METİN VERİLERDE DİZGİ EŞLEME VE SIKIŞTIRILMIŞ DİZGİ EŞLEME İŞLEMLERİ ARASINDAKİ PERFORMANS FARKLARININ İNCELENMESİ*

H. Nusret BULUŞ[1], Cihat ERDOĞAN[2], Banu DİRİ[3]

ÖZET

Bu çalışmada metin veriler üzerinde yapılmakta olan dizgi eşleme işlemi istatistikleri ile aynı veriler üzerinde gerçekleştirilen sıkıştırılmış dizgi eşleme işlemi istatistikleri karşılaştırılmıştır. Bu kıyaslamayı yapmak için daha önce geliştirdiğimiz bir uygulama* iyileştirilmiştir ve test sonuçları bu uygulama sayesinde elde edilmiştir. Çalışmanın amacına uygun olarak literatürde mevcut dizgi eşleme algoritmalarının üzerinde herhangi bir değişiklik yapılmadan, sıkıştırılmış dizgi eşlemede de kullanılabilmesini sağlayan bir sıkıştırma yöntemi de sunulmuştur.

Yapılan testlerde ikili ve üçlü kodlamaya dayanan sıkıştırma algoritması %30-%35 arası bir sıkıştırma faktörü sunarken, elde edilen sıkıştırılmış dizgi eşleme süresi, sıkıştırılmamış metin üzerinde yapılan dizgi eşleme süresinden daha düşük olarak bulunmuştur. Ayrıca, dizgi eşleme yaparken gerçekleştirilen karakter karşılaştırma sayılarının sıkıştırılmış metinde, sıkıştırılmamış metne göre daha az olduğu saptanmıştır. Dolayısıyla geliştirilen algoritmanın amacı yüksek sıkıştırma oranı sağlamak yerine, sıkıştırılmış dosya ile sıkıştırılmamış dosya arasındaki metin işleme süreleri farklarına dikkat çekmek ve başka uygulamalar için bir fikir oluşturmaktır.

Ayrıca, üretilen algoritma üzerinde bazı değişiklikler yapılarak sıkıştırma oranlarının %5 gibi iyileşmesi sağlanmış ve algoritmanın yeni hali çalışmada verilmiştir.

Anahtar Kelimeler: Veri sıkıştırma, Dizgi eşleştirme, Sıkıştırılmış dizgi eşleştirme, Dizgi değiştirme

EVALUATION OF THE PATTERN MATCHING PERFORMANCE OF COMPRESSED AND UNCOMPRESSED TEXTS*

ABSTRACT

In this study, statistics of the pattern matching on an un/compressed form of the same text data are compared. In order to achieve this goal, a previously developed* application was improved. This modified application provided the test results of this study. The purpose of the study is presenting a compression method that can be used in compressed pattern matching without any changes on pattern matching algorithms which are previously studied in the literature.

During the tests, the digram and trigram encoding based compression algorithm has provided a compression factor between 30-35%, and the as-obtained compressed pattern matching duration on the compressed text is calculated less than the one on the uncompressed text. In addition, it is confirmed that the total number of character comparisons on the compressed text matching is less than the one on the uncompressed texts. Therefore, the purpose of the as-developed algorithm is to draw attention to the pattern matching process time difference between the compressed and uncompressed text, instead of providing a high compression ratio. Besides, the aim of the study is to lead prospective pattern matching applications based on the points captured in this work. In addition, the changes made to the algorithm have increased the compression ratio by 5% and the new version of the algorithm is also explained in this study.

Keywords: Data compression, Pattern matching, Compressed pattern matching, Pattern substitution

* Bu çalışmanın yapılmasında "Analyzing The Performance Differences Between Pattern Matching and Compressed Pattern Matching on Texts" adlı bildiriden yararlanılmıştır.

