kalitesi de geliştirilen modele bağlı olarak değişir. Sözlük tabanlı sıkıştırma, veriyi sıkıştırmak için tamamen farklı bir metot kullanmaktadır. Bu tip algoritmalar bir sembolü değişken uzunluklu bit dizileri şeklinde kodlamazlar. Sembollerin değişken uzunluklu dizilerini tek belirteç şeklinde kodlarlar. Bu belirteçler sözlüğün dizin yapısını oluştururlar (Nelson ve Gailly, 1996). Sık kullanılan istatistiksel yöntemlere Shannon-Fano kodlaması (Shannon, 1948), Huffman kodlama (Huffman, 1952) ve aritmetik kodlama (Abramson, 1963) örnek olarak verilebilirken, sözlük tabanlı tekniklere ise Abraham Lempel ve Jacob Ziv tarafından geliştirilen LZ77 (Lempel ve Ziv, 1977) ve LZ78 (Lempel ve Ziv, 1978) yöntemleri örnek gösterilebilir.

Bu çalışmada metin sıkıştırma üzerine araştırmalar yapıldığından kayıpsız sıkıştırma yöntemleri incelenmiştir. Kayıpsız sıkıştırma yöntemlerinden ise istatistiksel veri sıkıştırma yöntemleri ele alınmıştır. Tezin ikinci bölümünde, bilgisayarın standart veri saklama şekli açıklanmış ve istatistiksel veri sıkıştırmanın ne olduğu ile birlikte daha iyi anlaşılabilmesi adına onunla ilgili kavramlar verilmiştir. Ayrıca istatistiksel veri sıkıştırma yöntemlerinden olan Shannon-Fano kodlaması ve Huffman kodlama üzerinde detaylı olarak çalışılmıştır. Üçüncü bölümde, çalışmada kullanılacak olan çeşitli belirteç türleri tanıtılmıştır. Bu amaçla Türkçenin yapısı ve Türkçe için heceleme algoritması, n-gram, düzgün deyimler kavramları ile bu kavramlar yoluyla oluşturulabilen belirteçler incelenmiştir. Tezin dördüncü bölümünde, oluşturulan belirteçlerin Huffman algoritması yardımıyla sıkıştırma işlemlerinde kullanımına dair yapılan çalışma açıklanmış ve çalışmanın deneysel sonuçları sunulmuştur. Son bölümde ise elde edilen sonuçlar tartışılmış ve öngörülen farklı yaklaşımlar ifade edilmiştir.

2. KAYNAK ÖZETLERİ

2.1. İstatistiksel Veri Sıkıştırma

Metin verilerinin bilgisayar sisteminde depolanması ve işlenmesi gerektiğinde her bir karaktere karşılık gelen bir sayısal değer atanır. Değer atama işlemleri bilgisayar 0 ve 1 bitleri ile çalıştığından ikilik sayı sisteminde olmaktadır. Bu sayı sistemine bağlı kalarak çeşitli kodlama sistemleri geliştirilmiştir. Geliştirilen her bir kodlamanın ise kendine özgü farklılıkları olduğundan temsil edilen karakterlere de farklı bit dizileri karşılık gelmektedir.

İlk olarak 1963 yılında ANSI (American National Standards Institute) tarafından standart olarak sunulan ASCII (American Standart Code for Information Interchange) kodlama sistemi ise günümüzde en sık kullanılan kodlama sistemidir. ASCII kodlama sisteminde her karakter 8 bit ile temsil edilmekle birlikte, bu karakterlerin ikilik sistemde karşılıkları en az 00000000 ve en fazla 11111111 olduğundan onluk sistemdeki karşılıklarının da 0 ve 255 arasında değiştiği görülmektedir. Buradan hareketle ASCII kodlama sistemiyle toplam 256 adet karakterin temsil edildiği bilgisine ulaşılabilinir.

256 adet karakterin tamamının hangi sayı sisteminde hangi kodlamaya karşılık geldiği standart olarak belirlidir. Örneğin, TAM kelimesinde onluk sistemde 84, 65 ve 77 numaralı ASCII karakterleri bulunur ve TAM kelimesi bilgisayarda ikili sayı sistemi karşılığı olarak 01010100 01000001 01001101 sayı dizisi ile tutulur.

ASCII kodlama sisteminde olduğu gibi her sembole aynı sayıda bitin atanması “sabit uzunluklu kodlama” olarak adlandırılır. Sembol dizilerini en az bitle temsil edebilmek adına metinde sık tekrarlanan sembollere daha az bit atama yoluyla her bir sembol farklı bitlerle gösterilebilir. Buna “değişken uzunluklu kodlama” denir.

Sıkıştırma işlemi için üzerinde çalışılan veri kümesindeki her bir sembolün ayrı ayrı veya birkaç sembolün birlikte kullanılmasıyla oluşturulan alt sembol gruplarının doküman içindeki görüntülenme olasılıklarının bulunması ve bu olasılık sonuçlarına bağlı olarak aynı veri kümesinin daha az yer kaplayacak şekilde farklı kodlarla temsil edilmesi mümkündür. Bu işleme “istatistiksel veri sıkıştırma” adı verilir. İstatistiksel veri sıkıştırma ise çeşitli teknikler kullanılarak

gerçekleştirilebilmektedir. En fazla kullanılan istatistiksel veri sıkıştırma teknikleri Shannon-Fano kodlaması, Huffman kodlama, aritmetik kodlama ve PPM (Prediction by Partial Machine) (Cleary ve Witten, 1984)'dir.

2.2. İstatistiksel Veri Sıkıştırma İle İlgili Temel Kavramlar

Değişken uzunluklu kodların tasarlayıcıları, birden fazla anlama gelmeyecek şekilde kodlanabilen kodlar atama ve en az ortalama boyutlu kodlar atama şeklinde iki problemle baş etmek zorundadır (Salomon ve Motta, 2010). Bu iki problemi çözümleyebilmek ve istatistiksel sıkıştırma yöntemlerini daha iyi anlayabilmek adına bazı önemli bilgi teorisi kavramlarının bilinmesi gerekmektedir. Bu kavramların bazıları aşağıda sunulmuştur.

2.2.1. Ön Ek Kodu ve Tek Türlü Çözülebilirlik

İstatistiksel veri sıkıştırma gibi kayıpsız sıkıştırma yöntemlerinde kodlanan veriden tekrar özgün verinin elde edilebilmesi oldukça önem arz etmektedir. Bunun gerçekleştirilebilmesi adına ise "ön ek kodu" ve "tek türlü çözülebilirlik" kavramlarının bilinmesi gereklidir.

Örnek olarak {v, e, r, i} kümesindeki sembollere karşılık gelen kodlar sırasıyla {0, 10, 010, 100} şeklinde tanımlanmış olsun ve 010010100 şeklindeki kodlanmış bit dizisinin kodunu çözerek özgün veriyi elde etmeye çalışalım. Bit dizisini 0 10 010 100 olarak çözümleyerek "veri" çıktısını elde edebileceğimiz gibi aynı bit dizisini 010 0 10 100 şeklinde de çözümleyebileceğimizden "rvei" çıktısını elde etmemiz de mümkündür. Bunlar dışında birkaç farklı çıktı daha elde edilmesi olasıdır. Bu durumdaki belirsizliği gidermek için yapılan uygun bir kodlama sonucunda, belirsizliğin ortadan kaldırılması ile elde edilen koda ise "tek türlü çözülebilir kod" denmektedir. Kodun tek türlü çözülebilir olması sıkıştırılmış veriden orijinal verinin dışında başka bir veri elde etme ihtimalinin olmadığı anlamına gelmektedir. Tek türlü çözülebilirliği sağlamak için kodlama sırasında kodlanan her bir sembole karşılık gelen kodları birbirinden ayırarak sembol bit sınırlarını belirleyen ayrı bir sembol kullanılabilir. Örnekteki 010010100 bit dizisi 0-10-010-100 şeklinde kod sözcükleri arasında "-" işareti kullanılarak kodlanırsa kod açma sırasında bu kısıtlara bağlı kalarak kod çözüleceğinden özgün verinin elde edilmesi kesinleşmiş olur. Fakat kullanılan sembollerin de kodlama sırasında veri

dosyasına eklenmesi gerektiği açıktır. Bu yüzden böyle bir kodlama yöntemi oldukça maliyetli ve kullanışsızdır. Dolayısıyla ek semboller kullanmadan, sadece bit dizisine bağlı kalarak tek türlü çözülebilir kodlar tasarlamak faydalı olacaktır. Tek türlü çözülebilir kod olarak ise "ön ek kodu" olarak adlandırılan kodlama yöntemini kullanmak oldukça kullanışlıdır.

Bir kodlamanın ön ek kodu olması veri kümesindeki hiçbir sembolün diğer sembollerin ön eki durumunda olmaması anlamına gelmektedir. Kodlamanın ön ek durumunda olup olmadığını görebilmek için veri kümesindeki herhangi bir sembolün kodunun boyutu "*n*" ile tanımlanır. Ele alınan sembolün uzunluğu, kendisi ile aynı veya kendisinden daha uzun boyutlu semboller ile karşılaştırılır. Eğer sembol kodu, karşılaştırma yapılan sembollerin kodunun ilk "*n*" bitinden farklılık gösteriyor ise sembolün ön ek durumunda olmadığı söylenir. Veri kümesindeki bütün semboller için bu karşılaştırmanın doğrulanması sonucunda ise kodlamanın ön ek kodu olduğu söylenebilir. Örneğin {v, e, r, i} kümesine karşılık gelen kodlar {0, 10, 111, 110} şeklinde ön ek kodu ile kodlanmışsa 010010100 bit dizisinin vereceği çıktı "veveev" şeklinde olacaktır ve bundan başka bir çıktı üretilemeyeceğinden ön ek kodun aynı zamanda tek türlü çözülebilir kod olarak da tanımlanabileceği söylenebilir.

Ek olarak, herhangi bir kodun ikili ağaç yapısına yerleştirilerek gösterimi de mümkündür. Bu yerleştirme sonucuna göre sembollerin tümünün ayrı ayrı ağacın yaprakları olup olmadığına bakarak da ön ek kod olup olmadığına karar verilebilir. Tüm semboller yapraklara ait ise kodun ön ek kodu olduğu söylenebilir.

2.2.2. Entropi

Temelde fizik anabilim dalında termodinamik bilim dalına ait bir kavram olan entropi, 1940'ların sonunda Claude Shannon tarafından, yine kendisinin yaratmış olduğu bilgi teorisine uyarlanmıştır (Shannon, 1948). Shannon bilginin anlamını düşünmeksizin bir sembolde saklanan bilgi miktarını ölçmek için araçlar geliştirmeye çalışmıştır. Logaritma fonksiyonu ve bilgi arasında bağlantı keşfetmiş ve P olasılıklı bir sembolün bilgi içeriğinin (bitler bazında) $-\log_2 P$ olduğunu göstermiştir. Eğer logaritmanın tabanı e ise, o zaman bilgi 'nat' olarak adlandırılan birimlerde ölçülür. Taban 3 ise, bilgi birimleri 'trit'lerdir ve taban 10 ise de birimler 'Hartley' olarak adlandırılır (Salomon ve Motta, 2010).

