Terim-Doküman Matrisleri için Sıralamaya Dayalı Bir Kayıpsız Sıkıştırma Şeması

Reordering Based Lossless Compression Scheme for Term-Document Matrices

Can ÖZBEY
IDEA Teknoloji Çözümleri
Ar-Ge Mühendisi
İstanbul, TÜRKİYE
can.ozbey@ideateknoloji.com.tr

Murat Cihan SORKUN
IDEA Teknoloji Çözümleri
Ar-Ge Mühendisi
Istanbul, TÜRKİYE
murat.sorkun@ideateknoloji.com.tr

Öz

Kayıpsız veri sıkıştırma, özellikle bellek içi veri tabanları ve önbellek kullanımlı bilgi geri kazanım sistemlerinde, harcanan disk alanını azaltmasının yanı sıra, etkin kod çözme algoritmaları aracılığıyla bilgiye erişimi hızlandırması sebebiyle önem arz etmektedir. Bu çalışmada, ters dizinlerin sıkıştırılması kapsamında yeni bir değişken sekiz ikili kodlama yöntemi geliştirilmiş ve terim-doküman matrisinin bant genişliğinin indirgenmesi amacıyla tepe tırmanmaya dayalı çift kutuplu dizilim şeması önerilmiştir. Bu şema, internet üzerinden toplanan haber metinlerine uygulanarak doküman dizmenin dizin sıkıştırma oranına olan etkisi incelenmiştir.

Anahtar Sözcükler— Terim-Doküman Matrisi, Kayıpsız Veri Sıkıştırma, Ters Dizin Sıkıştırma, Değişken Sekiz İkili Kodlama, Özyinelemeli Sekiz İkili Kodlama, Doküman Dizme, Matris Bant Genişliği İndirgeme, Çift Kutuplu Sıralama, Tepe Tırmanma

Abstract

Lossless data compression can have a great importance with regards to efficiency in data retrieval through effective decoding algorithms, especially for in-memory databases and information retrieval systems that use caching, not to mention the less required disk space for storage. In this work, we present a novel variable byte encoding technique to compress inverted indexes and a bipolar permutation scheme based on hill climbing to reduce the bandwidth of the term-document matrix.

Applying this scheme to a collection of news crawled from the Web, the effect of document reordering on index compression is investigated.

Keywords— Term-Document Matrix, Lossless Data Compression, Inverted Index Compression, Variable Byte Encoding, Recursive Byte Encoding, Document Reordering, Matrix Bandwidth Reduction, Bipolar Ordering, Hill Climbing

Gönderme ve kabul tarihi: 04.05.2018-17.07.2018
Makale türü: Araştırma

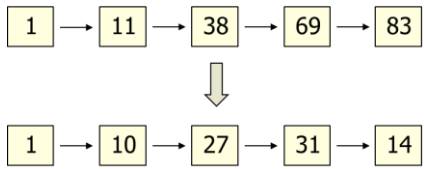
1. Giriş

Kayıpsız veri sıkıştırma tekniklerinden birçok veri tabanı ve bilgi geri kazanım (information retrieval) sistemlerinde faydalanılmaktadır. Sıkıştırılmış verinin disk üzerinde daha az yer kaplamasıyla depolama maliyetinin azaltılması ve bilgi yoğunluğunun artırılıp girdi-çıktı (I/O) işlemlerinin en aza indirgenmesiyle veri transfer hızının artırılması, veri sıkıştırma yöntemlerinin bu kapsamda araştırılmasını gerekli kılan en belirgin ihtiyaçlardandır [1]. Bunun yanında, sıkıştırılmış veri üzerinde sorgu işlemlerinin yapılmasını sağlayan sıkıştırma şemaları, özellikle C-Store [2] gibi sütun tabanlı (column-oriented) veri tabanı sistemlerinde, geleneksel satır tabanlı (row-oriented) veri tabanlarıyla kıyaslandığında yüksek verimde sorgu işlemeyi mümkün kılmaktadır [3]. Sütun tabanlı mimaride, ardışık kayıtların daha fazla benzerliğe sahip olması sıkıştırılabilirliği artırmakta, RLE (Run-Length Encoding) gibi temel kodlama yöntemlerinin veri üzerinde uygulanmasına olanak sağlamaktadır [4]. Benzer şekilde, bilgi geri kazanım sistemlerinde de veriyi önbellekte tutarak hızlı kod çözme teknikleriyle disk okuma maliyetini azaltma ve sıkıştırılmış veri üzerinde işlem yaparak verimliliği artırma imkanı, veri sıkıştırmayı cazip hale getirmektedir. Bu kapsamda yapılan veri sıkıştırma çalışmaları iki ayrı başlık altında incelenmektedir:

sözlük (dictionary) sıkıştırma ve dizin (index) sıkıştırma [5].

Bilgi geri kazanım sistemlerinde sözlük sıkıştırmanın amacı, dokümanlardan çıkarılan terimlerin (kelimeler, kökler, çok kelimeli ifadeler, vs.) ana bellekte alan açısından daha verimli bir şekilde tutulması ve tercihen kod çözmeye ihtiyaç duymadan terim indislerinin belirlenebilmesidir. Önkodlama (front-coding) şemaları [6], sıkıştırılmış son ek dizileri ve ağaçları [7], sıkıştırılmış komut tabloları (hash table) ve sıkıştırılmış Trie [8] bu amaçla kullanılan veri yapıları ve yöntemlere örnek olarak gösterilebilir. Özellikle biyoenformatik alanında, sözlük boyutlarının çok fazla olmasından ötürü sözlük sıkıştırma tekniklerinden yararlanılmaktadır.

Dizin sıkıştırma ise, terimlerin hangi dokümanlarda bulunduğu bilgisinin daha az yer kaplamasını ve mümkünse sorgu şemasında ihtiyaç duyulan temel mantıksal operatörlerin sıkıştırılmış vektörler üzerinde uygulanabilmesini amaçlamaktadır. Bu amacı gerçekleyen en bilindik yöntem, seyrek bir dağılıma sahip olan terim-doküman bit eşleminde yer alan bit vektörlerinin sadece '1' değerlerinin indis bilgilerinin tutulmasını sağlayan ters dizinleme (inverted indexing) tekniğidir [9]. Ters dizin veri yapısı, seyrek bit vektörlerini kayda değer bir oranda sıkıştırmakla kalmayıp üzerinde kesişim ve birleşim işlemlerinin kolayca yapılabilmesinden ötürü mantıksal operatörlerin kullanımını da mümkün hale getirmektedir. Literatürde, ters dizinlerde yer alan sayıların sıkıştırılması işlemi, dizin sıkıştırma (index compression) başlığı altında ele alınmakta olup evrensel (global) ve yerel (local) sıkıştırma şemaları uygulanarak gerçekleştirilmektedir. Söz konusu şemaların etkili olabilmesi için, öncelikle, dizinlere aralık kodlaması (difference coding) uygulanmaktadır. Ortaya çıkan doküman aralıkları (d-gaps) hem daha küçük sayılardan oluşmakta hem de daha az seyrek bir dağılım sergilemektedir. Böylece, entropi kodlaması ve evrensel sayı kodlaması (integer coding) teknikleriyle, dizinler, daha etkin bir şekilde sıkıştırılabilmektedir. Aralık kodlama uygulamasının bir örneği Şekil 1'de verilmiştir. Doküman indislerinin aralık değerlerinden geri çevrimi ise, dizinin ilk değerinden başlayarak özyinelemeli bir şekilde toplanmasıyla sağlanmaktadır. Denklem (1)'de aralık dizisi *A*'nın doküman indislerinin bulunduğu *D* dizisine özyinelemeli çevrimi formülize edilmiştir.