[1] Yrd.Doç.Dr., Namık Kemal Üniversitesi, nbulus@nku.edu.tr

[2] Arş.Gör., Namık Kemal Üniversitesi, cerdogan@nku.edu.tr

[3] Doç.Dr., Yıldız Teknik Üniversitesi, banu@ce.yildiz.edu.tr

Giriş

Metin sıkıştırma işlemi genellikle depolama alanını azaltmak veya iletişim maliyetini ve süresini düşürmek için kullanılmaktadır. Yapılan bu çalışmada bunlara ek olarak, fazlalıklardan kurtulan ve daha az yer kaplayan, veri üzerinde dizgi eşleme yaparken elde edilen kazançlar tablolar halinde sunulmaktadır.

Veri sıkıştırma kavramı ilk olarak Claude E. Shannon tarafından "A Mathematical Theory of Communication" adlı çalışmasında ortaya atılmıştır (Shannon, 1948). Bilgisayar bilimlerinde ya da bilgi teorisi kavramı içerisinde, veri sıkıştırma ya da kaynak kodlama bilginin daha az bit kullanılarak yeniden kodlanması anlamına gelmektedir.

Dizgi eşleme algoritmaları, metin işlemenin yaygın olarak kullanıldığı alanlarda çok önemli bir yere sahiptir. Dahası, sistem ve yazılım tasarımı gibi diğer bilgisayar bilimleri alanlarında kullanılan programlama yöntemlerinde de sıkça kullanılmaktadır (Crochemore ve Rytter, 2002).

Önceleri sıkıştırılmış metin verileri üzerinde dizgi eşleme işlemi, metni açarak doğrudan dizgi eşleme algoritmalarını kullanmak suretiyle yapılırken, 1990'lardan itibaren bu işlem sıkıştırılmış metni açmadan, dizgi eşleme algoritmalarını adapte ederek sıkıştırılmış metinler üzerinde gerçekleştirilmeye başlanmıştır.

Bu şekilde veri sıkıştırma alanına uygulanmış olan dizgi eşleme tekniklerine sıkıştırılmış dizgi eşleme teknikleri adı verilmiştir. Sıkıştırılmış metni açmadan dizgi eşleme yapma yaklaşımı olan sıkıştırılmış dizgi eşleme yaklaşımı ilk olarak Amir ve Benson (1992)'ın çalışmasında sunulmuştur.

$\sigma=s_1 \ldots s_u$, $\Sigma=\{a_1,\ldots,a_q\}$ alfabesi elemanlarından oluşmuş u uzunluklu bir metin katarı ve $\sigma.c=t_1 \ldots t_n$ de σ'nin $n \leq u$ şartını sağlayan sıkıştırılmış formu olsun.

GİRDİ: Sıkıştırılmış metin $\sigma.c=t_1 \ldots t_n$ ve dizgi $P=p_1 \ldots p_m$

ÇIKTI: Metnin i konumu. s_i konumunda dizgi geçmektedir ve $s_{i+j-1}=p_j$, $j=1,\ldots,m$'dir.

Verilen bir T metnine karşılık gelen sıkıştırılmış metin Z ve dizgi de P ise, sıkıştırılmış dizgi eşleme problemi P'nin T içerisindeki tüm varoluşlarının sadece Z ve P kullanılarak bulunması işlemi sıkıştırılmış dizgi eşlemenin tanımı olarak verilmiştir. Daha önceki algoritmalar Z metnini açarak, standart dizgi eşleme işlemini gerçekleştirirlerdi. Bu işlem $u=|T|$ ve $m=|P|$ olduğu durumda $O(u+m)$ zaman almaktaydı. Ancak kabul edilebilir bir algoritma $n=|Z|$ olduğu durumda en kötü $O(n+m)$ zaman almalıdır (Amihood Amir ve diğ., 1996).