Bilgisayar mimarisi 0 ve 1 olmak üzere ikilik sistemde bitlerle çalıştığından logaritma tabanı 2 olarak tanımlanmış şekilde $-\log_2 P$ bilgi içeriği kullanıldığında, a_i gibi tek bir sembolün entropisi, $-P_i \log_2 P_i$ olarak ifade edilmektedir. P_i doküman içerisindeki a_i sembolünün oluşma olasılığı olarak tanımlanır. Doküman içerisinde n tane farklı sembol var ise, a_i sembolü için ihtiyaç duyulan ortalama bit sayısı Eşitlik 2.1 deki gibi gösterilmekte ve H ile temsil edilmektedir (Salomon, 1997). Formülden mesajdaki bilginin arttıkça entropinin de arttığı çıkarımı yapılabilir.

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i \tag{2.1}$$

İlgili verilerin formüle yerleştirilmesiyle ortaya çıkan bu değer, sembollerin temsil edilebilmesi için gereken en az bit sayısını gösterir. Bundan dolayı, kayıpsız sıkıştırma işlemlerinde elde edilen kazancı hesaplarken entropi değerinden faydalanmak oldukça kullanışlıdır.

Örnek olarak Çizelge 2.1 deki gibi, bir metin içinde bulunma olasılıkları verilen ve yedi sembolden oluşan bir küme alınmış olsun.

Çizelge 2.1. Olasılık tablosu

	Sembol	**Olasılık**
1.	a	0,24
2.	n	0,21
3.	t	0,16
4.	e	0,14
5.	r	0,10
6.	s	0,09
7.	m	0,06

Elde edilen olasılık değerleri Eşitlik 2.1 de yerine konulursa veri kümesinin entropisi yaklaşık olarak 2,675 bulunur ve bu sonuç veri kümesine sıkıştırma işlemi uygulandığında sembol başına düşen bit sayısının en iyi ihtimalle 2,675 olacağını gösterir.

2.2.3. Artıklık

n sembollü bir veri kümesinin entropisi P_i olasılıklarına bağlıdır ve sembollerin tümünün eşit olasılığa sahip olması durumunda entropi en büyük değerini alır. Entropinin en üst sınır değerine ulaşması verilerin daha fazla sıkıştırılamayacağı anlamına gelir. Artıklık ise bir sembol kümesinin olası en büyük entropisi ile gerçek entropi değeri arasındaki fark olarak tanımlanır ve Eşitlik 2.2 deki gibi formülleştirilebilir. Formül 'artıklık' tanımının İngilizce karşılığı olan 'redundancy' kelimesinin ilk harfi olan '*R*' ile temsil edilmiştir.

$$R = \left[-\sum_{1}^{n} P \log_2 P \right] - \left[-\sum_{i=1}^{n} P_i \log_2 P_i \right] = \log_2 n + \sum_{i=1}^{n} P_i \log_2 P_i \qquad (2.2)$$

Bir başka ifade ile artıklık kavramını entropisinin üst sınırına yaklaştıkça 0'a giden bir nicelik olarak tanımlamak da mümkündür. Buradan hareketle, tamamen sıkıştırılmış, yani artıklığın olmadığı bir veri için $\log_2 n + \sum_{i=1}^{n} P_i \log_2 P_i = 0$ olur.

Örnek olarak Çizelge 2.1 deki veri kümesi tekrar ele alınırsa, yedi sembollü bir küme olduğundan, $\log_2 n = \log_2 7 = 2,807$ bulunur. Artıklık değeri ise bu değerden entropi değerinin çıkarılması ile bulunacağından, artıklık 2,807 – 2,675 = 0,132 olarak bulunur.

Artıklık kavramının değişken uzunluklu kodlamalar için özelleştirilmiş hali olan bir başka formül ise Eşitlik 2.3 te gösterilmektedir.

$$R = \sum_{i=1}^{n} P_i l_i - \sum_{i=1}^{n} \left[-P_i \log_2 P_i \right] \qquad (2.3)$$

Burada l_i her bir sembol veya sembol kümesinin sıkıştırma işlemi gerçekleştirildikten sonra sahip olduğu uzunluğu belirtmektedir. $\sum_{i=1}^{n} P_i l_i$ ifadesi ise "ortalama kod uzunluğu" olarak tanımlanır.

2.2.4. Sıkıştırma Oranı

Kayıpsız sıkıştırma algoritmalarında sıkıştırma gerçekleştirildikten sonra kullanılan yöntemin verimliliği ölçülmelidir. Bunun için birkaç yaklaşım olmakla birlikte, sıkıştırılmak istenen veri kümesinin sıkıştırma yöntemi uygulandıktan sonra kapladığı boyutun, sıkıştırma gerçekleştirilmeden önce kapladığı boyuta oranını tanımlayan kavram olan sıkıştırma oranı bunlardan birisidir ve Eşitlik 2.4 te gösterilmiştir.

$$\text{Sıkıştırma oranı} = \frac{\text{Sıkıştırma sonrası boyut}}{\text{Sıkıştırma öncesi boyut}} \tag{2.4}$$

2.3. Shannon-Fano Kodlaması

Shannon-Fano kodlaması, ismini Claude Shannon ve Robert Fano'dan alan bir sıkıştırma algoritmasıdır (Salomon ve Motta, 2010).

Bu kodlamada amaç, bir sembol kümesindeki sembollerin olasılıklarına göre, en yüksek olasılığa sahip olan sembole olası en düşük kod gelecek şekilde kod atamaktır. Buradan hareketle ilk olarak sembollerin metin içinde görüntülenme olasılıklarının belirlenmesiyle işe başlanır. Her bir sembole ait görüntülenme olasılıkları elde edildikten sonra, olasılık değerleri büyükten küçüğe doğru olacak şekilde semboller sıralanır.

Örnek olarak Çizelge 2.1 deki 7 sembolden oluşan küme alınsın. Bu kümeye ait sembollerin olasılık tablosu oluşturulduktan sonra olasılık değeri yarıdan (ya da yarıya en yakın olacak şekilde) bölünüp iki ayrı grup oluşturulur. Oluşturulan gruplardan üsttekine 1, alttakine 0 atanır. Sıra değiştirilebilirdir. Yani üst grup 0, alt grup 1 olabilir. Fakat diğer adımlarda da bu sıra izlenmelidir.

Örnek için ilk adım yapılırsa, 1 tam olasılığı 0,5 – 0,5 olarak veya buna yakın değerde ayrılmalıdır. Burada 0,45 ve 0,55 olasılığa sahip alt kümeler en uygunu olacaktır. Bu şekilde parçaların her birinde tek bir sembol kalana kadar adımlar sürdürülürse Çizelge 2.2 deki kodlama ağacı ve kodlar elde edilmiş olur.

Çizelge 2.2. Shannon-Fano kodlama ağacı

	Sembol	Olasılık	ADIMLAR				Kod Sözcüğü
1.	a	0,24	1	1			11
2.	n	0,21	1	0			10
3.	t	0,16	0	1	1		011
4.	e	0,14	0	1	0		010
5.	r	0,10	0	0	1		001
6.	s	0,09	0	0	0	1	0001
7.	m	0,06	0	0	0	0	0000

Dikkat edilirse hiçbir kod sözcüğü bir diğer kod sözcüğünün ön eki durumunda değildir. Örneğin ön eki “10” olan hiç bir kod sözcüğü mevcut değildir. Aynı şekilde ön eki “001” olan hiç bir kod sözcüğü yoktur.

Kod sözcükleri bulunduktan sonra Shannon-Fano sıkıştırması ile sembol başına kaç bitlik bir ihtiyaç olduğunu bulmak için ortalama kod uzunluğu hesaplanır: Bunun için Çizelge 2.2 de yer alan her bir sembolün olasılık değeri ile sembol için gereken bit sayısı çarpılır ve tüm semboller için bulunan sonuçlar toplanarak ortalama kod uzunluğu bulunur. Sonuç olarak ortalama kod uzunluğu = (0,24)×2 + (0,21)×2 + (0,16)×3 + (0,14)×3 + (0,10)×3 + (0,09)×4 + (0,06)×4 = 2,7 bit/sembol olacaktır.

Sıkıştırma öncesi ise her bir sembol 8 bit ile temsil edildiğinden 5,3 bit/sembol kazanç elde edilmiştir. Bu ise sıkıştırmanın (5,3) / 8 = 0,6625 oranında bir verim sağladığını gösterir.

2.4. Huffman Kodlaması

Huffman Algoritması, MIT’de Robert Fano tarafından kurulan bir sınıfta, öğrenci olan David Huffman tarafından 1952 yılında geliştirilmiş ve bir veri kümesinde daha çok rastlanan sembolü daha düşük uzunluktaki kodla, daha az rastlanan sembolleri daha yüksek uzunluktaki kodlarla temsil etme mantığı üzerine kurulmuştur (Huffman, 1952).

Huffman tekniğini kullanarak bir veri kümesi üzerinde sıkıştırma yapabilmek için, o veri kümesinde bulunan her bir sembolün kaç kez tekrar ettiğinin bilinmesi gerekir. Bu amaçla veri kümesi içinde kullanılan tüm sembollerin tekrar etme sıklığını gösteren bir tablo oluşturulur. Bu tablo "frekans tablosu" olarak adlandırılmaktadır.

Verilerin frekans tablosunu elde etmek için Statik Huffman ve Dinamik Huffman olmak üzere iki yaklaşım söz konusudur.

Statik yöntemin seçilmesi durumunda da iki farklı şekilde işlem yapılabilir. Yöntemlerden birincisi, üzerinde çalışılan metin verisi hangi dilde işlenmişse o dile ait sabit, yani metin verisindeki sembollerin bulunma sıklıklarından etkilenmeyen bir frekans tablosunun kullanılmasıdır. Bu frekans tablosu her dilin kendine özgü olan karakteristiği sayesinde belirlenir. Örnek olarak; Türk alfabesinde en yüksek dereceye sahip olan, yani en sık kullanılan harfler "A, E, İ, N, R, L" üst-orta derecede olanlar "I, D, K, M" alt-orta derecede olanlar "U, Y, T, S, B, O" düşük dereceli olanlar "Ü, Ş, Z, G, Ç, H, Ğ, V, C, Ö, P, F, J" dir (Anonim, 2015). Buradan hareketle de sık kullanılan harflere düşük uzunluklu kodlar atanırken, düşük derecede kullanılan harflere ise daha uzun kodlar atanabilir. İngilizce ve Fransızca bir metinde ise en fazla kullanılan harflerin sırasıyla ETAOINSHRDLU ve ESARTUNILOC olduğu bilinmektedir (Salomon ve Motta, 2010). Her dilin karakteristiği farklı olduğundan bu yöntemin dezavantajı, yöntemin dile bağımlı olması, başka bir dile uygulandığında doğru sonuçlar vermemesidir.

Frekans tablosunu oluşturmak için kullanılan diğer yöntem ise metin verisinin tamamını tarayarak her bir sembolün frekansını bulmaktır. Bu yöntemin çalışılan metin dosyası içeriğine bağımlı dolayısıyla da dilden bağımsız olması bakımından birinci yönteme göre daha kullanışlı olduğu söylenebilir. Bununla birlikte sıkıştırılan verilerde geçen sembollerin frekansının da bir şekilde saklanma zorunluluğunun olması yöntemin dezavantajıdır (Algan, 2004).

Metin dosyası içindeki sembollere yeni değerler yani kodlar verebilmek için oluşturulan frekans tablosunu kullanarak bir "Huffman Ağacı" oluşturulması gerekmektedir. Huffman ağacı hangi sembolün hangi kodlarla temsil edileceğini belirlemeye yaramaktadır.