Şekil 1. Aralık kodlaması

$$D[i] = \begin{cases} A[i] & if\ i = 0 \\ A[i\text{-}1] + A[i] & if\ i > 0 \end{cases} \quad (1)$$

Aralık değerlerinin parametrik olmayan evrensel şemalar ile sıkıştırılması için kullanılan yöntemler arasında birli kodlama (unary coding), değişken sekiz ikili kodlama (vByte encoding) [10], Elias-γ ve Elias-δ kodlamaları [11], Simple9 kodlaması [12], Huffman kodlaması [13] ve aritmetik kodlama [14] gibi entropi temelli teknikler bulunmaktadır. Birbirinden farklı aralık değeri sayısının genelde yüksek olması, kod kelimelerinin bulunduğu arama tablolarının da boyutunu artırdığından Huffman kodlamasının aralıklara doğrudan uygulanmasını zorlaştırmaktadır. Fraenkel ve Klein (1985), *LLRUN* isimli bit eşlemi sıkıştırma şemalarında, Elias kodlamasındaki üssel bölümlendirmeyi (bucketing) genelleştirerek Fibonacci dizisi gibi farklı sayım (enumeration) sistemlerinin sayı kodlamaya nasıl uygulabileceğini göstermiş ve Huffman kodlamasını, kod kelimelerindeki bölümlendirme seviyelerini belirten ön ekleri (prefix) kodlamak için kullanıp aralık sıkıştırma oranını optimize etmişlerdir [15].

Evrensel şemaların sıkıştırma performansı, aralıkların istatistiksel dağılımlarına göre değişkenlik göstermektedir. Örneğin, birli kodlamanın optimum sıkıştırma oranını yakalaması için x uzunluğundaki bir aralığın $\left(\frac{1}{2}\right)^x$ olasılığı ile geometrik bir dağılıma sahip olması, Elias-γ kodlamasının optimum çalışması içinse aralıkların $\left(\frac{1}{2x^2}\right)$ olasılığı ile dağılması gerekmektedir. Dolayısıyla, dizinlenecek derlemin doküman aralıklarının bu dağılımlara uymaması, evrensel sayı kodlamalarının yetersiz kalmalarına neden olmaktadır. Bu noktada, parametrik yerel kodlama şemaları, aralık değerlerinin dağılım özelliklerini kullanarak evrensel şemalara göre daha iyi bir sıkıştırma performansı gösterebilmektedir. Bu bağlamda, Golomb kodlaması [16], parametrik kodlama şemaları arasında yaygın olarak kullanılan en eski yöntemlerden biridir. Bu

şemada, I sayısı, seçilen bir M böleniyle çarpan ve kalanlarına ayrılır:

$$I = \lfloor I/M \rfloor * M + (I \bmod M) \qquad (2)$$

Golomb kodu, $\lfloor I/M \rfloor$ çarpanının birli kodlaması ile $(I \bmod M)$ kalan değerinin azami $\lceil \log_2 M \rceil$ bitle temsil edilecek ikili kodlamasının birleşiminden oluşmaktadır. Kodlanacak sayının belirli bir p olasılığı ile geometrik dağılım sergilediği varsayımı, Golomb kodlamasının temelini oluşturmaktadır. Bu nedenle, dizin aralıklarının sıkıştırılması işleminde, rastgele seçilen bir terimin rastgele seçilen bir dokümanda bulunma ihtimali p ise, söz konusu dizinde X uzunluğunda bir doküman aralığının oluşma olasılığı aşağıdaki şekilde hesaplanmaktadır:

$$P(x) = (1-p)^{x-1} * p \qquad (3)$$

Geometrik dağılan sayı değerlerini optimum Golomb koduna dönüştüren M parametresi, p olasılığı kullanılarak tahminlenebilmektedir [17]. Dolayısıyla, dizin aralıkları sıkıştırılırken terim başına düşen ortalama doküman sayısına göre global bir M parametresi ile uygulanan Golomb kodlamasının, Elias kodlaması gibi parametrik olmayan tekniklere kıyasla daha iyi sıkıştırma oranları verdiği gözlenmiştir [10, 18]. Ayrıca, Golomb kodlamasının terim dizinlerine farklı parametrelerle uygulanması doküman aralıkları dağılımının daha iyi temsil edilmesini mümkün kılmaktadır. Böyle bir durumda, kazanılan alanın, bellekte tutulması gereken terim sayısı kadar parametrenin harcayacağı alandan büyük olması beklenmektedir.

Dizin sıkıştırma için kullanılan şemalardan bir diğeri de değişken sekiz ikili kodlama yöntemlerinden biri olan vByte kodlamasıdır. Bu yöntemde, sayı değerleri değişken sayıda sekiz ikili (byte) ile temsil edilmekte olup sekiz ikililerinin her birinin ilk bit değeri, yeni bir sayının kodlamasına başlanıp başlanmadığı bilgisini taşımaktadır. Bu sayede, kodlanan sayının kaç tane sekiz ikili ile temsil edileceği belirlenmiş olur. Sekiz ikili seviyesinde kodlama yapan vByte, bit seviyesinde kodlama yapan Elias ve Golomb kodlamalarıyla kıyaslandığında, sıkıştırma oranı açısından daha verimsizdir. Bunun nedeni, sayıyı kodlayan sekiz ikililerin sabit uzunlukta olmasının son sekiz ikilinin fazlalığına yol açmasıdır. Ancak, diğer taraftan, vByte kodlamasında 8-bit seviyesinde gerçekleşen kod çözme işlemi, bit seviyesinde işlem yapan yöntemlere göre daha hızlı tamamlanmaktadır [18, 19]. Bu nedenle, değişken sekiz ikili kodlama şemaları, alan kazancından feragat etmelerine karşın kod çözme verimliliklerinden ötürü bilgi geri kazanım sistemlerinde kullanılmaktadır [20].

Çalışmamızın ilk bölümünde, daha önce tanıtmış olduğumuz yeni bir değişken sekiz ikili kodlama yöntemi olan özyinelemeli sekiz ikili kodlama (recursive byte encoding) [21] şemasının daha detaylı bir incelemesi yapılmış ve sıkıştırma oranı açısından vByte kodlamasıyla karşılaştırılmıştır. İkinci bölümde ise, çift kutuplu sıralamaya dayalı olan ve çok sayıda doküman için ölçeklendirilebilir bir doküman dizme (document reordering) şeması geliştirilmiştir. Bu şemanın eniyilemesi için tepe tırmanma (hill climbing) algoritması uygulanmış ve çift kutuplu sıralamanın farklı sayı kodlama şemaları çerçevesinde, internetten toplanan 100000 adet haber dokümanı üzerinde dizin sıkıştırmaya olan katkısı ölçülmüştür.