Sıkıştırılmış metin verilerde dizgi eşleme birçok yazar tarafından çalışılmıştır. Alanın önemli ve ilk çalışmalardan olan Amihood Amir ve diğ.'nin (1996) yaptığı çalışma UNIX-Z sıkıştırma algoritması üzerine uygulanmıştır. Gasieniec ve Rytter (1999) yaptıkları çalışmada LZW sıkıştırma algoritması üzerinde sıkıştırılmış dizgi eşleme gerçekleştirmiştir. Farach ve Thorup (1998) LZ1 olarak da adlandırılan Lempel-Ziv algoritmasında sıkıştırılmış dizgi eşlemeyi uygulamıştır. Kärkkäinen ve diğ. (2003) Lempel-Ziv sıkıştırma algoritması ailesi, özellikle de LZ78 ve LZW algoritmaları üzerine yaklaşık dizgi eşleme gerçekleştirmiştir. Kida ve diğ. (1999) LZW sıkıştırma algoritması üzerinde sıkıştırılmış dizgi eşleme yapmak için Shift-Or yaklaşımını geliştirmiştir. Moura ve diğ. (1998) yarı statik kelime tabanlı modelleme ve bit yerine bayt kodlamaya dayanan Huffman Kodlamayı bir arada kullanan bir sıkıştırma algoritması sunmuş ve sundukları bu algoritma üzerinde sıkıştırılmış dizgi eşlemeyi gerçekleştirmiştir. Navarro ve Raffinot (1999) yaptıkları çalışmada LZW sıkıştırma algoritması ile sıkıştırılmış metni bloklara ayırarak sıkıştırılmış dizgi eşleme geçekleştirmiştir. Shibata ve diğ. (2000) bu çalışmadakine benzer bir şekilde ikili yer değiştirmeye dayanan Byte-Pair Encoding (BPE) üzerinde çeşitli arama algoritmalarının performanslarını incelemişlerdir. Manber (1997) yine bu çalışmadakine benzer bir şekilde kullanılan ikili yer değiştirmedeki önemli problemlerden olan üst üste binme (overlapping) problemine bir çözüm getirerek sıkıştırılmış dizgi eşleme gerçekleştirmiştir. Klein ve Shapira (2002) LZ ailesinden bir sıkıştırma algoritması olan LZSS algoritması üzerinde bir takım değişiklikler yaparak sıkıştırılmış dizgi eşlemeyi yaptıkları çalışmada gerçekleştirmiştir.

Bu çalışmada geliştirilen sıkıştırma algoritması Moura ve diğ. (1998)'de ki gibi yarı statik ve kelime tabanlı bir model kullanmakta, ancak Moura ve diğ. (1998)'den farklı olarak bayt tabanlı Huffman Kodlama yerine Manber (1997) gibi ikili yer değiştirme kodlaması yapmaktadır. Geliştirilen algoritmada ikili yer değiştirmeye ilave olarak üçlü yer değiştirme de kullanılmıştır.

Bu çalışmada, bu özellikler göz önüne alınarak hızlı, basit ve byte kodlamaya dayanan, ayrıca sıkıştırılacak metin üzerinde iki geçiş yaparak metne özgü sözlük oluşturan bir sıkıştırma algoritması tasarlanmıştır. Tasarlanan bu algoritma sıkıştırma oranı temel alındığında yeterli seviyede değilse de, sıkıştırılmış dizgi eşleme ile normal dizgi eşleme arasındaki oransal farkları bize tam olarak verebilmektedir.

Sıkıştırma Algoritması bölümünde tasarlanan algoritma anlatılmış ve kullanılan dizgi eşleme algoritmalarından kısaca bahsedilmiştir. *Geliştirilen Uygulama* bölümünde çalışmada yapılan tüm işlemler için kullanılan uygulama görselleriyle anlatılmıştır. *Deneysel sonuçlar* bölümünde geliştirilen algoritma ile kullanılan dizgi eşleme algoritmalarının normal ve sıkıştırılmış dizgi eşleme performansları nümerik sonuçlar ile karşılaştırılmıştır.