Bir Huffman ağacı oluşturabilmek için yukarıda da bahsedildiği gibi frekans tablosuna ihtiyaç duyulur. Ele alınan örnek bir metin dosyasından Çizelge 2.3 deki sembol frekansları elde edilmiştir.

Çizelge 2.3. Frekans tablosu

Sembol (Karakter)	Sembol Frekansı
a	27
e	26
d	21
b	20
c	15
f	14
g	12
ş	11

Adım 1. Huffman ağacını oluştururken ilk adımda frekans tablosundaki semboller Şekil 2.1 deki gibi küçükten büyüğe doğru sıralanır.

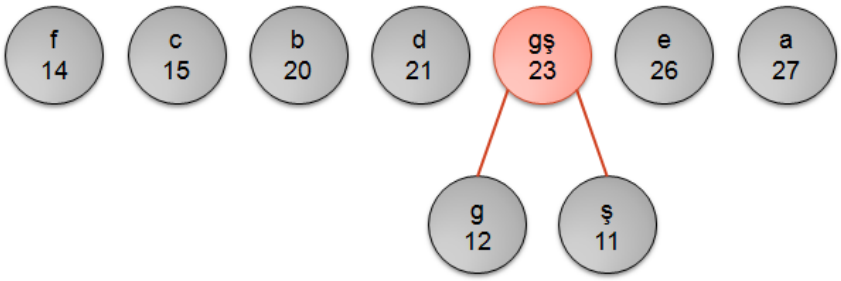
Şekil 2.1. Frekansa göre sembollerin sıralanması

Adım 2. Ağaç yapısını oluşturmak için en küçük frekansa sahip iki sembolün frekansları toplanarak yeni bir düğüm oluşturulur. Oluşturulan bu yeni düğüm diğer düğümler arasına küçükten büyüğe doğru sıralanma kuralına uygun olarak yerleştirilir.

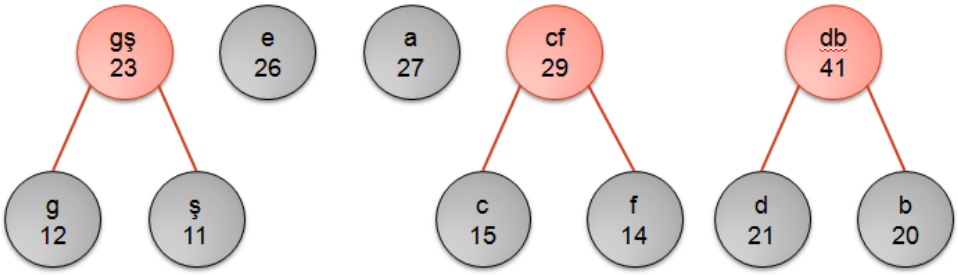
Örneğin Şekil 2.2 de “ş” ve “g” sembolleri toplanarak “23” frekanslı “gş” düğümü oluşturulur. 23 frekanslı sembol de “d” ve “e” sembolleri arasına yerleşir. “g” ve “ş” düğümleri ise yeni oluşturulan düğümün dalları şeklinde kalır.

Eğer birleştirme aşamalarında en düşük frekansa sahip ilk iki sıradaki sembollerden en az birisinde yeni oluşturulmuş bir düğüm bulunmuyorsa, yani

ikisi de yaprak ise, birleştirme ikinci adımdaki gibi devam eder. Şekil 2.3 de bu adımlar sonucunda elde edilen yapı görülmektedir.



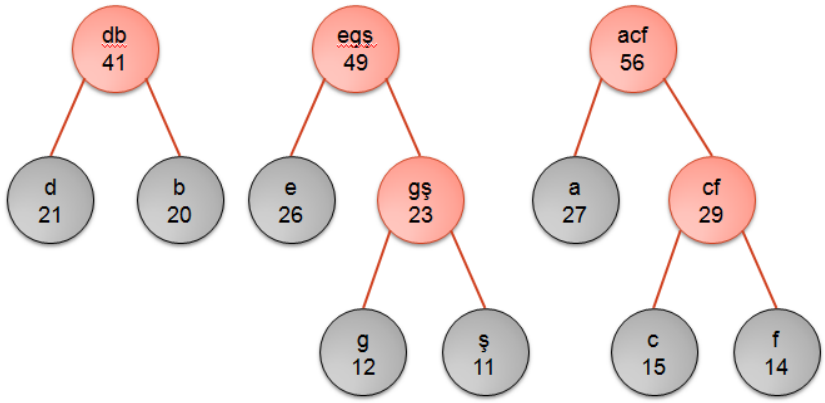
Şekil 2.2. İlk düğümün oluşturulması



Şekil 2.3. İkinci ve üçüncü düğümlerin oluşturulması

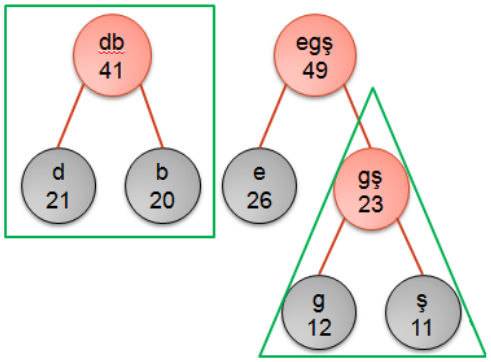
Adım 3. Huffman ağacı birleştirme işleminde yararlanılan birden fazla yöntem bulunmaktadır. Bu çalışmada iki farklı yöntem ele alınacaktır.

Birinci yöntemde, bir ağaç yapısı ile bir yaprağı birleştirirken yaprağın frekansının ağacın frekansından büyük veya küçük olması önem kazanacaktır. Eğer yaprağın frekansı düğümün frekansından büyükse yaprak sola düğüm sağa alınarak birleştirme işlemi yapılır. Aksi durumda ise yaprak ile düğüm bir yer değiştirme olmadan aynen birleştirilir. İki durumdan birisi ile birleştirme yapıldıktan sonra frekans sıralamasında uygun yere yerleştirilme işlemi yapılır. Şekil 2.4 de bu şekilde oluşturulan düğümler görülmektedir.



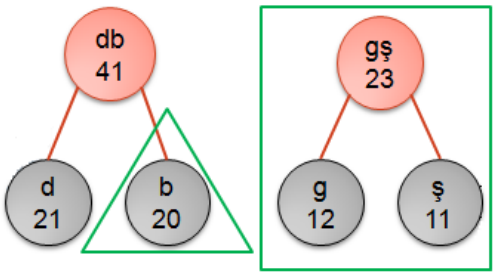
Şekil 2.4. Dördüncü ve beşinci düğümlerin oluşturulması

Adım 4. Şekil 2.4 de görüldüğü gibi düşük frekanslı ilk iki düğüm yapraklara sahip ağaç yapılarıdır. Bu tür iki ağaç birleştirilirken her iki ağacın sol yaprak frekansına göre değerlendirme yapılır. Eğer birinci ağacın sol düğümünün frekansı ikinci ağacın sol düğümünün frekansından küçükse, Şekil 2.5 de gösterildiği gibi ağaç1'in tamamı ile ağaç2'nin sağ düğümü birleştirilir.



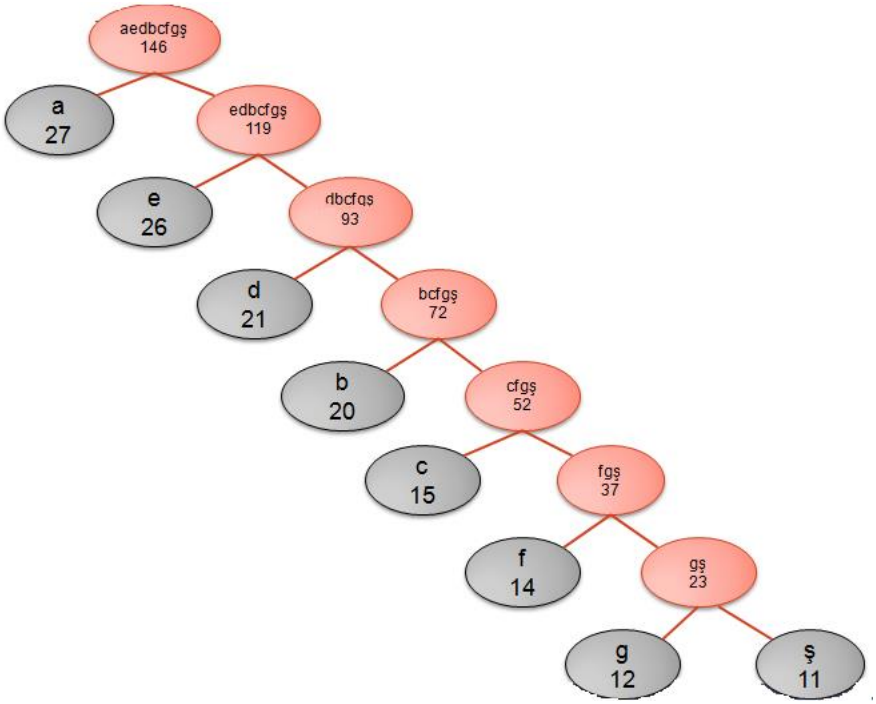
Şekil 2.5. İki ağacın birleştirilmesi

Eğer birinci ağacın sol frekansı ikinci ağacın sol frekansından büyükse Şekil 2.6 da gösterildiği gibi ağaç2'nin tamamı ile ağaç1'in sağ düğümü birleştirilir.



Şekil 2.6. İki ağacın birleştirilmesi

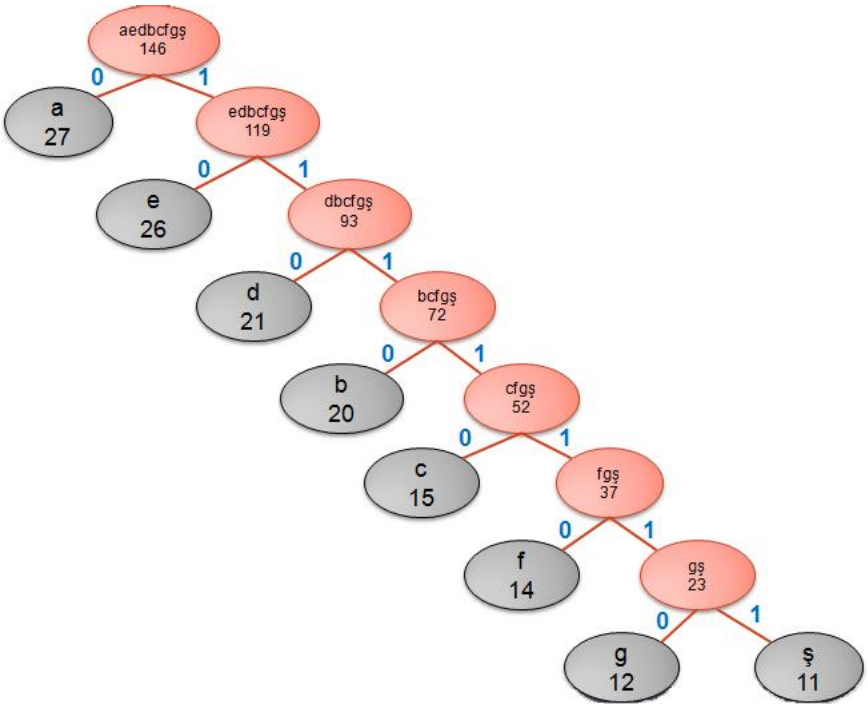
Ağaç birleştirilmesi sırasında yukarıda gösterilen ihtimaller dâhilinde birleştirme işlemlerine devam edildiğinde Şekil 2.7 deki ağaç yapısı oluşur. Bu ağaç yapısı bir Huffman ağacıdır.