2. Özyinelemeli Sekiz İkili Kodlama

Özyinelemeli sekiz ikili kodlama (RBE), Golomb kodlamasında olduğu gibi bir X sayısının bölen, çarpan ve kalan ile temsil edilmesine dayalıdır. Golomb kodlamasında isteğe bağlı olarak seçilen M parametresi, sekiz ikili kodlamada 256 olarak sabitlenmiştir. Böylece, X sayısı aşağıdaki şekilde temsil edilmiş olur:

$$X = 256 * C + R \qquad (4)$$

İfadedeki C çarpanı 8-bit ile temsil edilemeyecek kadar büyükse, bu çarpan da (4) ifadesiyle çarpan ve kalanına ayrıştırılarak kodlama işlemi devam eder. Bu işlem, en son üretilen çarpan 256'dan küçük oluncaya kadar tekrarlanır. İşlemin özyinelemeli ifadesi aşağıdaki gibidir:

$$f(x) = \begin{cases} x - 1 & if\ x < 256 \\ 256 * f\left(\left\lfloor \frac{x}{256} \right\rfloor\right) + (x \bmod 256) & if\ x \geq 256 \end{cases} \qquad (5)$$

Örneğin, 1000 sayısı (256*3 + 232) şeklinde çarpan ve kalanına ayrıldığında, 3 sayısı 256'dan küçük olduğu için sayının 1 eksiği çarpan bilgisi olarak tutulmaktadır. Böylece, 1000 sayısı [255, 2, 232] şeklinde 3 sekiz ikili, yani 24 bit ile temsil edilmiş olur. Diğer yandan, 158965 sayısı kodlanacak olursa, çarpan değeri $\lfloor 158965/256 \rfloor$, 256 değerinden küçük olmadığı için kodlamaya devam edilir ve 620 çarpanı [255, 1, 108] olarak kodlanır. Böylece, 158956 sayısı [255, 255, 1, 108, 245] şeklinde 5 sekiz ikili, yani 40

bit ile temsil edilmiş olur. Kodlama ve kod çözme işlemlerinin algoritmik adımları aşağıda sırasıyla verilmiştir.

Algoritma 1. Kodlama

```
1:  bytes ← {}, i ← 0
2:  ENCODE(N, bytes)
3:      IF N < 256 THEN
4:          bytes[i] ← N – 1
5:          i ← i + 1
6:          RETURN bytes
7:      ENDIF
8:      WHILE N ≥ 256
9:          unshift(bytes, 255)
10:         i ← i + 1
11:         R ← N mod 256
12:         N ← ⌊N/256⌋
13:         bytes← ENCODE(N, bytes)
14:         bytes[i] ← R
15:         i ← i + 1
16:         RETURN bytes
17:     ENDWHILE
18: ENDENCODE
```

Algoritma 2. Kod Çözme

```
1:  c ← 0, i ← 0
2:  DECODE(bytes)
3:      i ← i + 1
4:      IF bytes[i-1] < 255 THEN
5:          IF c == 0 THEN
6:              c = bytes[i-1] + 1
7:              RETURN c
8:          ELSE
9:              c = 256*c + bytes[i]
10:             RETURN c
11:         ENDIF
12:     ELSE
13:         x = 256*DECODE(bytes) + bytes[i]
14:         i ← i + 1
15:         RETURN x
16:     ENDIF
17: ENDDECODE
```

RBE'nin harcadığı bit sayısı vByte ile karşılaştırmalı şekilde Tablo 1'de gösterilmiştir. Tablodan görüldüğü üzere RBE, sıkıştırma performansı bakımından ortalaması daha küçük olan sayı dağılımlarında başarılıdır. Aralıkların geometrik dağılım gösterdiği varsayıldığında, RBE'nin aralıkları vByte'tan daha iyi sıkıştırması için p değerinin en az kaç olması gerektiği yaklaşık olarak tahminlenebilir. Bu amaç doğrultusunda, kümülatif geometrik dağılım fonksiyonu kullanılarak bir aralığın Tablo 1'de verilen sayı değerleri arasında kalma olasılıkları hesaplanmıştır. Rastgele seçilen bir terimin rastgele seçilen bir dokümanda bulunma olasılığı p ise, aralık değerlerinin N sayısından küçük olma olasılığı aşağıdaki şekilde hesaplanmaktadır:

$$P(x < N) = 1 - (1-p)^N \qquad (6)$$

Bu fonksiyon kullanılarak aralık değerlerinin M ve N sayıları arasında kalma olasılığı bulunur:

$$P(M \le x < N) = P(x < N) - P(x < M) = (1-p)^M - (1-p)^N \qquad (7)$$

Tablo 1. RBE ve vByte kodlamalarında sayı aralıkları ve gereken bit sayıları

RBE	vByte	Sayı Aralığı
8	8	$0 \le N < 2^7$
8	16	$2^7 \le N < 2^8$
24	16	$2^8 \le N < 2^{14}$
24	24	$2^{14} \le N < 2^{16}$
40	24	$2^{16} \le N < 2^{21}$
40	32	$2^{21} \le N < 2^{24}$
56	32	$2^{24} \le N < 2^{28}$
56	40	$2^{28} \le N < 2^{32}$

RBE'nin hangi koşulda vByte'a göre daha iyi sıkıştırma performansı gösterdiğinin bulunması için kazanç sağlayan ve kayba yol açan sayı aralıklarının (7) ile hesaplanan olasılık toplamlarının birbirlerine eşit olduğu p değerinin bulunması gerekmektedir. Aşağıda, RBE'nin daha iyi sıkıştırması için sağlanması gereken eşitsizlik verilmiştir:

$$P(2^7 \le x < 2^8) * 8 > P(2^8 \le x < 2^{14}) * 8 + P(2^{16} \le x < 2^{21}) * 16 + P(2^{21} \le x < 2^{24}) * 8 + \cdots \qquad (8)$$

Eşitsizliği basitleştirmek amacıyla, sağ tarafta yer alan olasılık değerlerinin, $P(2^8 \le x < 2^{14})$hariç, eşitliği sağlayan p değerinde yok sayılabilir oldukları varsayılmıştır. Böylece, yukarıdaki ifade $P(2^7 \le x < 2^8) > P(2^8 \le x < 2^{14})$ ifadesine indirgenmiş olur. Aşağıda bu ifade kullanılarak p değerinin çıkarım adımları gösterilmiştir.

