KESIDAL: SIKIŞTIRMA ORANINI ARTTIRAN KELİME TABANLI METİN DÖNÜŞTÜRME ALGORİTMASI

Aydın CARUS[1], Altan MESUT[2]
[1,2]Trakya Üniversitesi, Mühendislik-Mimarlık Fakültesi, Bilgisayar Mühendisliği Bölümü
[1]aydinc@trakya.edu.tr, [2]altanmesut@trakya.edu.tr

ABSTRACT

A new word-based natural language text transform algorithm that improves compression ratio of compression algorithms is presented in this paper. The algorithm, named as KESIDAL, uses word-based static dictionaries. The main criteria while selecting the words for these dictionaries is the gain of these words in compression ratio. It was seen from our experiments that KESIDAL can also compress the natural language text files approximately to 2/3 of their sizes. It was also seen that if these transformed files are later compressed with other compression algorithms, the obtained results are 0.4% to 11.1% better than original compression ratios of these algorithms.

Key words: Word-Based Text Transform, Text Compression, Static Dictionary.

1. GİRİŞ

Genellikle, veri sıkıştırmada kullanılan sözlük tabanlı kayıpsız sıkıştırma yöntemleri, LZ ailesinde [1, 2, 3, 4, 5] olduğu gibi metin içindeki karakter kümelerine daha kısa bir kod atayıp onu sözlüğe ekleyerek, sonradan bu karakter kümesi yerine kodunun yazılması esasına göre sıkıştırma işlemi yapmaktadır. Bu sözlükler sıkıştırma aşamasında dinamik olarak oluşturulabildiği gibi sözlüklerin önceden belirlenip sıkıştırma aşamasında kullanılması da mümkündür. Benzer şekilde karakter kümesi yerine kelime tabanlı sıkıştırma yapan yaklaşımlar da sunulmuştur [6]. Ayrıca sıkıştırılacak metnin mevcut sıkıştırılma algoritmaları tarafından daha iyi sıkıştırılmasını sağlayacak kelime tabanlı dönüşümler yapan bazı çalışmalar da yapılmıştır [7, 8]. Kelime tabanlı dönüştürme işleminden sonra sıkıştırma yapan çalışmalar incelendiğinde dil bağımlı olanların daha yüksek performansa sahip olduğu görülmektedir [7].

Bu çalışmada sabit kelime sözlükleri kullanarak sözlükte mevcut olan bir kelimeyi 2 bayt olarak kodlayan bir yaklaşım verilmiştir. Kodlanan 2 baytın ilk baytı sözlükte kelimenin bloğunu ikinci bayt ise blok içinde kelimenin sırasını temsil etmektedir. Sözlüklerin sabit olmasından dolayı sözlük blok sayılarının ve sözlükte yer alacak kelimelerin belirlenmesinde, kelimelerin o dilde yazılan dokümanlarda yer alma sıklığı göz önünde bulundurularak sabit sözlükler oluşturulmuştur. İngilizce dili için oluşturulan sözlükler kullanılarak yapılan sıkıştırma denemelerinde, kelimelerin 2 baytlık kalıba dönüştürülürken ortalama %33 oranında sıkıştırılabildiği de görülmüştür. KESIDAL algoritması ile sıkıştırılmış doğal dil metin dosyalarının, daha sonra mevcut bazı sıkıştırma algoritmalarıyla sıkıştırıldığında, mevcut sıkıştırma algoritmalarının sıkıştırma oranlarını %11 kadar arttırabildiği görülmüştür.

Takip eden bölümde sabit sözlüklerin oluşturulması ve KESIDAL algoritması açıklanmış, daha sonraki bölümlerde ise kullanılan sıkıştırma algoritmalarının özet bilgileri verilerek, bu algoritmalar ile yapılan sıkıştırma denemelerinin sonuçları verilmiştir.