Şekil 2.7. Huffman ağaç yapısı

Huffman ağacı oluşturulduktan sonra her bir sembole karşılık gelen kod oluşturulabilir. Sembol kodunu oluştururken ağacın en tepesinden yani kök düğümden başlanır. Kök düğümün sol ve sağ düğümlerine giden dallarına sırasıyla

"0" ve "1" kodları verilir. Bu sıra değiştirilebilirdir. Fakat ilk hangi tarafa “0” veya “1” verildiyse aynı şekilde devam edilmelidir. Dalların kodlarla işaretlenmiş hali ise Şekil 2.8 de gösterilmektedir.



Şekil 2.8. Kodlanmış Huffman ağacı

Dallar kodlarla işaretlendikten sonra ise her bir sembol için, kökten başlayarak sembolün bulunduğu yaprağa gelene kadar olan kodlar birleştirilip sembol kodu bulunur. Çizelge 2.4 de her bir sembolün Huffman kodu karşılığı gösterilmektedir.

Shannon-Fano kodlamasında olduğu gibi burada da ön ek durumu söz konusudur. Yani hiçbir Huffman kodu bir diğer Huffman kodunun ön eki durumunda değildir. Örneğin ön eki “1110” olan hiç bir Huffman kodu mevcut değildir. Huffman kodları ile kodlanmış herhangi bir veri dizisi de “tek türlü çözülebilir bir kod” olduğu görülmektedir.

Çizelge 2.4. Huffman kod tablosu

Sembol (Karakter)	Sembol Frekansı	Huffman Kodu	Bit Sayısı
A	27	0	1
E	26	10	2
D	21	110	3
B	20	1110	4
C	15	11110	5
F	14	111110	6
G	12	1111110	7
Ş	11	1111111	7

Kodlama işlemi yapıldıktan sonra ise sıkıştırmanın ne oranda etkili olduğuna bakılır.

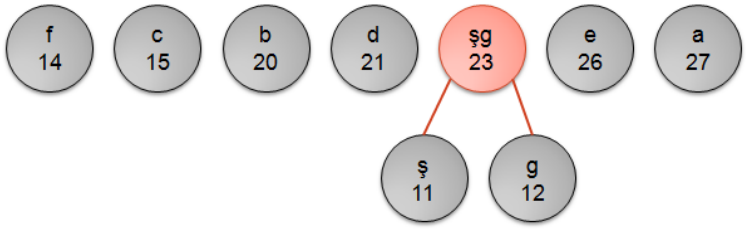
Huffman algoritması kullanılarak sıkıştırma yapılırsa, kaç bitlik alana ihtiyaç duyulduğunu hesaplamak için sembollerin frekansları ile temsil edildikleri bit sayılarını çarpar ve tüm sembollerin çıkan sonuçlarının toplamını buluruz. Sonuç olarak Huffman sıkıştırması ile gereken bit sayısı (27×1 + 26×2 + 21×3 + 20×4 + 15×5 + 14×6 + 12×7 + 11×7) = 542 bit olarak bulunur.

Sıkıştırma öncesi gereken bit sayısı bulunmak istenirse, her bir karakter 8 bit ile temsil edildiğinden toplam karakter sayısı olan (27+26+21+20+15+14+12+11) = 146 ile 8 sayısının çarpılması gerekir. Bu durumda, orijinal veri sıkıştırılmadan saklanırsa 146×8 = 1168 bit gerekmektedir.

Sonuç olarak 1168 bitlik gereksinim 542 bite indirilmiştir. Bu da yaklaşık %53,6 gibi bir sıkıştırma gerçekleştirilmiş demektir. Gerçek bir sistemde sembol frekanslarını da saklamak gerektiği için sıkıştırma oranı %53,6'dan biraz daha az olacaktır. Bu fark genelde sıkıştırılan veriye göre çok küçük olduğu için ihmal edilebilir. Elde edilen sıkıştırma oranı başta bahsettiğimiz iki yöntemden ilkine aittir. İkinci yaklaşımda ise sırasıyla aşağıdaki adımlar uygulanır.

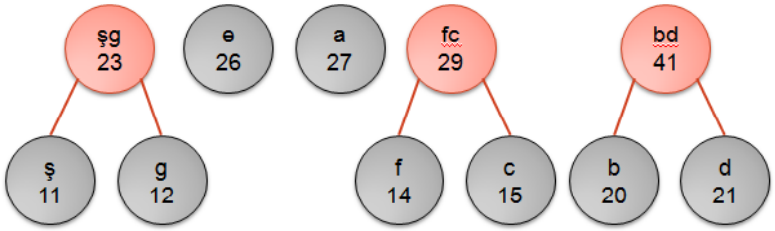
Adım 1. Öncelikle Şekil 2.1 deki gibi sembol frekanslarına göre bir sıralama ile başlanır.

Adım 2. Sıralama sonucu ilk iki yaprak birleştirilerek bu yaprakların sembollerinin ve frekans değerlerinin toplamından oluşan bir düğüm oluşturulur. Bunu yaparken küçük frekanslı yaprak solda büyük olan ise sağda kalacak şekilde birleştirme işlemi yapılır. Oluşturulan yeni düğüm de diğer düğümler arasına uygun yere yerleştirilir. Örneğin Şekil 2.9 da görülen "ş" ve "g" sembolleri için birleştirme yapılmıştır.



Şekil 2.9. İkinci yöntemde ilk düğümün oluşturulması

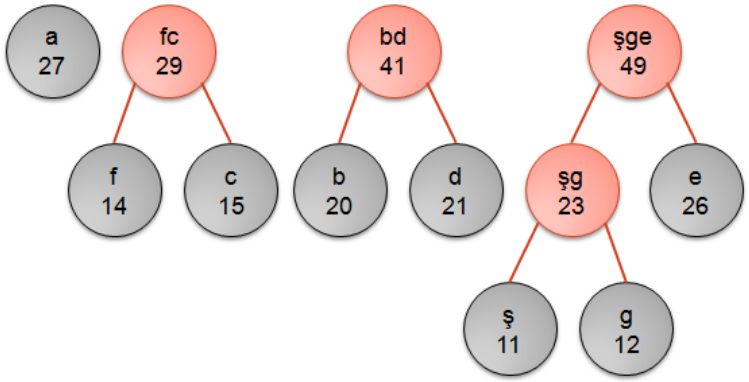
İşlemler diğer yapraklar için de aynen yapılarak Şekil 2.10 daki yapı oluşur.



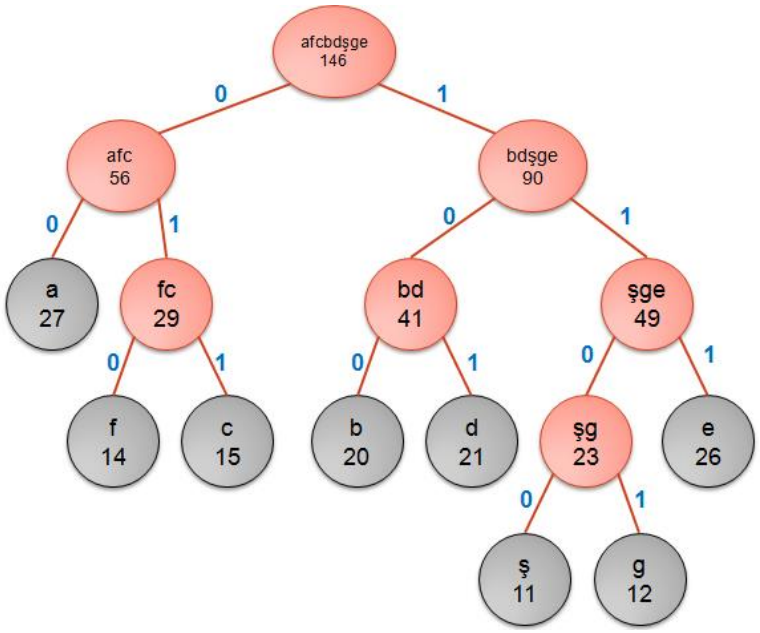
Şekil 2.10. İkinci ve üçüncü düğümlerin oluşturulması

Adım 3. Şekil 2.10 da görüldüğü gibi en düşük frekanslı ilk iki düğümden birisi yaprak birisi ağaçtır. Bu durumda ilk yöntemde yaprağın frekans durumuna göre birleştirme yapılırken ikinci yöntemde Adım 2. deki gibi birleştirme işlemine devam edilir. Şekil 2.11 de bu durum gösterilmektedir.

Adım 4. Birleştirme işlemine Adım 2 deki gibi devam edilerek Şekil 2.12 deki Huffman ağaç yapısı oluşturulmuş olur. Kodlama işlemi ise ilk yöntemde bahsedildiği gibi sağ dalları 1, sol dalları 0 ile işaretleyerek yapılır.



Şekil 2.11. Dördüncü düğümün oluşturulması



Şekil 2.12. Huffman ağacının kodlanmış son hali

Huffman ağacının kodlanmış halinde kökten başlayarak oluşturulan, tüm semboller için Huffman kodları Çizelge 2.5 de gösterilmektedir.

Huffman sıkıştırmasının bu yönteminde kaç bit alan gereksinimi olduğuna bakılırsa (27×2 + 26×3 + 21×3 + 20×3 + 15×3 + 14×3 + 12×4 + 11×4) = 434 bit olarak bulunur.

Çizelge 2.5. İkinci yöntem ile oluşturulan Huffman kodlarının tablosu

Sembol (Karakter)	**Sembol Frekansı**	**Huffman Kodu**	**Bit Sayısı**
a	27	00	2
e	26	111	3
d	21	101	3
b	20	100	3
c	15	011	3
f	14	010	3
g	12	1101	4
ş	11	1100	4

Huffman veri sıkıştırma uygulamadan 1168 bit gerektiği daha önce bulunmuştu. Bununla birlikte ikinci yöntemimizin sıkıştırma oranı da yaklaşık %62,8 bulunur.

Buna göre birinci yöntemle ikinci yöntem arasında bir karşılaştırma yaparsak, birincide 542 bite ihtiyaç varken ve sıkıştırma oranı %53,6 iken ikincide ihtiyaç 434 bite düşmüş ve sıkıştırma oranı da artmıştır.

Verilen bir doküman ve seçilen belirteç türüne göre Huffman ağacını ve ilgili Huffman kod tablosunu oluşturan kodlar Ek 1 de sunulmuştur.

3. MATERYAL VE YÖNTEM

3.1. Belirteç Türleri

Veri sıkıştırma algoritmalarından genel olarak karakter veya sözcük dizilimleri temel belirteç türü olarak seçilir ve her bir belirtece ayrı bir kod verilir. Bu çalışmada karakter ve sözcük dizilimlerine ek olarak, Türkçe yapısına uygun olacak şekilde hece dizilimleri ve dilden bağımsız olarak tanımlanabilecek düzgün deyim dizilimleri belirteç türü olarak tanımlanmıştır. Bu bölümün takip eden kısımlarında tanımlanan bu belirteç türlerine dair bilgilendirme ve örneklendirmeler yapılmıştır.