$$(1-p)^{128} - (1-p)^{256} > (1-p)^{256} - (1-p)^{16384}$$

$$(1-p)^{16384} \approx 0$$

$$(1-p)^{128} > 2 * (1-p)^{256}$$

$$(1-p)^{128} < 0.5$$
$$(1-p) < 0.9946$$

$$p > 0.0054 \qquad (9)$$

Geometrik dağılan aralıkların beklenen değeri, aralık değerleri ve karşılık gelen olasılıkların çarpımlarının toplamıdır:

$$E(x) = \sum_{x=1}^{\infty} x * (1-p)^{x-1} * p$$

$$E(x) = (1-p)/p \qquad (10)$$

Beklenen değer, p olasılığı 0.0054 alındığında 184.19 olmaktadır. Dolayısıyla, RBE'nin vByte'tan daha iyi sıkıştırması için terim dizinlerinin ortalama doküman aralığı yaklaşık olarak 184 değerini geçmemelidir. Ancak, çok sayıda doküman içeren geniş derlemlerde ortalama aralık değeri büyük olasılıkla 184'ten fazla olacağından RBE yerine vByte kodlamasının kullanılması daha az sayıda sekiz ikili ile dizinlerin temsil edilmesini sağlayacaktır. Bu noktada doküman dizme işlemi, aralıkların yerelliğini (locality) artırarak ortalama aralık değerini küçülttüğünden önem arz etmektedir. Bu nedenle bir sonraki bölümde, çift kutuplu doküman dizme şeması gerçek veri üzerinde uygulanarak RBE'nin sıkıştırma performansına olan etkisi ölçülmüştür.

3. Doküman Dizme

Doküman dizme, terimlerin dizin aralıklarının toplamının minimum olmasını sağlayacak doküman sıralamasının bulunması işlemidir. Bu işlemin öncelikli amacı ortalama doküman aralığı değerini küçülterek dizinlerin sıkıştırılabilirliğini artırmaktır. Bunun yanı sıra, aralıkların küçülmesi sıkıştırılmış dizinlerin kod çözme verimliliğini de artırarak ortalama sorgu işleme süresini kısaltmaktadır [22]. Bu bakımdan doküman dizme işlemi, bilgi geri kazanım sistemlerinde hem alan hem de hız açısından fayda sağlaması nedeniyle öneme sahiptir.

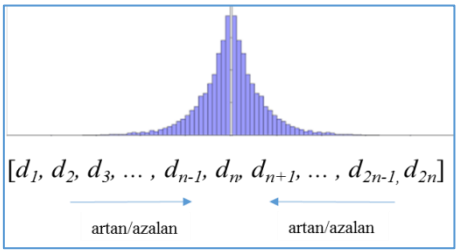
Doküman dizme, aslında, terim-doküman matrisi A'dan elde edilen simetrik doküman kovaryans matrisi A^TA'nın bant genişliği indirgeme problemidir. Bant genişliği indirgeme bir birleşimsel eniyileme (combinatorial optimization) problemi olup NP-tam karmaşıklığındadır [23]. Dolayısıyla, problemin çözümüne yönelik farklı yaklaşımlar içeren birçok akademik çalışma mevcuttur. Bu bağlamda, kullanılan metasezgisel yöntemler arasında genetik algoritma [24], benzetimli tavlama (simulated annealing) [25] ve tabu araması [26] örnek olarak gösterilebilir. Diğer yandan, Cuthill-McKee algoritması [27], GPS algoritması [28] ve Sloan algoritması [29] gibi çizge temelli yöntemler, PCA (Principal Component Analysis) [30] ve MDS (Multi-Dimensional Scaling) [31] gibi boyut indirgeme teknikleri ve ikili kümeleme (biclustering) [32] simetrik matrislerin bant genişliğini indirgeme amacıyla uygulanan farklı yaklaşımlardır. Ancak, bu yöntemler çok büyük matrislere uygulandıkları takdirde yüksek miktarda hafıza ve hesaplama kaynağına ihtiyaç duymaktadırlar. Bu nedenle çalışmamızda dokümanlar, matris veya çizge yapısı yerine bir dizide temsil edilmişlerdir. Bu dizi, bir sıralama ölçütüne göre çift kutuplu (bipolar) olacak şekilde sıralanmış ve ardından tepe tırmanma algoritması ile eniyilemesi yapılmıştır.

3.1 Çift Kutuplu Dizilim Şeması

Doküman dizme işleminin çok sayıda dokümandan oluşan derlemlere ölçeklenebilmesi için basit sıralama şemalarının önerildiği çalışmalar literatürde yer almaktadır. Silvestri [33], dokümanların indirildikleri URL'e göre alfabetik şekilde sıralanmasının dizin aralıklarını önemli ölçüde azalttığını göstermiştir. Benzer şekilde, Ramaswamy vd. [34], e-ticaret ürünlerine ait dokümanların ürün kategorisi gibi ontolojik özelliklere göre sıralanmasının dizin sıkıştırma oranı ve sorgu işleme hızına olan katkısını incelemiştir. Bu yöntemler, hızlı ve etkili olmalarına rağmen dokümanlarla ilgili harici bilgi gerektirdiklerinden ötürü her türlü derleme uygulanamıyor olup bilhassa dokümanların URL bilgilerinin tutulduğu büyük ölçekli arama motorlarının doküman sıralama optimizasyonu için tercih edilmektedirler.

Dizin aralıklarını küçülten diğer bir doküman dizme yöntemi, dokümanların içerdikleri ayrık terim sayısına göre sıralanmasıdır [35]. Bu yöntem, aslında, toplam ayrık terim sayıları benzer dokümanların paylaştıkları ortak terim sayısının, genel olarak daha yüksek olacağı beklentisine dayalıdır. Diğer taraftan, terim sayıları arasındaki fark yüksek olan iki doküman, doğal olarak daha az sayıda ortak terime sahip olacaktır. Bunun nedeni, ortak terim sayısının alabileceği azami değerin daha az sayıda terim içeren dokümanda bulunan terim

sayısıyla sınırlanmış olmasıdır. Çift kutuplu dizilimde ise dokümanlar, toplam terim sayısı gibi bir sıralama ölçütü ile sıralaması en yüksek dokümandan en aza doğru, dizinin ortasından başlayacak şekilde sağ ve sol kutuplara dağıtılır. Bu şekilde sıralaması yapılmış doküman dizisi Şekil 2'de resmedilmiştir.



Şekil 2. Çift Kutuplu Dizilim

Görselde görüldüğü üzere dokümanlar, belirlenen bir sıralama ölçütüne göre dizinin ortasından uçlara doğru dizilmektedirler. Dizme sürecinde sağ ve sol kutupta toplanan doküman sayıları eşitse, dokümanın ekleneceği taraf rastgele belirlenmektedir. Aksi halde, eksik sayıda olan tarafa sıradaki doküman eklenmektedir. Çift kutuplu dizilimin belirlenen bir$R(d)$ sıralama ölçütü ile oluşturulma adımları aşağıda verilmiştir.