2. KESIDAL ALGORİTMASI

2.1. Sözlüğün Oluşturulması

Sözlük oluşturulmasında 173MB boyutunda İngilizce bir derlem kullanılmıştır. Kullanılan derlemin içinde yer alan dokümanlar internet ortamından elde edilmiş roman, hikâye, gazete makaleleri, bilimsel makaleler gibi çeşitli konulara ait belgelerdir.

Sözlüğü oluşturmanın ilk aşamasında derlemin tüm kelimeleri harf sayılarına göre gruplandırılır. Daha sonra doğal dildeki kullanım miktarlarına göre büyükten küçüğe doğru sıralanır. Sözlükte kullanılacak olan kelime sayıları belirlendikten sonra, hangi kelimelerin sözlüğe dâhil edileceğinin belirlenmesinde bu sıralama kullanılmaktadır. Kelimelerin farklı ekler almış halleri farklı kelime olarak ele alınmaktadır.

Sunulan algoritmada sözlük içinde kelimeler bloklar halinde saklanmaktadır. Bloğu temsil etmek için 1 bayt yer tahsisi yapılmaktadır. Diğer bir deyişle sözlükteki her bir kelime 256 farklı bloğun birine dâhil edilmektedir. Ancak, aranılan kelimenin sözlükte bulunmadığını belirtmek için de bir bloğun ayrılması gerekmektedir. Yani sözlükte 0-254 arası bloklara kelimeler yerleştirilirken, 255. blok ise kelimenin sözlükte yer almadığını göstermek için

kullanılmaktadır. Her bloktaki kelimelerin yerini belirtmek için ise ikinci bir bayt daha kullanılmaktadır. Bu nedenle her bir blokta 256 farklı kelime yer alabilmektedir. Bir başka deyişle sözlükte toplam olarak 255 × 256 = 65280 farklı kelime bulunmaktadır.

Sözlük bloklarının her kelime uzunluğuna eşit sayıda atanmasının sıkıştırma oranını azalttığının görülmesi sebebiyle, hangi kelime uzunluğuna kaç adet bloğun tahsis edileceği aşağıdaki formül kullanılarak belirlenmiştir:

$$Blok\ sayısı = (uzunluk - maliyet) \times oran \times 256 \quad (1)$$

Yukarıdaki formülde; *uzunluk*, kelimenin karakter sayısını; *maliyet*, kelime kodlandığında o kelimeyi ifade etmek için kullanılması gereken bayt sayısını (2 bayt); *oran* ise, derlem içinde aynı sayıda harfe sahip kelimelerin toplam sayısının tüm derlemdeki kelime sayısına oranını göstermektedir. Bu değerler 256 ile çarpılarak o kelime grubuna ayrılması gereken blok sayısı bulunmaktadır.

Kelime Uzunluğu	Blok Sayısı
1	1
2	1
3	2
4	10
5	17
6	25
7	29
8	29
9	25
10	23
11	21
12	18
13	15
14	13
15	10
16	8
17	5
18	2
19	1

Tablo 1. Kelime uzunluğuna göre blok sayıları

Kullanılan derlemden elde edilen kullanım sıklığına göre sıralanmış kelimelerin hangilerinin sözlükte yer alacağı Tablo 1'de verilen blok dağılımlarına göre belirlenmiştir. Kelimelerin uzunlukları belirlenirken kelime sonundaki noktalama karakterleri kelimeye dâhil edilerek kelimeler belirlenmiştir. Bloklara ayrılmış şekilde sözlükte saklanan kelimelerin sıkıştırılma aşamasında etkin olarak aranabilmesi için blok içinde kelimeler alfabetik sıralı olarak saklanmaktadır. Bu sayede sözlük blokları içerisinde ikili arama yapılabilmektedir.

Ayrıca sözlük oluşturulduktan sonra ikinci bir tabloya daha gereksinim duyulmaktadır. *Bloklar Tablosu* olarak adlandırılan bu tablo, hangi bloklarda hangi kelime uzunluğuna sahip kelimelerin saklandığının belirlenebilmesi için gereklidir.