3.2. *n*-gram

Günümüzde konuşma tanıma, doküman sınıflandırma, doküman sıkıştırma gibi bilgisayar ortamında gerçekleştirilen ve doğal dil işleme alanına giren tüm uygulamalarda dilin modellenmesi oldukça önem taşımaktadır. Dilin modellenmesi, üzerinde çalışılan dile ait derlem veya derlemlerin istatistiksel olarak incelenmesi ve dilin yapısı hakkında bilgi verecek özelliklerin çıkarılması anlamına gelmektedir. Dilin modellenmesi birkaç farklı yöntem kullanılarak yapılabilmekle birlikte en sık kullanılan yöntemlerden birisi *n*-gram dil modellemesidir.

n-gram, eldeki sembol dizisinden her biri *n* birimden oluşan alt diziler elde ederek diziyi istenilen birimlere ayrıştıran olasılıksal bir modeldir. Burada bahsedilen birimler çalışmanın amacına göre seçilen karakter, hece, sözcük gibi ses birimleridir.

Bir *n*-gram'ın boyutu 1 ise "monogram"; boyut 2 ise "bigram"; boyut 3 ise "trigram"; ve boyut 4 veya daha fazla ise de "*n*-gram" veya "(*n*-1). dereceden Markov modeli" olarak adlandırılır (Zhuang vd., 2004). *n*-gram'ın Markov modeli olarak da adlandırılmasının sebebi Markov modellerin stokastik bir sürece sahip olmasıdır. Bunun anlamı mevcut durumdan yola çıkılarak olasılıksal değerlendirmelerle gelecek durumların öngörülebilmesi ve bu gerçekleştirilirken geçmiş durumların gelecek durumlara etki etmemesidir. Bu ifade *n*-gram

modellerini tanımlarken (*n*-1) terim kullanılarak *n*. terimin tahmin edilmesi olarak yeniden ifade edilebilir ve Eşitlik 3.1 deki gibi formülleştirilebilir.

$$P(t_i \mid t_{i-(n-1)} \ldots t_{i-1}) = \frac{C(t_{i-(n-1)} \ldots t_{i-1} t_i)}{C(t_{i-(n-1)} \ldots t_{i-1})} \tag{3.1}$$

Eşitlik 3.1 de t_i doküman içerisindeki karakter, hece, sözcük gibi ses birimlerini belirtmekle birlikte $t_{i-(n-1)} \ldots t_{i-1}$ ondan önce gelen (*n*-1) birimi ifade etmektedir. Ayrıca $C(t_{i-(n-1)} \ldots t_{i-1} t_i)$ doküman içerisindeki $t_{i-(n-1)} \ldots t_{i-1} t_i$ dizisinin toplam frekansını, $C(t_{i-(n-1)} \ldots t_{i-1})$ de $t_{i-(n-1)} \ldots t_{i-1}$ dizisinin toplam frekansını vermektedir.

Aşağıdaki örneklerde farklı ses birimleri farklı *n*-gram düzeyleriyle gösterilmiştir. Örneklerde ilk 3 *n*-gram düzeyi incelenmiş olup diğer düzeylerde de benzer şekilde işlem yapılır.

Karakterlerin ses birimleri olarak kabul edildiği "istatistik" sözcüğü ele alınsın.

1-gram: i – s – t – a – t – i – s – t – i – k

2-gram: is – st – ta – at – ti – is – st – ti – ik

3-gram: ist – sta – tat – ati – tis – ist – sti – tik

Hece ses birimleri kullanarak "istatistiksel" sözcüğünü ele alalım.

1-gram: is – ta – tis – tik – sel

2-gram: ista – tatis – tistik – tiksel

3-gram: istatis – tatistik – tistiksel

Ses birimlerinin sözcük seçilmesi durumunu incelerken de "istatistiksel veri sıkıştırma" sözcük grubu ele alınsın.

1-gram: istatistiksel – veri – sıkıştırma

2-gram: istatistiksel veri – veri sıkıştırma

3-gram: istatistiksel veri sıkıştırma

3.3. Türkçenin Yapısı ve Heceleme Algoritması

Türkçe, Ural-Altay dil ailesinin Altay dilleri koluna ait bir dildir. Morfolojik açıdan ise Korece, Macarca, Fincenin de olduğu gibi sondan eklemeli bir dildir. Bu özellik sayesinde en küçük anlamlı dil yapı birimi olan bir kök sözcüğe ekler ekleyerek birçok sözcük elde edilebilmektedir. Örneğin, "göz" kök sözcüğünden "göz-lük", "göz-lük-çü", "göz-lük-çü-lük" şeklinde yeni sözcükler türetilebilmektedir. Bunun yanı sıra ses uyumu kuralları gibi çeşitli kurallar dilin yapısının analiz edilmesini zorlaştırmaktadır. Türkçe Çizelge 3.1 de gösterilen sekizi sesli yirmi biri sessiz olmak üzere 29 farklı harfe sahiptir.

Çizelge 3.1. Türk alfabesi

Sesliler	**:** a e ı i o ö u ü
Sessizler	**:** b c ç d f g ğ h j k l m n p r s ş t v y z

Türkçede heceler ise en az bir harf en çok beş harften oluşmakla birlikte rastlanılması mümkün olan tüm hece yapıları ve bu yapılara ait örnek heceler Çizelge 3.2 de gösterilmiştir. Bu çizelgede C sessiz harfi, V ise sesli harfi temsil etmektedir.

Çizelge 3.2. Türkçe hece yapıları

Hece Yapıları	Örnek Heceler
V	a, e, ı, i, o, ö, u, ü
VC	ab, ac, aç, az, el, en, …
CV	ba, be, bı, za, le, ne, …
CVC	bel, gel, tam, son, …
VCC	alt, üst, ört, …
CCV	bre, pro, …
CVCC	kurt, sırt, renk, …
CCVC	tren, krem, flaş, …
CCVCC	tvist, trans, …

Türkçedeki hece yapıları, Türk Dil Kurumu'nun resmi internet sitesinde belirtildiği gibi iki kural ile tanımlanmaktadır (Anonim, 2015). Kuralların birincisinde, iki sesli harf arasında bulunan bir sessiz harfin kendinden sonraki sesli harf ile bir hece kuracağı belirtilmiştir. Bu kuralı tanımlamak için "-" hece ayırıcı sembol olmakla birlikte VCV→V-CV gösterimi kullanılabilir. Örnek olarak ise "araba" sözcüğü "a-ra-ba" olarak hecelenebilir.

Heceleme kurallarından ikincisinde ise bir sözcükte sessiz harflerin yan yana gelmesi durumu ele alınmıştır. Eğer iki sessiz harfin yan yana gelme durumu mevcut ise sessiz harflerden ilkinin kendinden önceki sesli harf ile, ikinci sessiz harfin ise kendinden sonraki sesli harf ile hece kuracağı söylenir. Eğer üç sessiz harf yan yana gelmişse ilk iki sessiz harf kendinden önceki sesli harf ile, üçüncüsü de kendinden sonraki sesli harf ile hece oluşturmaktadır. Örneğin sırasıyla "sesli" ve "harfli" sözcüklerini hecelerine ayırmak istersek "ses-li" ve "harf-li" şeklinde hecelenmiş olur. Bu kural ise sırasıyla $VC_1C_2V \rightarrow VC_1\text{-}C_2V$ ve $VC_1C_2C_3V \rightarrow VC_1C_2\text{-}C_3V$ gösterimi ile ifade edilebilir.

Ayrıca yabancı dillerden Türkçeye geçmiş sözcükler de Türkçe hece kurallarına göre hecelerine ayrılarak aykırı bir durum oluşturmamış olurlar. Örneğin "program", "santral", "sürpriz", "tundra" gibi sözcüklerin hecelenmiş biçimleri sırasıyla "prog-ram", "sant-ral", "sürp-riz", "tund-ra" şeklindedir.

Heceleme kurallarından yola çıkılarak, bir sesli harfin hece oluşturabilmesine rağmen sessiz harflerin sesli harf olmadan hece oluşturamadıkları görülmektedir. Buradan hareketle sözcüklerdeki sesli harf sayısının hece sayısını verdiği söylenebilir. Bu yüzden heceleme için sözcükteki sesli harflerin yerlerinin belirlenmesi önemlidir. Türkçede sekiz tane sesli harf bulunduğundan sesli harf dizininin belirlenmesi için her bir karakter üzerinde sekiz kez karşılaştırma işleminin yapılması gerekir.

Sözcüklerdeki sesli harflerin yerlerini doğru bir şekilde tespit edebilmek amacıyla, Türkçe için kullanılan farklı heceleme algoritmaları bulunmaktadır. Literatürde üç adet heceleme algoritması (Ucoluk ve Toroslu, 1997; Akın, 2005; Aşlıyan ve Günel, 2011) öne çıkmakla birlikte, çalışmada daha hızlı ve doğru sonuçlar veren Ek 2 de kodu sunulan RASAT Heceleme Algoritması (Aşlıyan ve Günel, 2011) kullanılmıştır.

3.4. Düzgün Deyimler

Tanım 3.1. Sonlu sembol dizisinden oluşan kümeye "alfabe" denir ve Σ sembolü ile gösterilir.

Tanım 3.2. Σ alfabesi verildiğinde bu alfabedeki sembollerden oluşabilecek tüm sözcüklerin kümesine "alfabenin kapanışı" veya "Kleene kapanışı" adı verilir ve Σ^* sembolü ile gösterilir (Kleene, 1956).

Tanım 3.3. Alfabenin hiçbir sembolünü içermeyen ve tek başına bir anlam ifade etmeyen sözcüğe "boş sözcük" denir ve Λ sembolü ile gösterilir.

Örnek 3.1. Σ = {x} ile gösterilen x sembolüne sahip bir alfabe alınmış olsun. Öyle ise Kleene kapanışı Σ^* = {Λ, x, xx, xxx, xxxx, …} kümesi ile ifade edilmektedir.

Örnek 3.2. Eğer alfabe Σ = {a,b} ise, o zaman Kleene kapanışı Σ^* = {Λ, a, b, aa, ab, ba, bb, baa, bab, …} olur.

Alfabe üzerinde tanımlanan Kleene kapanışı kümeler üzerine genişletilebilir. Örnek olarak S = {b, ab} ise, S^* = {Λ, b, ab, bab, abb, bb, abab, … } olur.

Bir metin içinde dili karakterize eden karakter, hece, sözcük gibi çeşitli sembol dizilimleri vardır. Bununla birlikte bu çalışmada, V sesli, C sessiz harfi temsil etmek üzere S = {xy | x ϵ C^* , y ϵ V∪{Λ, ξ}} formundaki düzgün deyimi tanımlı sembol dizilimi yapısı oluşturulmuştur ve bu formlardaki sembol dizilimlerinin sıkıştırma üzerindeki etkisi incelenmiştir. Ek olarak, bahsedilen formlardaki sembol dizilimlerinin en az ikisinin birleştirilmesiyle oluşturulan hibrit yapıların da sıkıştırmada kullanılıp kullanılamayacağı araştırılmıştır.