Algoritma 3. Çift Kutuplu Dizme

```
SORT DocArray by R(d)
SET Left_Size to 0
SET Right_Size to 0
FOR each Doc in DocArray
    IF Left_Size == Right_Size
        IF Rand() <0.5
            REMOVE Doc from DocArray
            UNSHIFT DocArray with Doc
          SET Left_Size to Left_Size + 1
        ELSE
          SET Right_Size to Right_Size + 1
        ENDIF
    ELSE
        IF Left_Size < Right_Size
            REMOVE Doc from DocArray
            UNSHIFT DocArraywithDoc
          SET Left_Size to Left_Size + 1
        ELSE
          SET Right_Size to Right_Size + 1
          ENDIF
     ENDIF
ENDFOR
RETURN DocArray
```

Çalışmamızda iki farklı sıralama ölçütü kullanılarak çift kutuplu dizilim şemasının dizin sıkıştırmaya olan etkisi ölçülmüştür. Bunlardan ilki, dokümanda geçen ayrık terim sayısı; ikincisi de dokümanda geçen ayrık terimlerin IDF (Inverse Document Frequency) değerlerinin toplamıdır. Toplam doküman sayısının *N* olduğu bir derlemde *t* terimini içeren doküman sayısı *f(t)* ise, terimin IDF değeri aşağıdaki şekilde hesaplanmaktadır:

$$idf(t) = \log\left(\frac{N}{f(t)}\right) \qquad (11)$$

Bu durumda, dokümanlar için kullanılan sıralama ölçütü aşağıdaki şekilde elde edilir:

$$R(d) = \sum_i idf(t_i) \qquad (12)$$

Tanımlanan bu iki sıralama ölçütü kullanılarak çift kutuplu dizilimin terimlerin ortalama bant genişliğini azaltma oranının ölçülmesi için 100000 adet doküman üzerinde deneyler yapılmıştır. Bu dokümanlar, geliştirilen bir veri toplayıcı aracılığıyla internetten çekilen Türkçe haber metinlerinden oluşmaktadır. Terimlerin çıkarılması sürecinde herhangi bir köke indirgeme işlemi yapılmadığından 346003 adet terimden oluşan oldukça seyrek bir terim-doküman matrisi elde edilmiştir. Terim dizinlerinin başlangıç ve bitiş noktaları arasındaki uzaklıkların toplamının en aza indirgenmesi doğrultusunda oluşturulan hedef fonksiyonu aşağıdaki gibidir:

$$min\ F(x) = \sum(d_i - d_j) \qquad (13)$$

Hazırlanan veri setiyle oluşturulan dizinlerin ortalama bant genişliği, 5 farklı doküman dizme şeması kullanılarak (13)'te ifade edilen toplam genişliğin terim sayısına bölünmesiyle hesaplanmıştır. Bu şemalar sırasıyla rastgele dizilim, terim sayısına göre artan (ascending) dizilim, toplam IDF değerine göre artan dizilim, terim sayısına göre çift kutuplu dizilim ve son olarak toplam IDF değerine göre çift kutuplu dizilimdir. Tablo 2'de ortalama dizin genişliği değerleri karşılık gelen dizilim şemalarıyla birlikte verilmiştir.

Tablo 2. Dizilim şemalarına karşılık gelen ortalama bant genişlikleri

Dizilim Şeması	Ortalama Bant Genişliği
Rastgele	35545
Terim Sayısı (Artan)	15638
Toplam IDF (Artan)	15097
Terim Sayısı (Çift Kutuplu)	13100
Toplam IDF (Çift Kutuplu)	12888

Tablodan görüldüğü üzere çift kutuplu dizilim, tekdüze artan dizilime göre terim sayısı ölçütünde %16.2, toplam IDF ölçütünde de %14.6 oranında terim dizinlerinin ortalama bant genişliğini azaltmıştır. Toplam IDF ölçütünü kullanan çift kutuplu dizilim, rastgele dizilime göre %63.7 oranında bant genişliğini azaltarak en iyi sonucu veren dizilim olup terim sayısı ölçütünü kullanan dizilime göre %1.6 oranında iyileştirme sağlamıştır. Bu nedenle, bir sonraki bölümde değinilecek olan tepe tırmanma algoritması bu şema üzerinden uygulanmıştır.

3.2 Tepe Tırmanma

Çift kutuplu dizilim, bant genişliğini başlı başına önemli ölçüde azaltmasının yanı sıra, aynı zamanda dizilen dokümanları sağ ve sol kutup olmak üzere ikiye ayırarak ikili yer değiştirmeye (swap) dayalı tepe tırmanma ile eniyileme yapılmasını mümkün kılmaktadır. Burada hedeflenen, terim dizinlerinin başlangıç ve bitiş noktalarının olabildiğince tek bir kutupta yer almasını sağlayarak yerelliğin artırılmasıdır. Başka bir ifadeyle, ortak terim sayısı minimum düzeyde olan dokümanların zıt kutuplarda bulundurulması amaçlanmaktadır. Bu doğrultuda, çift kutuplu dizilimdeki toplam IDF ölçütüne dayalı sıralama asgari ölçüde bozulacak şekilde, simetrik bir yer değiştirme operatörü tanımlanmıştır. Bu operatör, bir dokümanı, önceden tanımlı bir tolerans aralığında, yalnızca toplam IDF değeri kendisine yakın olan zıt kutuptaki bir doküman ile değiştirebilmektedir. Böylece, N tane elemana sahip çift kutuplu dizilmiş bir D dizisinde yer alan $D[i]$ dokümanının yer değiştirebileceği dokümanlar, belirlenen bir A tolerans değeri ile $D[N - i - 1 \pm A]$ indis aralığında yer alan elemanlarla sınırlandırılmış olur. Bu bakımdan her doküman için, uçlarda bulunan istisnai durumlar hariç, zıt kutupta yer alan $2A$+1adet yer değiştirilebilecek aday bulunmaktadır.

İki doküman arasında yer değiştirme işlemi, bu dokümanlarda bulunan tüm ayrık terimlerin toplam bant genişliğini azalttığı takdirde gerçekleşmektedir. Tolerans aralığında bu şartı sağlayan birden fazla sayıda aday mevcut ise bant genişliğini en çok azaltan doküman seçilmektedir. Derlemde yalnızca bir kere geçen terimler bulundukları dokümanın yer değiştirmesi durumunda toplam bant genişliğini etkilememektedir. Diğer taraftan, derlemde birden fazla sayıda geçen bir terim için bulunduğu dokümanın bir başka dokümanla yer değiştirmesi halinde, dizin genişliğinin ne kadar değiştiğinin bulunabilmesi için üç farklı durumun ayrı ayrı incelenmesi gerekmektedir. Bir t teriminin D dizisinde ilk defa geçtiği doküman indisi i, son defa geçtiği doküman doküman indisi j ise, t terimini içeren k indisine sahip bir dokümanın m indisine sahip başka bir dokümanla yer değiştirmesi durumunda, bant genişliği değişimi aşağıdaki koşullara göre şekillenmektedir:

1. $i<k<j$: Eğer k değeri i ve j arasında kalıyorsa aşağıdaki koşullar incelenmelidir:
    a. $i \leq m \leq j$: Bu durumda t terimine ait dizinin genişliği etkilenmez ve $j - i$ olarak kalır.
    b. $m<i$:Bu durumda t terimine ait dizinin genişliği artarak $j - m$ olur.
    c. $j<m$:Bu durumda t terimine ait dizinin genişliği artarak $m - i$ olur.
2. $i == k$: Eğer k değeri dizinin başlangıç indisi olan i değerine eşitse, yani, yeri değiştirilecek olan doküman t terimini içeren ilk dokümansa, aşağıdaki koşullar incelenmelidir:
    a. $m<i$:Bu durumda t terimine ait dizinin genişliği artarak $j - m$ olur.
    b. $i<m \leq j$: Bu durumda m indisinde bulunan doküman t terimini içeriyorsa, t terimine ait dizinin genişliği etkilenmez ve $j - i$ olarak kalır. Aksi halde, t terimini içeren ilk dokümanın yeri değiştiğinden dizinin yeni başlangıç indisi bulunur ve $\bar{\imath}$ olarak belirlenir. Sonuç olarak dizinin genişliği azalarak $j - \bar{\imath}$olur.
    c. $j<m$: Bu durumda m indisinde bulunan doküman t terimini içeriyorsa, dizinin genişliği artarak $m - i$ olur. Aksi halde, t terimini içeren ilk dokümanın yeri değiştiğinden dizinin yeni başlangıç indisi bulunur ve $\bar{\imath}$ olarak belirlenir. Dizinin genişliği $m - \bar{\imath}$olur ve bu değerlere bağlı olarak eski değerine göre azalmış veya artmış olabilir.
3. $j == k$: Eğer k değeri dizinin bitiş indisi olan j değerine eşitse, yani, yeri değiştirilecek olan doküman t terimini içeren son dokümansa, aşağıdaki koşullar incelenmelidir:
    a. $j<m$: Bu durumda t terimine ait dizinin genişliği artarak $m - i$ olur.

b. $i \leq m < j$: Bu durumda m indisinde bulunan doküman t terimini içeriyorsa, t terimine ait dizinin genişliği etkilenmez ve $j - i$ olarak kalır. Aksi halde, t terimini içeren son dokümanın yeri değiştiğinden dizinin yeni bitiş indisi bulunur ve $\bar{j}$ olarak belirlenir. Sonuç olarak dizinin genişliği azalarak $\bar{j} - i$ olur.

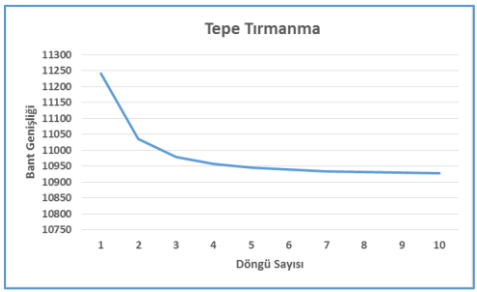
c. $m < i$: Bu durumda m indisinde bulunan doküman t terimini içeriyorsa, dizinin genişliği artarak $j - m$ olur. Aksi halde, t terimini içeren son dokümanın yeri değiştiğinden dizinin yeni bitiş indisi bulunur ve $\bar{j}$ olarak belirlenir. Dizinin genişliği $\bar{j} - m$ olur ve bu değerlere bağlı olarak eski değerine göre azalmış veya artmış olabilir.

Yukarıda belirtilen adımlar uygulandığında ikili yer değiştirmeye uğrayan bir dokümanda bulunan bir terime ait dizin genişliğinin ne kadar arttığı veya azaldığı bilgisi elde edilmektedir. İki doküman arasında gerçekleşen yer değiştirmenin tüm dizinlerin toplam bant genişliğine olan etkisi ise iki dokümanda bulunan tüm ayrık terimlerin dizin genişliği değişimlerinin toplanmasıyla hesaplanmaktadır. İki doküman ancak ve ancak ortaya çıkan safi değere göre toplam bant genişliğinin azalması durumunda yer değiştirebilmektedir. Bu işlem, dizideki her dokümanın belirlenen bir tolerans aralığıile yer değiştirebileceği adayların belirlenmesiyle uygulanır ve bir döngü tamamlanmış olur. Tepe tırmanma süreci, bu döngülerin toplam bant genişliği artık azalmayacak bir noktaya geldiğinde veya değişimi belirlenen bir eşik değerinin altında kaldığında sona erer. Tablo 3'te toplam IDF ölçütüyle çift kutuplu sıralaması yapılmış doküman dizisine farklı tolerans aralıklarıyla uygulanan tepe tırmanmanın ortalama bant genişliğine etkisi gösterilmiştir.

Tablo 3. Tepe tırmanmanın farklı tolerans aralıklarıyla ortalama bant genişliğine etkisi

A	Döngü Sayısı	Ortalama Bant Genişliği
1	10	11203
2	10	11017
3	10	10962
4	10	10927

Tablodan görüldüğü üzere tolerans aralığı arttıkça ortalama bant genişliği daha fazla oranda azalmaktadır. Ancak tepe tırmanma algoritmasının tamamlanma süresi $2A+1$ oranında arttığından A değeri en fazla 4'e kadar çıkarılmış ve 10 döngü sonrası 10927 ortalama bant genişliği elde edilmiştir. Şekil 3'te verilen grafikte, tepe tırmanmanın her döngüde bant genişliğini ne kadar azalttığı gösterilmiştir.



Şekil 3. Tepe tırmanma döngüleriyle elde edilen ortalama bant genişlikleri (A = 4)

Şekilde görüldüğü üzere tepe tırmanmanın, döngü sayısı arttıkça bant genişliğine olan etkisi azalmakta ve yeterli sayıda döngü gerçekleştiği takdirde belirli bir seviyeye yakınsaması beklenmektedir. Bu sebeple, yapılan deneylerde algoritmanın tamamlanma süresini makul seviyede tutmak amacıyla döngü sayısı 10 ile sınırlandırılmıştır. Tablo 4'te tepe tırmanma sonucunda, başlangıç ve bitiş indisleri yalnızca sol veya sağ kutupta yer alan dizinlerin sayısının %23.2 oranında çoğalıp her iki kutupta bulunan dizinlerin sayısının %25.9 oranında azalarak yerelliğin arttığı ve terim dizinlerinin ortalama bant genişliğinin çift kutuplu dizilime göre %15.2 oranında azaldığı gözlenmektedir. Sonuç olarak, çift kutuplu dizilim şemasının tepe tırmanma ile eniyilemesi yapıldığında ortalama bant genişliğinin rastgele dizilime göre %69.2 oranında azaldığı görülmüştür.

Tablo 4. Tepe tırmanmanın yerelliğe etkisi

Dizilim Şeması	Sol Kutup	Sağ Kutup	Her İki Kutup	Bant Genişliği
Toplam IDF (Çift Kutuplu)	90242	92329	163432	12888
Toplam IDF (Çift Kutuplu) + Tepe Tırmanma (A = 4)	115045	109805	121153	10927

4. Değerlendirme

Bu bölümde tepe tırmanma ile eniyilemesi yapılmış çift kutuplu dizilim şemasının dizin aralıklarının sıkıştırılabilirliğine katkısı RBE, vByte ve Elias-δ kodlamaları kullanılarak ölçülmüştür. Bunun yanı sıra, RBE ve vByte kodlama şemaları, ürettikleri sekiz ikililerin Huffman kodlamasıyla ne kadar oranda sıkıştırılabildiği üzerinden karşılaştırılmıştır. Tablo 5'te 100000 doküman ve 346003 terimden oluşan terim-doküman bit eşleminin ters dizinleme sonucu ne kadar alan harcadığı gösterilmiştir.