2.2. Algoritmanın Çalışma Prensibi

KESIDAL dönüştürme algoritması sözlükte bulunan bir kelimeyi 2 bayt kullanarak sıkıştırmaktadır. Kullanılan ilk bayt kelimenin sözlükte yer aldığı blok numarasını, ikinci bayt ise o blok içinde kelimenin yerini belirtmektedir.

KESIDAL algoritması iki boşluk karakteri arasındaki tüm karakterleri bir kelime olarak kabul eder (varsa kelimeyi takip eden noktalama işareti de kelimeye dâhildir). Harf sayısı hesaplandıktan sonra algoritma o kelimenin sözlükte mevcut olup olmadığını bulmak için o kelimenin harf sayısına ayrılmış bloklarına konumlanmaktadır. Bu konumlanma için yukarıda belirtilen *Bloklar Tablosu* kullanılmaktadır. Algoritma bu tablodan harf sayısı-blok sınırı karşılığını bulur ve o bloklar arasına konumlanır. Bu bloklar ikili arama yöntemiyle taranır. Eğer kelimeye rastlanırsa dosyaya öncelikle bulunduğu blok numarası yazılır. İkinci olarak kelimenin o bloktaki sırası yazılır. Dolayısıyla kelime gerçekte kaç bayt yer kaplarsa kaplasın eğer sözlükte yer alıyorsa dönüştürme işlemi sonunda sadece iki bayt yer kaplayacaktır. Bu nedenle bu dönüştürme işlemi aynı zamanda sıkıştırma da yapmaktadır.

Eğer aradığımız kelime sözlük içerisinde yer almıyorsa, kelimenin sözlükte yer almadığını belirtmek için 255 değerine sahip bir baytlık bir bilgi PPM [9] algoritmasındakine benzer olarak kaçış karakteri gibi kullanılmaktadır. Sözlükte yer almayan her kelimenin önüne bu karakterin eklenmesiyle, dönüştürülemeyen kelimeler için bir bayt ekstra maliyet söz konusu olmaktadır.

3. KULLANILAN SIKIŞTIRMA ALGORİTMALARI

3.1. LZ77 Algoritması

Abraham Lempel ve Jakob Ziv tarafından geliştirilen ve 1977 yılında yayınladıkları makalelerinde [1] tanımladıkları bu yöntemde, sıkıştırılmak istenilen veri daha önce kodlanan veri içinde aranır, bulunursa kaç bayt geriye gidilince bulunduğu ve kaç baytlık kesimin bire bir eşit

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

3.2. LZW Algoritması

Lempel ve Ziv 1978 yılında yayınladıkları bir başka makalelerinde [2] kodlanan verilerin aynı zamanda bir sözlüğe eklendiği, daha sonra kodlanacak olan verilerin bu sözlükte arandığı, bulundukları takdirde sadece sözlükteki sıralarının kodlandığı bir yöntemi sunmuşlardır. LZ78 olarak bilinen bu algoritma daha sonra Terry Welch tarafından geliştirilmiş ve ortaya çıkan yeni algoritma LZW olarak kabul görmüştür [3]. LZW algoritması sadece metin tipinde veriler üzerinde değil, tüm veriler üzerinde iyi bir sıkıştırma oranına sahiptir.

3.3. LZRW1 Algoritması

Ross N. Williams tarafından 1991 yılında geliştirilen LZRW1 algoritması [4] LZ77 ile aynı prensipte çalışmaktadır. Fakat LZ77'den farklı olarak, sıkıştırma oranından bir miktar feragat edip sıkıştırma ve açma işlemlerini hızlandıran basit bir hash tablosu mekanizması kullanmaktadır. LZRW2 ve LZRW3 algoritmaları LZRW1 algoritmasından daha fazla sıkıştırma oranı sağlamakta, fakat daha yavaş çalışmaktadırlar.