Örnek 3.3. S = {xy | x ϵ C^* , y ϵ V∪{Λ, ξ}} yapısı ve DNA'da bulunan dört baz olan adenin (A), sitozin (C), guanin (G) ve timin (T) bazları ile bunların bir dizilimi ele alınsın. S = {xy | x ϵ C^* , y ϵ V∪{Λ, ξ}} yapısı ile "gatctccatatacaacggtatctccacctcaggtttagatctccaacaacggaaccatag" baz diziliminin nasıl kodlanabileceği gösterilecektir. ξ sembolü dosya sonu simgesi olarak tanımlanmıştır ve $\xi \in V$ kabul edilir.

Baz dizilimini S yapısına göre ses birimlerine ayırmak istersek “ga – tctcca – ta – ta – ca – Λa – cggta – tctcca – cctca – ggttta – ga – tctcca – Λa – ca – Λa – cgga – Λa – cca – ta – g” elde edilir. Burada Λ sessiz harf olarak alınmıştır. Ayrıca son sembol olan “gξ” nin g ϵ S olabilmesi için ξ sembolü sesli harf kümesine de dahil edilmiştir.

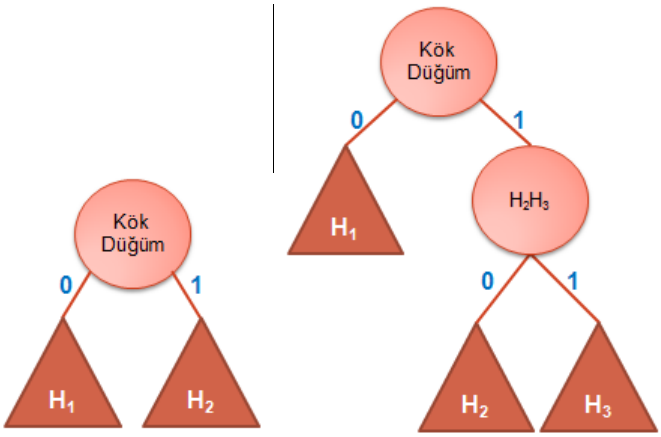
Ek 3 de verilen metni yukarıda tanımlanan haliyle düzgün deyimler cinsinden ayrıştıran program sunulmuştur.

4. BULGULAR

4.1. Belirteç Türlerine Göre Huffman Kodlaması

Tek türlü belirteç kullanılarak sıkıştırma gerçekleştirilecekse Şekil 2.12 ile verilen dengeli Huffman ağacı oluşturulur ve bu ağaç üzerinde kodlama gerçekleştirilir.

İki veya daha fazla belirteç türü kullanılarak hibrit sıkıştırma yapılmak istenirse, öncelikle belirteç türlerinden biri için Huffman ağacı (H_1) oluşturulur. Ardından bu ağacın derinliği (d) hesaplanır. İkinci belirteç türü için yeniden bir Huffman ağacı (H_2) oluşturulmalıdır. Ancak bu Huffman ağacının da bir önceki Huffman ağacıyla aynı derinliğe sahip olması beklenmektedir. Dolayısıyla ikinci ağacın en fazla 2^d düğüm içermesi gerektiği varsayılır ve ikinci ağaç oluşturulurken doküman içerisinde en sık görülen ilk 2^d belirteç dikkate alınır. Her belirtece karşılık gelen Huffman kod tablosu oluşturulurken öncelikle her ağaç için ayrı ayrı kod tabloları oluşturulur. Ardından Şekil 4.1 de örneklendiği üzere bu kodların başına belirteç türünü belirleyen ayrı bir kod eklenir. Hibrit Huffman ağacı olarak adlandırdığımız bu ağaç yapısında H_1 ağacı ilk belirteç türüne göre ve H_2 ağacı ise ikinci belirteç türüne göre oluşturulmuştur.



Şekil 4.1. Hibrit Huffman Ağaçları

Metin sıkıştırılırken önce metin birinci belirteç türüne göre parçalanır ve H_1 ağacı için oluşturulan kod tablosunda karşılık gelen kod bulunursa başına 0 kodu eklenir. Eğer kod tablosunda birinci belirteç türüne ait bir kod mevcut değilse bu durumda belirteç ikinci belirteç türüne göre tekrar parçalanır ve her parçanın kodu

H_2 ağacı için belirlenen kod tablosunda aranır. Bu kodun başına da Şekil 4.1 de görüleceği üzere 1 eklenerek belirtecin yeni kodu elde edilir. İki farklı Huffman ağacını birleştiren Matlab kodları Ek 4 de sunulmuştur.

Sıkıştırılan dosya açılırken ise öncelikle ilk bitin değerine göre hangi Huffman ağacında arama yapılacağı belirlenir. Geri kalan işlem standart Huffman kodlamasıyla aynıdır. Sıkıştırma işleminden sonra her bir belirtece karşılık gelen kod sözcükleri bulunmuş ve kod uzunlukları hesaplanmıştır.

4.2. Deneysel Çalışmalar

Bu çalışmada biri gen dizilimi, biri müzik notaları, üçü Türkçe ve ikisi İngilizce metin içeren yedi farklı doküman üzerinde sıkıştırma işlemi yapılmıştır. Deneyler sırasında öncelikle doküman içeriğinden yola çıkılarak Ek 5 ile verilen kodla seçilen belirteç türüne göre bir alfabe oluşturulmuştur. Ardından doküman içeriğinin karakter *n*-gram, hece *n*-gram ve S kümesinin elemanları olan belirteçler bazında frekansları çıkarılmıştır. Ardından her bir belirteç türü için ayrı ayrı Huffman kodlama algoritması uygulanarak sıkıştırma gerçekleştirilmiştir. Örnek oluşturması açısından hece 1-gram ve karakter 1-gram hibrit Hufman ağacı oluşturma ve sıkıştırma kodları Ek 6 da sunulmuştur.

Belirteç türüne göre sıkıştırma performanslarının karşılaştırılması gerçekleştirilmiştir ve elde edilen sonuçlar Çizelge 4.1 de sunulmuştur.

Çizelge 4.1. Farklı belirteç türlerine göre sıkıştırma kazançları

Belirteç türü	Dokümanlar						
	d_1	d_2	d_3	d_4	d_5	d_6	d_7
Karakter 1-gram	70,10	43,18	45,87	44,99	39,02	44,20	57,41
Karakter 2-gram	71,22	50,72	54,19	10,22	48,91	02,27	45,63
C*V	81,76	**79,70**	79,35	63,18	**77,23**	**84,18**	**84,89**
C*V + Karakter 1-gram	62,54	51,25	54,51	50,93	46,51	49,00	74,04
C*V + Karakter 2-gram	70,84	59,47	63,54	51,54	53,77	60,98	80,60
C*V + Karakter 3-gram	76,45	76,33	**82,45**	**69,85**	66,80	81,14	80,75
Karakter 2-gram + Karakter 1-gram	71,41	44,49	47,99	41,47	42,67	46,01	67,57
Karakter 3-gram + Karakter 1-gram	**84,84**	55,27	60,83	50,00	51,89	41,50	76,84
C*V + Karakter 2-gram + Karakter 1-gram	71,18	57,41	58,32	50,26	55,86	56,64	79,39
C*V + Karakter 3-gram + Karakter 1-gram	63,37	48,93	50,07	47,23	49,31	45,04	78,33
Hece 1-gram + Karakter 1-gram	66,61	46,27	42,54	34,72	30,81	48,97	65,29
Hece 2-gram + Karakter 1-gram	67,84	50,62	50,98	45,35	34,93	54,79	54,63
Hece 3-gram + Karakter 1-gram	66,79	41,68	53,45	41,88	28,22	45,47	55,14
Hece 1-gram + Karakter 2-gram + Karakter 1-gram	66,42	45,78	41,48	33,50	29,46	48,50	64,74
Hece 2-gram + Karakter 2-gram + Karakter 1-gram	74,62	64,79	64,95	57,58	55,77	67,14	73,38
C*V + Hece 1-gram + Karakter 1-gram	66,92	42,93	36,22	29,77	27,21	44,33	58,22
C*V + Hece 2-gram + Karakter 1-gram	64,14	40,12	34,89	28,69	24,76	41,00	58,22

5. TARTIŞMA VE SONUÇ

Bu tez çalışmasında, metin verisi içeren dosyaların mümkün olan en uygun boyutta saklanabilmesi için çalışmalar yapılmıştır. Sıkıştırma işlemini gerçekleştirmek amacıyla istatistiksel veri sıkıştırma yöntemlerinden olan Huffman sıkıştırma algoritması kullanılmıştır. Sıkıştırma işleminin performansında kullanılan yöntemin etkisinin yanında, farklı belirteç türleri kullanarak sıkıştırmanın iyileştirilebileceği gösterilmeye çalışılmıştır. Bu amaçla, metnin karakter ve hece bakımından *n*-gramlarının çıkarılmasıyla ve C sessiz, V sesli harfi temsil etmek üzere özel olarak belirlenmiş C^*V yapısına parçalanmasıyla oluşturulan farklı belirteçler ile sıkıştırma işlemleri gerçekleştirilmiştir. *n*-gramlar için 1-gram, 2-gram ve 3-gram olmak üzere ilk üç seviye kullanılmıştır. Bunun dışında belirtilen belirteçlerin birlikte kullanılmalarıyla oluşturulan hibrit yapılardan da yararlanılmıştır.

Sıkıştırma işlemleri yedi farklı doküman üzerinde 17 farklı belirteç türü ile gerçekleştirilmiş ve sıkıştırma kazançları hesaplanmıştır. Her bir belirteç türünün tüm dokümanlar üzerindeki sonuçları kullanılarak ortalama sıkıştırma kazançları ve standart sapma değerleri elde edilmiştir. Bu hesaplamalar ile birlikte, klasik Huffman yaklaşımı olan karakter 1-gram belirteç türüne ait sonuçlara göre sıkıştırma kazancının %49±10 olduğu gözlemlenmiştir. En iyi sıkıştırmanın ise %79±7 ile C^*V belirtecine ait olduğu görülmüştür. Buradan hareketle, herhangi bir metin dosyasının sıkıştırma oranının hecelemeden ve dolayısıyla dilden bağımsız olarak iyileştirilebileceğini söylemek mümkündür. Ayrıca çalışmada elde edilen en iyi sıkıştırma kazancına sahip ikinci yapı C^*V + karakter 3-gram ve üçüncü yapı Hece 2-gram + Karakter 2-gram + Karakter 1-gram hibrit yapıları olmuştur. Bu yapılara ait sıkıştırma kazançları ise sırasıyla %76±6 ve %66±7 olarak bulunmuştur. Bu sonuçlar ile hibrit yapıların kullanımıyla da kayda değer kazançların sağlanabileceği görülmüştür.

Elde edilen tüm bu sonuçların ışığında, farklı belirteç türleri ve yapıların kullanılması ile sıkıştırma performansının artırılabileceği söylenebilir. Ayrıca, bahsedilen yöntemin Huffman algoritması dışında diğer istatistiksel sıkıştırma metotlarına ve sözlük tabanlı yaklaşımlara da uygulanabileceği öngörülmektedir.

KAYNAKLAR

Abramson, N. 1963. Information Theory and Coding. McGraw-Hill, 201p, New York.

Akın, A.A., Akın, M.D. 2007. Zemberek, an open source NLP framework for Turkic Languages. Technical Report, [https://zemberek.googlecode.com/files/zemberek_makale.pdf], Erişim Tarihi: 23.07.2015.