Sıkıştırma Algoritması

Temel Algoritma (v1)

Sıkıştırmanın temeli birçok araştırmacı tarafından değerlendirilmiş çok basit bir dizge değiştirme (pattern-substitution) yöntemidir. Amaç yaygın olarak kullanılan karakter gruplarını (ikili ve üçlü) tek bir karakter ile yani 1 bayt ile ifade etmektir. 1 bayt 256 farklı koda izin verirken, normal bir metin dosyasında bu kodların çoğu kullanılmaz. ASCII kodlaması 128 kodu kullanırken, geri kalan 128 kod standart metin dosyalarında kullanılmayan karakterlere ayrılmıştır. Sıkıştırma işlemi bu yaygın olarak kullanılan karakter çiftlerinin ve üçlülerinin 128 ile 255 arasındaki kodlara karşılık gelecek şekilde ilişkilendirilmesiyle oluşturulan tablo kullanılarak yapılır. Burada 128 ile 191 arası kodlar metin içerisinde en çok kullanılan üçlüler için, 192 ile 255 arası kodlar ise metin içerisinde en çok geçen ikililer için kullanılmıştır. Açma (decompression) bu işlemin tam tersi yönde gerçekleştirilmesiyle yapılmaktadır.

Sıkıştırma algoritması hızlı bir algoritmadır ancak, LZ ailesi gibi etkin sıkıştırma oranlarına sahip değildir.

Geliştirilen sıkıştırma algoritması iki geçişli bir sıkıştırma algoritmasıdır. Birinci geçişte modelleyici sıkıştırılacak metne özgü istatistikler tutarak bir sözlük oluşturur. İkinci geçişte ise oluşturulan bu sözlük yardımıyla sıkıştırma işlemi gerçekleştirilir.

Sıkıştırma işlemine geçmeden önce metinde geçen ikililerin ve üçlülerin sıklıkları bulunur. Bunlardan en fazla sıklığa sahip 64 üçlü ve 64 ikili seçilir. Seçilen bu üçüler 128 ile 191 aralığına ait, ikililer de 192 ile 255 aralığına ait ASCII kodlarıyla kodlanarak bir tablo oluşturulur. Sıkıştırma işlemi için sözlük oluşturma aşaması olan birinci geçiş istatistikleri ve sıralamaları ikililer ve üçlüler üzerinde yapıldığı için oldukça az zaman harcamaktadır.

Sözlük oluşturma işleminin gerçekleştirildiği birinci aşamada dikkat edilmesi gereken bazı önemli noktalar mevcuttur. Bunlardan ilki üst üste gelme durumu, ikincisi ise bu üst üste gelme durumundan kaynaklanmakta olan bir katarın sıkıştırılmış metin içerisinde birden fazla sıkıştırılmış formda olabilmesidir.

Tablo 1'de farklı kodlama problemi açıklanmaya çalışılmıştır. Metin içerisinde birkaç farklı yerde "compression" kelimesi geçiyor ise, bu kelime farklı şekillerde kodlanabilmektedir.

Tablo 1: İkili ve üçlülerden oluşan sözlüğün bir kısmı

Sözlük İndeks	İkili/Üçlü
132	th
142	ss
156	om
195	com
203	pre
215	ion
225	e_c

Yukarıdaki sözlük yapısına göre "compression" kelimesi, kendisinden önce gelen harf gruplarına da bağlı olarak;

$\boxed{\text{th}}\,\boxed{\text{e\_c}}\,\boxed{\text{om}}\,\boxed{\text{pre}}\,\boxed{\text{ss}}\,\boxed{\text{ion}}$ → 132, 225, 156, 203, 142, 215

$\boxed{\text{com}}\,\boxed{\text{pre}}\,\boxed{\text{ss}}\,\boxed{\text{ion}}$ → 195, 203, 142, 215