3.4. DEFLATE Algoritması

Phil Katz tarafından PKZIP sıkıştırma programının ikinci sürümünde kullanılmak üzere geliştirilen DEFLATE algoritması [5] aslında LZ77 algoritması ile Huffman kodlamasının [10] birleşimidir. LZW algoritmasının tersine, patent koruması altında olmaması Gzip, LZOP gibi sıkıştırma programlarında ve PNG görüntü dosyalarında kullanılmasını sağlamıştır. Günümüzde çok kullanılan sıkıştırma araçlarından biri olan WinZip te DEFLATE algoritmasını kullanmaktadır.

3.5. LZOP Algoritması

LZOP algoritması Markus F.X.J. Oberhumer tarafından 1997 yılında geliştirilmeye başlanmış ve ilerleyen yıllarda birçok sürümü yayınlanmıştır. DEFLATE benzeri olan bu algoritma, yine Oberhumer tarafından geliştirilmiş olan LZO veri sıkıştırma kütüphanesini kullanmaktadır. ANSI C'de yazılmış olan bu kütüphane oldukça hızlı sıkıştırma ve çok hızlı açma sağlamaktadır.

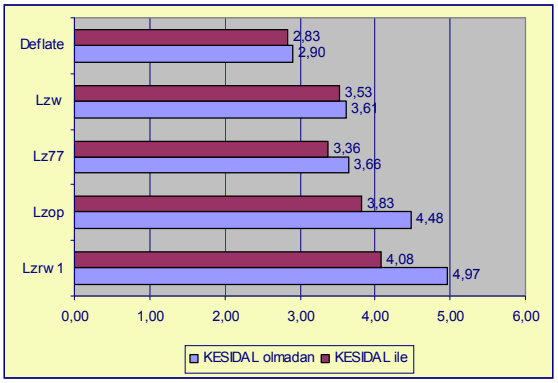
4. SIKIŞTIRMA DENEMELERİ VE SONUÇLAR

Dönüştürme algoritmasının uygulaması C programlama dili ile geliştirilmiştir. Geliştirilen uygulamayı denemek için Calgary Corpus [11], Canterbury Corpus [12], Silesia Corpus [13] gibi bilinen derlemlerden ve Gutenberg Projesinden [14] metin dosyaları ve tarafımızdan oluşturulan İngilizce metin dosyası kullanılmıştır. Bu dönüştürme denemelerinde dosyaların ne kadar sıkıştırıldığı karakter başına bit (bpc) biçiminde Tablo 2'de verilmiştir.

Dosya ismi	Dosya Uzunluğu (bayt)	Sıkıştırma Oranı (bpc)
book1[11]	768.771	5,28
book2[11]	610.856	5,60
paper1[11]	53.161	5,94
paper2[11]	82.199	4,89
Bible[12]	4.047.392	4,58
Asyoulik[12]	125.179	6,03
alice29.txt[12]	152.089	5,74
icet10.txt[12]	426.754	5,15
plrabn12.txt[12]	481.861	5,18
1musk12.txt[13]	1.349.141	5,35
anne11[13]	587.053	5,19
dickens.txt[14]	10.192.446	5,05
ingilizce.txt	16.453.299	4,98
	Ortalama	**5,31**

Tablo 2. Dönüştürmenin sıkıştırma oranları

Tablo 2'de görüldüğü gibi, KESIDAL algoritması kendi başına kullanıldığında metin dosyalarının büyüklüğünü 2/3 oranına indirgeyecek kadar sıkıştırma sağlayabilmektedir. Dönüştürme işlemlerinin her dosya için yapılmasından sonra elde edilen kısmen sıkıştırılmış dosyalar Bölüm 3'te verilen sıkıştırma algoritmalarının programları ile tekrar sıkıştırılmıştır. Bu programlardan LZ77 algoritması için olanı Rich Geldreich tarafından, LZW algoritması için olanı Mark Nelson tarafından, LZRW1 algoritması için olanı ise Ross N. Williams tarafından geliştirilmiştir. LZOP algoritması için LZOP 1.02, DEFLATE için ise Gzip 1.2.4 sürümleri kullanılmıştır. Bu programların tek başlarına kullanılması ile elde edilen sıkıştırma oranları ve KESIDAL algoritmasından sonra kullanılması ile elde edilen sıkıştırma oranları Şekil 1'de verilmiştir.