Akkoyunlu, B. 1998. Eğitimde teknolojik gelişmeler. Çağdaş Eğitimde Yeni Teknolojiler (Özer, B.) Anadolu Üniversitesi, pp. 1-12, Türkiye.

Algan, S. (08.08.2004). Huffman veri sıkıştırma algoritması ve uygulaması [http://www.csharpnedir.com/articles/read/?id=189], Erişim Tarihi: 29.04.2015.

Anonim (21.04.2015). Vikipedi, Türk alfabesindeki harflerin kullanım sıklıkları [https://tr.wikipedia.org/wiki/Türk_alfabesindeki_harflerin_kullanım_sıklıkları], Erişim Tarihi: 22.06.2015.

Anonim (23.07.2015). Türk Dil Kurumu, Hece Yapısı ve Satır Sonunda Kelimelerin Bölünmesi [http://tdk.gov.tr/index.php?option = com_content & view = article & id = 208 : Hece-Yapisi-ve-Satir-Sonunda-Kelimelerin-Bolunmesi & catid = 50 : yazm-kurallar & Itemid = 132], Erişim Tarihi: 23.07.2015.

Aşlıyan, R., Günel, K. 2011. A comparison of syllabifying algorithms for Turkish. **Journal of Advanced Research in Computer Science**, 3(1): 58-78.

Cleary, J.G., Witten, I.H. 1984. Data compression using adaptive coding and partial string matching. **IEEE Transactions on Communications**, COM-32(4): 396-402.

Huffman, D.A. 1952. A method for the construction of minimum-redundancy codes. **Proceedings of the IRE**, 40(9): 1098-1101.

Nelson, M., Gailly, J. 1995. The Data Compression Book, 2nd ed., M&T Books, 541p, New York.

Salomon, D. 1997. Data Compression: The Complete Reference. Springer-Verlag, 427p, New York.

Salomon, D., Motta, G. 2010. Handbook of Data Compression, 5th ed., Springer, 1361p, London.

Shannon, C.E. 1948. A mathematical theory of communication. **The Bell System Technical Journal**, 27(3): 379-423.

Üçoluk, G., Toroslu, İ. H. 1997. A genetic algorithm approach for verification of the syllable based text compression technique. **Journal of Information Science**, 23(5): 365-372.

Zhuang, L., Bao, T., Zhu, X., Wang, C., Naoi, S. 2004. A Chinese OCR spelling check approach based on statistical language models. **IEEE International Conference on Systems, Man and Cybernetics**, Volume 5. (10-13 Oct. 2004), pp. 4727–4732.

Ziv, J., Lempel, A. 1977. A universal algorithm for sequential data compression. **IEEE Transactions on Information Theory**, IT-23(3): 337-343.

Ziv, J., Lempel, A. 1978. Compression of individual sequences via variable-rate coding. **IEEE Transactions on Information Theory**, IT-24(5): 530-536.

EKLER

Ek 1. Huffman ağacı ve ilgili Huffman kod tablosunu oluşturmak için kullanılan fonksiyon

```
function
[huffmann_agaci,HuffmanTablo,H,L]=huffmann_agaci_olustur(me
tin,mod)

alfabe=alfabe_olustur(metin,mod);
k=0;
for i=1:size(alfabe,2)
    frekans=size(strfind(metin,alfabe{i}),2);
    if (frekans>0)
        k=k+1;
       dugum(k).str=alfabe{i};
       dugum(k).frekans=frekans;
       dugum(k).sol={};
       dugum(k).sag={};
    end
end

sayac=0;
tic
while (size(dugum,2)>1)
    sayac=sayac+1;
    %En Küçük iki frekansa sahip düğümleri belirle
    if  (dugum(1).frekans<=dugum(2).frekans)
        min1=dugum(1).frekans; min1_indis=1;
        min2=dugum(2).frekans; min2_indis=2;
    else
        min1=dugum(2).frekans; min1_indis=2;
        min2=dugum(1).frekans; min2_indis=1;
    end
    for i=3:size(dugum,2)
        if (min1>dugum(i).frekans)
            min2=min1;min2_indis=min1_indis;
            min1=dugum(i).frekans; min1_indis=i;
        else
            if (min2>dugum(i).frekans)
                min2=dugum(i).frekans;min2_indis=i;
            end
        end
    end

    % Şimdi bu iki düğüm birleştiriliyor.

yeni_dugum=agac_birlestir(dugum(min1_indis),dugum(min2_indi
s));
    if (min1_indis<min2_indis)
        dugum=[dugum(1:min1_indis-1)
dugum(min1_indis+1:min2_indis-1)  dugum(min2_indis+1:end)
];
    else
```

```
        dugum=[dugum(1:min2_indis-1)
dugum(min2_indis+1:min1_indis-1)  dugum(min1_indis+1:end)
];
    end
    dugum=[yeni_dugum dugum];
end
huffmann_agaci=dugum;
huffmann_agaci.kod='';
huffmann_agaci=kodOlustur(huffmann_agaci);
disp('Huffmann agacı oluşturuldu.');
toc

disp('Kod tablosu oluşturuluyor.');
HuffmanTablo={};
HuffmanTablo=huffmanTablosuOlustur(HuffmanTablo,huffmann_ag
aci);
% Tabloyu frekansa göre azalan sırada sırala
for i=1:size(HuffmanTablo,2)-1
    for j=i+1:size(HuffmanTablo,2)
        if
(HuffmanTablo(i).frekans<HuffmanTablo(j).frekans)
            gecici = HuffmanTablo(i).frekans;
            HuffmanTablo(i).frekans =
HuffmanTablo(j).frekans;
            HuffmanTablo(j).frekans = gecici;
            gecici_karakter = HuffmanTablo(i).karakter;
            HuffmanTablo(i).karakter =
HuffmanTablo(j).karakter;
            HuffmanTablo(j).karakter = gecici_karakter;
            gecici_kod = HuffmanTablo(i).kod;
            HuffmanTablo(i).kod = HuffmanTablo(j).kod;
            HuffmanTablo(j).kod = gecici_kod;
        end
    end
end
disp('Huffmann Kod Tablosu');
disp('  i      Belirteç       Frekans      Huffman Kodu');
disp('-------------------------------------------------------
');
for i=1:size(HuffmanTablo,2)
    fprintf('%3d %12s %12d\t\t %s\n', i,
HuffmanTablo(i).karakter, HuffmanTablo(i).frekans,
HuffmanTablo(i).kod');
end

% Entropy ve ortalama kod uzunluğu hesaplanıyor
toplam_frekans=0;
for i=1:size(HuffmanTablo,2)

toplam_frekans=toplam_frekans+HuffmanTablo(i).frekans;
end
t1=0;
t2=0;
for i=1:size(HuffmanTablo,2)
```

```
t1=t1+HuffmanTablo(i).frekans*length(HuffmanTablo(i).kod);
    p(i)=HuffmanTablo(i).frekans/toplam_frekans;
    t2=t2+p(i)*log2(p(i));
end
H=-t2; % Entropy
L=t1/toplam_frekans; %Ortalama kod uzunluğu
fprintf('\n Ortalama kod uzunluğu = %8.4f bittir\n', L);
fprintf('\n Entropy = %8.4f \n',H);
save HuffmanTablo.mat HuffmanTablo;
```

```
function agac=kodOlustur(dugum)
  if ~yaprakmi(dugum.sol)
    dugum.sol.kod = [dugum.kod '0'];
    dugum.sol = kodOlustur(dugum.sol);
  else
    dugum.sol.kod = [dugum.kod '0'];
  end
  if ~yaprakmi(dugum.sag)
    dugum.sag.kod = [dugum.kod '1' ];
    dugum.sag = kodOlustur(dugum.sag);
  else
    dugum.sag.kod = [dugum.kod '1' ];
  end
  agac=dugum;
end
```

```
function HuffmanTablo =
huffmanTablosuOlustur(HuffmanTablo,dugum)
  if ~yaprakmi(dugum.sol)
    HuffmanTablo =
huffmanTablosuOlustur(HuffmanTablo,dugum.sol);
  else
    % sol düğümün kodunu tabloya ekle

HuffmanTablo(size(HuffmanTablo,2)+1).karakter=dugum
.sol.str;

HuffmanTablo(size(HuffmanTablo,2)).frekans=dugum.so
l.frekans;
    HuffmanTablo(size(HuffmanTablo,2)).kod =
dugum.sol.kod;
  end
  if ~yaprakmi(dugum.sag)
    HuffmanTablo =
huffmanTablosuOlustur(HuffmanTablo,dugum.sag);
  else
    % sag düğümün kodunu tabloya ekle

HuffmanTablo(size(HuffmanTablo,2)+1).karakter=dugum
.sag.str;

HuffmanTablo(size(HuffmanTablo,2)).frekans=dugum.sa
g.frekans;
    HuffmanTablo(size(HuffmanTablo,2)).kod =
dugum.sag.kod;
```

```
  end

end
```

```
1   function y=yaprakmi(dugum)
2      if ((isempty(dugum.sol)) &&
3   (isempty(dugum.sag)))
4           y=1;
5      else
6           y=0;
       end
```

Ek 2. Türkçe sözcüğü heceleyen Matlab fonksiyonu

```
function liste = hecele(w)

str=w;
str(~isstrprop(str,'alphanum')) = ' ';
str2=str;
sozcukler = textscan(str,'%s','Delimiter',' ','MultipleDelimsAsOne',1);
k=0;

for i=1:size(sozcukler{1},1)
    h=hecele_monogram(sozcukler{1}{i});
    for j=1:size(h,2)
        k=k+1;
        S1{k}=h{j};
    end
end

%Tekrarlanan verileri sil
S1=unique(S1);

% Frekanslar hesaplanıyor
for i=1:size(S1,2)
    liste.s{i} = S1{i};
    liste.f(i) = size(strfind(str2,S1{i}),2);
end

% Frekansa göre küçükten büyüğe sıralanıyor.
% Sıralama
for i=1:size(liste.s,2)-1
    for j=i+1:size(liste.s,2)
        if (liste.f(i)>liste.f(j))
            gecici = liste.f(i);
            liste.f(i) = liste.f(j);
            liste.f(j) = gecici;

            gecici2 = liste.s{i};
            liste.s{i} = liste.s{j};
            liste.s{j} = gecici2;
        end
    end
end
end %function
```

```
function hece = hecele_monogram(w)
% w sözcüğünü hecelerine ayıran Matlab fonksiyonudur.
  unluIndis = regexp(w,'[AaEeIıİiOoÖöUuÜü]');
  heceSay = 0;
  while size(unluIndis,2)>1
      % iki sesli harf arasındaki karakter sayısını belirle
      sessizSay = unluIndis(2) - unluIndis(1) - 1;
      heceSay = heceSay + 1;

      if(sessizSay==0)
          hece{heceSay} = w(1:unluIndis(1));
          w = w(unluIndis(1)+1:end);
      else
          hece{heceSay} = w(1:unluIndis(1)+sessizSay-1);
          w = w(unluIndis(1)+sessizSay:end);
      end

      unluIndis = regexp(w,'[AaEeIıİiOoÖöUuÜü]');
  end
  hece{heceSay+1} = w(1:end);
end %function
```