Tablo 5.Ters dizinlemenin sıkıştırma oranı

Bit Eşlemi	Ters Dizin (32-bit)	Ters Dizin (17-bit)
4.028 GB	26.13 MB	13.88 MB

Ters dizinlerin sabit uzunlukta kod kelimeleriyle temsil edilmesi durumunda 100000 dokümanı temsil edecek minimum uzunluğa sahip kod kelimesi uzunluğu 17'dir ($2^{16} < 100000 < 2^{17}$). Bu durumda, 4.028 GB yer kaplayan bit eşlemi %0.34'üne inerek 13.88 MB yer kaplamaktadır. 17-bit uzunlukta kod kelimeleriyle sayıların temsil edilmesi durumunda aralık kodlaması veya doküman dizmenin sıkıştırma oranına herhangi bir etkisinin olması mümkün değildir. Ancak, aralıkların minimum sabit uzunluğa sahip kodlarla temsil edildiğinde elde edilen sıkıştırma oranı, değişken bit ve sekiz ikili kodlama şemalarının karşılaştırmalı biçimde ne kadar sıkıştırdıklarının görülmesi bakımından önemlidir. Tablo 6'da, aralıkların RBE, vByte ve Elias-δ kodlamalarıyla kodlandıklarında, sırasıyla rastgele dizilim ve tepe tırmanma ile optimize edilmiş çift kutuplu dizilim uygulandığında ne kadar yer kapladıkları verilmiştir. Son satırda geliştirilen doküman dizme şemasının kullanılan kodlama tekniklerinin sıkıştırma oranına ne kadar katkı sağladığı görülmektedir.

Tablo 6.Doküman dizmenin aralık sıkıştırmaya katkısı

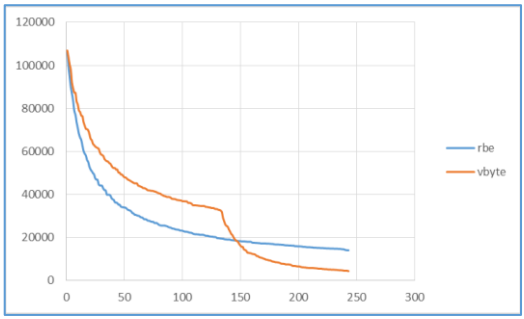
	17-bit	RBE	vByte	Elias-δ
Rastgele Dizilim	13.88 MB	12.26 MB	10.35 MB	10.52 MB
Çift Kutuplu + Tepe Tırmanma	13.88 MB	10.13 MB	9.17 MB	8.50 MB
Katkı	%0	%17.4	%11.4	%19.2

Terim dağılımının oldukça seyrek olduğu bir veri setinin kullanılmış olması, ortalama dizin aralığının, RBE'nin vByte'tan daha iyi sıkıştırması için gereken seviyeden fazla olmasına neden olduğundan vByte kodlaması RBE'ye göre aralıkları daha iyi oranda sıkıştırmıştır. Doküman dizme şemasının RBE'nin sıkıştırma oranına daha fazla katkıda bulunması ise ortalama dizin aralığının azalması nedeniyle beklenen bir durumdur. Bu noktada, RBE ve vByte kodlamaları sonucu ortaya çıkan sekiz ikililere Huffman kodlaması uygulandığında elde edilen sıkıştırma oranları Tablo 7'de verilmiştir.

Tablo 7.Sekiz ikili Huffman kodlaması sonuçları

	RBE + Huffman	vByte + Huffman
Rastgele Dizilim	9.63 MB	9.31 MB
Çift Kutuplu + Tepe Tırmanma	7.89 MB	7.80 MB
Katkı	%18.1	%16.2

RBE'den çıkan sekiz ikililere Huffman kodlaması uygulandığında %22.1, vByte'tan çıkanlara uygulandığında ise %14.9 oranında alan kazancı sağlanmaktadır. Bunun nedeni, RBE sonucu ortaya çıkan sekiz ikililerin, frekansları büyükten küçüğe sıralandığında vByte'a göre daha keskin bir düşüş seyri göstermesidir. Bu durum Şekil 4'te daha açık bir şekilde görselleştirilmiştir. Grafikte, vByte sekiz ikililerinin iki farklı dağılımın birleşimi görünümü sergilemesinin nedeni ilk bit değerinin 'sürdürme biti' (continuation bit) olarak kullanılmasıdır.



Şekil 4. RBE ve vByte sekiz ikili frekans dağılımı

5. Sonuç ve Gelecek Çalışmalar

Bu çalışmada, yeni bir sekiz ikili kodlama yöntemi olan özyinelemeli sekiz ikili kodlama (RBE) şeması incelenmiş ve vByte kodlamasıyla karşılaştırılmıştır. Bunun yanı sıra, doküman dizme problemi kapsamında çok sayıda dokümana ölçeklenebilen çift kutuplu dizilim şeması geliştirilmiş ve bu şemanın dizin aralıklarının sıkıştırılabilirliğine olan katkısı ölçülmüştür. Aynı zamanda, RBE'nin, vByte'a göre hangi koşullar altında daha iyi sıkıştırma performansı gösterdiği irdelenmiş; doküman dizme ve Huffman kodlamasının sıkıştırma oranına olan katkısı mukayeseli bir şekilde gösterilmiştir. Sonuçlar maddeler halinde sıralanacak olursa,

1. RBE, bu çalışmadaki gibi çok seyrek matrislerde vByte'a göre daha düşük sıkıştırma performansı sergilemektedir.
2. RBE, doküman dizme işlemi sonucu artan yerellikten ve sonrasında uygulanan Huffman kodlamasından vByte'a göre sıkıştırma performansı açısından daha çok fayda sağlamaktadır. Bu nedenle, geometrik dağıldığı varsayılan doküman aralık değerlerinin ortalama değeri teorik limit olan 184 değerini geçse bile söz konusu işlemlerin yapılması, seyrek terim-doküman matrislerinde dahi RBE'nin daha iyi sıkıştırma performansı göstermesini sağlayabilir.
3. Doküman dizme kapsamında tepe tırmanmayla eniyilemesi yapılan çift kutuplu dizilim şeması, dizin aralıklarının değişken bit seviyesinde kodlama yapan yöntemlerce sıkıştırılabilirliğini %19'a varan oranda artırmıştır. Yöntemin lineer karmaşıklığa sahip olup çok sayıda dokümana kolaylıkla ölçeklenebilir olması da diğer bir avantajıdır.