Şekil 1. Sıkıştırma oranları (bpc)

KESIDAL ile dönüştürülmüş dosyaların verilen sıkıştırma algoritmaları ile sıkıştırıldığı durumda, bu algoritmaların kendi başlarına kullanıldığı duruma göre daha yüksek sıkıştırma oranlarına ulaşılmıştır. Sıkıştırma yüzdelerine göre hesaplanırsa KESIDAL kullanılmasındaki kazanç oranları: LZRW1 algoritmasında %11,1; LZOP algoritmasında %8,1; LZ77 algoritmasında %3,7; LZW algoritmasında %1 ve DEFLATE algoritmasında %0,9 olmuştur.

5. SONUÇLAR VE TARTIŞMA

Veri sıkıştırma algoritmaları temel olarak iki amaç için kullanılmaktadır: Verinin büyüklüğünü azaltarak saklama kapasitesini arttırmak ve bilgisayar ağlarında verilerin daha hızlı gönderilebilmesini sağlamak. KESIDAL dönüştürme algoritmasının ihtiyacı olan ekstra zaman nedeniyle veri iletişimde sıkıştırma algoritmaları ile birlikte kullanılmasının kazanç sağlamayacağı aşikârdır. Ancak sıkıştırma algoritmalarının saklama birimlerinde yer kazanılmasına yönelik kullanıldığı durumlarda, sunulan algoritmanın sıkıştırma algoritmalarının önünde bir dönüşüm algoritması gibi kullanıldığında sıkıştırma oranını arttırabildiği görüldüğü için, bu algoritmanın kullanım alanı bulabileceği düşünülmektedir.

KAYNAKLAR

[1] Ziv, J., Lempel, A., "A Universal Algorithm for Sequential Data Compression", *IEEE Transactions on Information Theory*, IT-23(3):337-343, 1977.

[2] Ziv, J., Lempel, A., "Compression of Individual Sequences via Variable-Rate Coding", *IEEE Transactions on Information Theory*, IT-24(5):530-536, 1978.

[3] Welch, T. A., "A Technique for High-Performance Data Compression", *IEEE Computer*, 17(6):8-19, 1984.

[4] Williams, R. N., "An Extremely Fast Ziv-Lempel Data Compression Algorithm", *Proceedings of IEEE Data Compression Conference*, Snowbird, Utah, 362-371, 1991.

[5] Deutsch, P., "DEFLATE Compressed Data Format Specification, version 1.3", *Network Working Group*, *Request for Comments 1951*, 1996.

[6] Çelikel, E., Dalkılıç, M. E., Dalkılıç, G., "Word-Based Fixed and Flexible List Compression", *Int'l Symposium on Computer and Information Sciences (ISCIS) XXth*, Istanbul, Turkey, 2005.

[7] Skibiński, P., Grabowski, Sz., Deorowicz, S., "Revisiting dictionary-based compression", *Software - Practice & Experience*, 35(15):1455-1476, 2005.

[8] Awan, F. S., Zhang, N., Motgi, N., Iqbal, R. T., Mukherjee, A., "LIPT: A Reversible Lossless Text Transform to Improve Compression Performance", *Proceedings of IEEE Data Compression Conference*, Snowbird, Utah, 481, 2001.

[9] Cleary, J. G., Witten, I. H., "Data Compression Using Adaptive Coding and Partial String Matching", *IEEE Transactions on Communications*, 32(4):396-402, 1984.

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

[11] Witten, I. H., Bell, T., "The Calgary text compression corpus", *Anonymous ftp*, ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/

[12] Arnold, R., Bell, T., "A Corpus for the Evaluation of Lossless Compression Algorithms", *Proceedings of IEEE Data Compression Conference*, Snowbird, Utah, 1997.

[13] http://sun.iinf.polsl.gliwice.pl/~sdeor/corpus.htm

[14] http://www.gutenberg.org