Ek 3. C*V düzgün deyim parçalanışını üreten Matlab fonksiyonu

```
function liste = C_starV(str)
indis = regexp(str,'[aeıioöuüAEIİOÖUÜ]');

S1{1}=str(1:indis(1)); % ilk karakter dizisi
k=1;
for i=1:length(indis)-1
    if length(str(indis(i)+1:indis(i+1)))>1
        k=k+1;
        S1{k}=str(indis(i)+1:indis(i+1));
    end
end
if (indis(end)<length(str))
    if (length(str(indis(end)+1:end))>1)
        S1{k+1}=str(indis(end)+1:end);
    end
end

% Frekanslar hesaplanıyor
for i=1:size(S1,2)
    liste.s{i} = S1{i};
    liste.f(i) = size(strfind(str,S1{i}),2);
end

% Frekansa göre küçükten büyüğe sıralanıyor.
for i=1:size(liste.s,2)-1
    for j=i+1:size(liste.s,2)
        if (liste.f(i)>liste.f(j))
            gecici = liste.f(i);
            liste.f(i) = liste.f(j);
            liste.f(j) = gecici;

            gecici2 = liste.s{i};
            liste.s{i} = liste.s{j};
            liste.s{j} = gecici2;
        end
    end
end
end
```

Ek 4. İki Huffman ağacını birleştirerek yeni bir Huffman ağacı oluşturan Matlab fonksiyonu

```
function huffmann=agac_birlestir(agac1,agac2)
% Bu fonksiyon ağacı birleştirerek bir Huffmann  ağacı
oluşturur
dugum.frekans=agac1.frekans+agac2.frekans;
if (agac1.frekans>=agac2.frekans)
    dugum.str=[agac1.str agac2.str] ;
    dugum.sol=agac1;
    dugum.sag=agac2;
else
    dugum.str=[agac2.str agac1.str] ;
    dugum.sol=agac2;
    dugum.sag=agac1;
end

huffmann=dugum;
```

Ek 5. Belirteç türüne göre alfabenin çıkarılması

```
function alfabe=alfabe_olustur(metin,mod)
k=0;
if strcmp(mod,'karakter_monogram')
    for i=0:255
        karakter_frekans=size(strfind(metin,char(i)),2);
        if (karakter_frekans>0)
            k=k+1;
            alfabe{k}=char(i);
        end
    end

elseif strcmp(mod,'karakter_bigram')
    while (length(metin)>=2)
        k=k+1;
        alfabe{k}=metin(1:2);
        metin=metin(3:end);
    end
    if (length(metin)>0)
        alfabe{k+1} = metin(1:end);
    end

elseif strcmp(mod,'karakter_trigram')
    while (length(metin)>=3)
        k=k+1;
        alfabe{k}=metin(1:3);
        metin=metin(4:end);
    end
    if (length(metin)>0)
        alfabe{k+1} = metin(1:end);
    end

elseif strcmp(mod,'hece_monogram')

 str=metin;
 str(~isstrprop(str,'alphanum')) = ' ';
 sozcukler = textscan(str,'%s','Delimiter','
',
'MultipleDelimsAsOne',1);

 for i=1:size(sozcukler{1},1)
        h=hecele_monogram(sozcukler{1}{i});
        for j=1:size(h,2)
            disp(h{j});
```

```
43              k=k+1;
44              alfabe{k}=h{j};
45          end
46      end
47
48  elseif strcmp(mod,'hece_bigram')
49
50      str=metin;
51      str(~isstrprop(str,'alphanum')) = ' ';
52      sozcukler = textscan(str,'%s','Delimiter','
53  ','MultipleDelimsAsOne',1);
54
55      for i=1:size(sozcukler{1},1)
56
57          h=hecele_bigram(sozcukler{1}{i});
58          boyut=size(h,2);
59
60          if (boyut>1)
61              for j=1:boyut
62                  disp(h{j});
63                  k=k+1;
64                  alfabe{k}=h{j};
65              end
66
67          else
68              unluIndis = regexp(h{1},'[AaEeIıİiOoÖöUuÜü]');
69              if(size(unluIndis,2)>1)
70                  disp(h{1});
71                  k=k+1;
72                  alfabe{k}=h{1};
73              else
74                  continue;
75              end
76          end
77      end
78
79  elseif strcmp(mod,'hece_trigram')
80
81      str=metin;
82      str(~isstrprop(str,'alphanum')) = ' ';
83      sozcukler = textscan(str,'%s','Delimiter','
84  ','MultipleDelimsAsOne',1);
85
86      for i=1:size(sozcukler{1},1)
```

```
 87
 88          h=hecele_trigram(sozcukler{1}{i});
 89          boyut=size(h,2);
 90
 91          if (boyut>1)
 92              for j=1:boyut
 93                  disp(h{j});
 94                  k=k+1;
 95                  alfabe{k}=h{j};
 96              end
 97
 98          else
 99              unluIndis = regexp(h{1},'[AaEeIıİiOoÖöUuÜü]');
100              if(size(unluIndis,2)>2)
101                  disp(h{1});
102                  k=k+1;
103                  alfabe{k}=h{1};
104              else
105                  continue;
106              end
107          end
108      end
109
110  elseif strcmp(mod,'C*V')
111      unlu='aeıioöuüAEIİOÖUÜ';
112      unlu_index=[];
113      for i=1:length(unlu)
114          unlu_index=[unlu_index strfind(metin,unlu(i))];
115      end
116      % Boşluk ve paragraf unlu gibi düşünülürse
117      %unlu_index=[unlu_index strfind(metin,'\n')];
118      %unlu_index=[unlu_index strfind(metin,' ')];
119
120      unlu_index=sort(unlu_index);
121
122      if ((length(metin(1:unlu_index(1)))>1)
123  &&(size(strfind(metin,metin(1:unlu_index(1))),2)>0))
124          alfabe{1}=metin(1:unlu_index(1));
125          k=1;
126      else
127          k=0;
128      end
129      for i=1:size(unlu_index,2)-1
130          str=metin(unlu_index(i)+1:unlu_index(i+1));
```

```
131          if ((length(str)>1)
132   &&(size(strfind(metin,str),2)>0))
133              k=k+1;
134              alfabe{k}=str;
135          end
136      end
137
138   else
139      alfabe=[];
140
      end

      %Tekrarlanan verileri sil
      alfabe=unique(alfabe);
```

Ek 6. Hece 1-gram + karakter 1-gram için sıkıştırma işlemini gerçekleştiren program

```
clear all
clc;

dosya_adi='haber';
dosya_uzantisi='txt';
fid=fopen([dosya_adi '.'
dosya_uzantisi],'r','native','windows-1254');

metin=fscanf(fid,'%c',inf);
fclose(fid);

mod='hece_monogram';
[huffman_agaci_1,Huffman_Tablo1,H1,L1]=huffman_agaci_olustu
r(metin,mod);

mod='karakter_monogram';
[huffman_agaci_2,Huffman_Tablo2,H2,L2]=huffman_agaci_olustu
r(metin,mod);

kodlanmis_metin='';

%Metin ayrıştırılıyor
A = isstrprop(metin, 'alpha');
indis = strfind(A,0);

while ~isempty(indis)
    if indis(1)>1
        w = metin(1:indis(1)-1);
    else
        w = metin(indis(1));
    end
    h = hecele_monogram(w);

    % h dizisinin elemanları Huffman_Tablo1 tablosunda var
mı?
    % Varsa tabloda yer alan kodun başına 0 eklenir
    % Yoksa hecenin karakterleri Huffman_Tablo2'de aranır

    for i=1:size(h,2)
     bulundu=0;
     str=h{i};
```

```
42      for j=1:size(Huffman_Tablo1,2)
43        if strcmp(str,Huffman_Tablo1(j).karakter)==1
44          kodlanmis_metin = [kodlanmis_metin '0'
45  Huffman_Tablo1(j).kod];
46          bulundu=1;
47          break;
48        end
49      end
50
51         if (bulundu==0)
52             str2=str(1);
53
54             for j=1:size(Huffman_Tablo2,2)
55                 if
56  (strcmp(str2,Huffman_Tablo2(j).karakter))
57                     bulundu=1;
58                     break;
59                 end
60             end
61
62             if (bulundu==1),
63                 kodlanmis_metin=[kodlanmis_metin '1'
64  Huffman_Tablo2(j).kod];
65             end
66
67             for i=2:size((str),2)
68                 str2=str(i);
69                 bulundu=0;
70                 for j=1:size(Huffman_Tablo2,2) %
71  Tablo1'deki veri sayısının yarısı kadar Tablo1 üzerinde
72  araştır.
73                     if
74  (strcmp(str2,Huffman_Tablo2(j).karakter))
75                         bulundu=1;
76                         break;
77                     end
78                 end
79                 if (bulundu==1),
80                     kodlanmis_metin=[kodlanmis_metin '1'
81  Huffman_Tablo2(j).kod];
82                 end
83             end
84         end
85     end
```

```
    % Takip eden noktalama işareti  (metin(indis(1))
Huffman_Tablo2 tablosunda var mı?
    % Kodun başına 1 ekle
    for j=1:size(Huffman_Tablo2,2)
        if
(strcmp((metin(indis(1))),Huffman_Tablo2(j).karakter))
            kodlanmis_metin=[kodlanmis_metin '1'
Huffman_Tablo2(j).kod];
        end
    end

    % metni güncelle
    metin = metin(indis(1)+1:end);
    A = isstrprop(metin, 'alpha');
    indis = strfind(A,0);

end

% Kodlama
eklenecek_bit_sayisi=8-rem(size(kodlanmis_metin,2),8);
for i=1:eklenecek_bit_sayisi
    kodlanmis_metin=['0' kodlanmis_metin];
end

for i=1:length(kodlanmis_metin)/8
    yeni_metin(i)=bin2dec(kodlanmis_metin((i-1)*8+1:i*8));
end

fid=fopen([dosya_adi
'_Hece_monogram_Karakter_monogram.huff'],'w','native','wind
ows-1254');
fprintf(fid,'%c',eklenecek_bit_sayisi);

for i=1:size(yeni_metin,2)
    fprintf(fid,'%c',yeni_metin(i));
end
fclose(fid);

disp('Hece_monogram + Karakter_monogram tabanlı sıkıştırma
tamamlandı.');
```

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Onur DİNCEL

Doğum Yeri ve Tarihi : İzmir, 24.10.1988

EĞİTİM DURUMU

Lisans Öğrenimi : Adnan Menderes Üniversitesi, Fen-Edebiyat Fakültesi, Matematik Bölümü

Yüksek Lisans Öğrenimi: Adnan Menderes Üniversitesi, Fen Bilimleri Enstitüsü, Matematik A.B.D.

Yabancı Diller : İngilizce

BİLİMSEL FAALİYETLERİ

A) Yayınlar

-Uluslararası :

Ulvi I., Asliyan R., Gunel K., Dincel O., Cagrici A. 2013. Textile Image Classification Using Artificial Neural Networks, Global Journal on Technology, Vol. 4, pp.615-620.

B) Bildiriler

-Uluslararası :

Ulvi I., Asliyan R., Gunel K., Dincel O., Cagrici A. 2013. Textile Image Classification Using Artificial Neural Networks, 3rd World Conference on Innovation and Computer Science (INSODE-2013).

-Ulusal :

Günel K., Dincel O. 2016. Belirteç Seçiminin Huffman Kodlaması Üzerine Etkisi, Akademik Bilişim 2016.

İLETİŞİM

E-Posta Adresi : onurdincel@hotmail.com

Tarih : 28/06/2016