En son aşamada Huffman kodlaması uygulanmış sekiz ikililerin hızlı kod çözümü için komut tablolarının [36] kullanılması öngörülmektedir. Kodlanacak sembol sayısının yalnızca 256 adet olması, oluşturulacak komut tablolarının makul seviyede alan harcamasını sağlayacaktır. Huffman kodlaması uygulanmış dizin aralıklarının ters dizinlere geri çevrimi sürecinde sırasıyla gerçekleştirilecek olan komut tabloları aracılığıyla sekiz ikililerin geri edinimi, değişken uzunlukta sekiz ikililerin kod çözümü ve son olarak aralıkların doküman indislerine çevrimi işlemlerinin tamamlanma süresinin ölçülmesi gelecekte yapılacak çalışmalar arasındadır. Ayrıca, tepe tırmanmanın, yüksek tolerans değerleriyle test edilebilmesi için dinamik programlamayla daha verimli hale getirilmesi ve RBE şemasının kod çözme hızının vByte'ın hızıyla karşılaştırılması da bu araştırmanın devamı niteliğinde olacak çalışmalardır.

Kaynakça

[1] G.Graefe and L.Shapiro. Data compression and database performance. In ACM/IEEE-CS Symp. On Applied Computing pages 22 -27, April 1991.

[2] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-Store: A column-oriented DBMS. In VLDB, pages 553–564, 2005.

[3] D. J. Abadi, S. Madden, and M. Ferreira. Integrating Compression and Execution in Column-Oriented Database Systems. In SIGMOD, 2006.

[4] C. Lin, J. Wang, and Y. Papakonstantinou. Data Compression for Analytics over Large-scale In-memory Column Databases, ACM 2016.

[5] H. Schütze, C. D. Manning, and P. Raghavan. Introduction to Information Retrieval. Vol. 39. Cambridge University Press, 2008.

[6] I. H. Witten, A. Moffat, T. C. Bell. Managing Gigabytes: Compressing and Indexing Documents and Images. San Francisco, CA, USA, 1999.

[7] R. Grossi and J. S. Vitter. Compressed Suffix Arrays and Suffix Trees with Applications to Text Indexing and String Matching. In 32nd ACM Symposium on Theory of Computing, pages 397–406, 2000.

[8] M. A. Martinez-Prieto, N. Brisaboa, R. Canovas, F. Claude, and G. Navarro. Practical compressed string dictionaries. Information Systems, 56:73–108, 2016.

[9] R. Baeza-Yates and B. Ribeiro-Neto. Modern Information Retrieval. Addison-Wesley, 1999.
[10] H. E. Williams,J. Zobel. Compressing Integersfor Fast File Access. Comput.J.42, pp. 193-201, 1999.
[11] P. Elias. Universal codeword sets and representations of the integers. IEEE Transactions on Information Theory, IT-21(2):194–203, 1975.
[12] V. Anh and A. Moffat. Index compression using fixed binary codewords. In Proc. of the 15th Int. Australasian Database Conference, pages 61–67, 2004.
[13] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40.9, pages 1098-1101, 1952.
[14] I. H. Witten, M. N. Radford, and G. C. John. Arithmetic coding for data compression. *Communications of the ACM* 30.6, pages 520-540, 1987.
[15] A. S. Fraenkel and S. T. Klein. Novel compression of sparse bit-strings. Preliminary report. *Combinatorial Algorithms on Words, Volume 12.* NATO ASI Series F, pages 169-183, 1985.
[16] S. W. Golomb. Run-length encoding. IEEE Transactions on Information Theory, vol. IT-12, pp. 399-401, 1966.
[17] R. Gallager, and D. Van Voorhis. Optimal source codes for geometrically distributed integer alphabets. *IEEE Transactions on Information theory* 21.2,pages: 228-230, 1975.
[18] A. Trotman. Compressing inverted files. Information Retrieval, 6.1: 5-19, 2003.
[19] F. Scholer, H.E. Williams, J. Yiannis, J. Zobel. Compressionof Inverted Indexes for Fast Query Evaluation. In SIGIR 2002, pp. 222-229, 2002.
[20] J. Plaisance, N. Kurz, and D. Lemire. Vectorized vbyte decoding. CoRR, 2015.
[21] M. C. Sorkunand C. Özbey. Compression experiments on term-document index. Computer Science and Engineering (UBMK), 2017 International Conference on. IEEE, 2017.
[22] H. Yan, S. Ding, and T. Suel. Inverted index compression and query processing with optimized document ordering. In Proceedings of the 18th international conference on World wide web. ACM, 401–410, 2009.
[23] C. Papadimitriou. The NP-completeness of the bandwidth minimization problem.Computing, vol. 16, pp. 263–270, 1976.
[24] A. Lim, B. Rodrigues, F. Xiao. Integrated genetic algorithm with hill climbing forbandwidth minimization problem. GECCO 2003. LNCS, vol. 2724, pp. 1594–1595, 2003.
[25] E. Rodriguez-Tello, J. K. Hao, J. Torres-Jimenez. An improved simulated annealing algorithm for bandwidth minimization, European Journal of Operational Research 185, pp. 1319–1335, 2008.
[26] R. Martí, M. Laguna, F. Glover and V. Campos.Reducing the bandwidth ofa sparse matrix with tabu search. European Journal of Operational Research 135, pp. 450–459, 2001.
[27] E. Cuthill, J. McKee. Reducing the bandwidth of sparse symmetric matrices. In Proceedings of the 1969 24th National Conference (New York, NY, USA, 1969), ACM '69, ACM, pp. 157–172, 1969.
[28] N. E. Gibbs, W. G. Poole and P. K. Stockmeyer. An algorithm for reducing the bandwidth and profile of a sparse matrix. SIAM Journal on Numerical Analysis 13, pp. 236-250, 1976.
[29] S. W. Sloan. An algorithm for profile and wavefront reduction of sparse matrices. International Journal for Numerical Methods in Engineering 23, 2. 239–251, 1986.
[30] D. Harel and Y. Koren. Graph drawing by high-dimensionalembedding. In Revised Papers from the 10th International Symposium on Graph Drawing. GD '02, Springer-Verlag, pp. 207–219, 2002.
[31] I. Spence andJ. Graef. The determination of the underlying dimensionalityof an empirically obtained matrix of proximities. Multivariate Behavioral Research 9, pp. 331–341, 1974.
[32] Y. Cheng, G. M. Church. Biclustering of expression data. In Ismb, vol. 8, pp. 93–103, 2000.
[33] F. Silvestri. Sorting out the document identifier assignment problem. In Proc. of 29th European Conf. on Information Retrieval, pages 101–112, 2007.
[34] V. Ramaswamy, R. Konow, A. Trotman, J. Degenhardt, and N. Whyte. Document Reordering is Good, Especially for e-Commerce. In Proceedings of the SIGIR 2017 Workshop on eCommerce (ECOM 17), 2017.
[35] S. Büttcher, C. Clarke, and G. Cormack. Information Retrieval: Implementing and Evaluating Search Engines. MIT Press, p. 214, 2010.
[36] Y. Nekritch. Decoding of canonical Huffman codes with look-up tables. In Proc. Conf. Data Compression, p. 566, 2000.