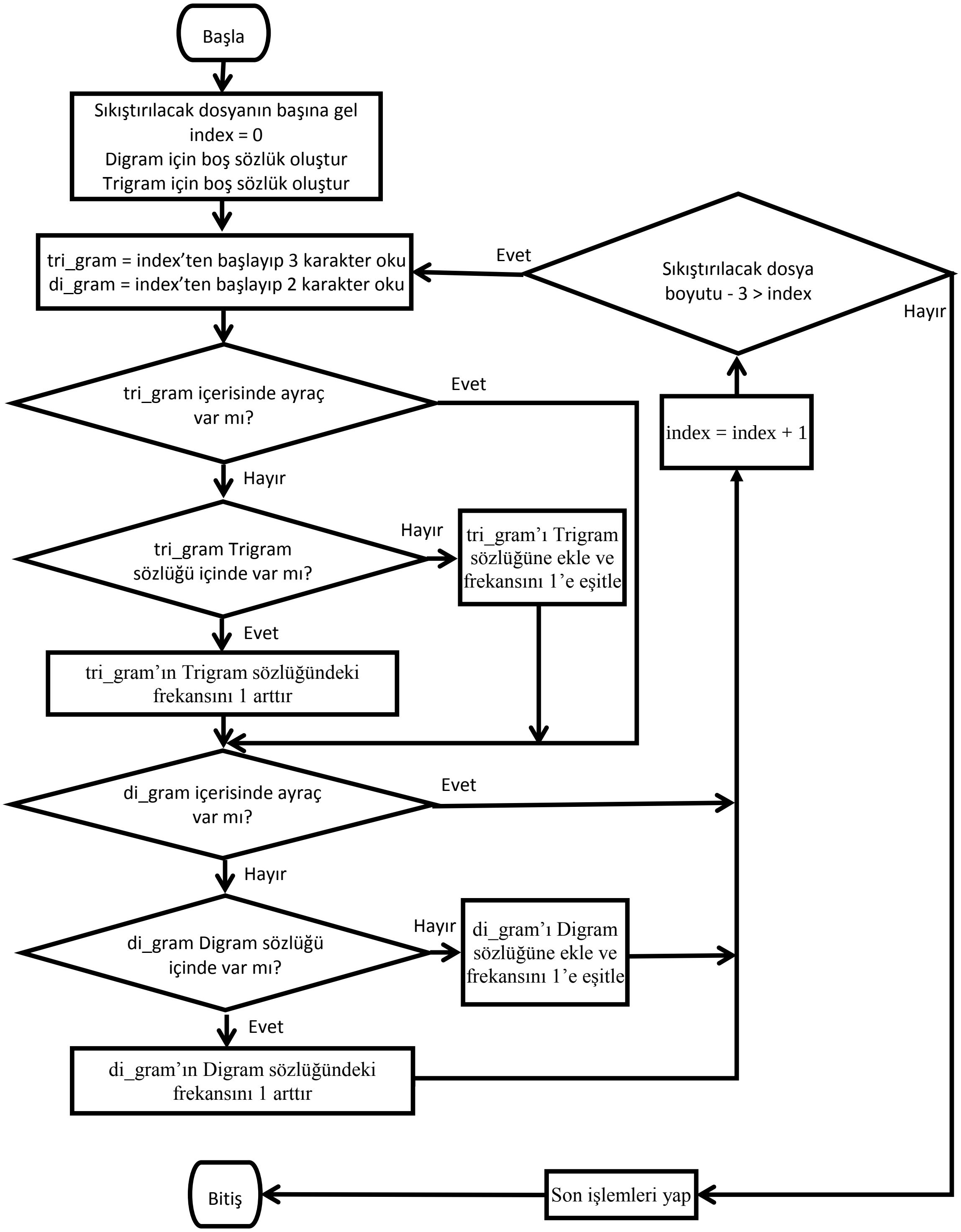
kodlanabilecektir. Bu durum da sıkıştırılmış dizgi eşlemeyi çok ağır bir işlem haline getirecektir.

Bu problemin çözümü için bu çalışmada geliştirilmiş olan algoritma kelime tabanlı olarak tasarlanmıştır. Bu sayede sözlük oluşturma aşamasında üst üste gelme problemine direkt olarak bir çözüm sunulmasa da, sıkıştırma aşamasında bir kelime içerisinde – sıkıştırma kelimenin başından başladığı için – herhangi bir üst üste gelme problemi veya farklı kodlama problemi oluşmamaktadır. Ayrıca, farklı kodlama problemi ortadan kalktığı için, sıkıştırılmış dizgi eşleme yapmak da mümkün olacaktır.

Sözlük oluşturma aşamasında kelimeler baz alındığından üçlülerin birinci ve ikinci karakterlerinin, ikililerin de ilk karakterinin ayraç karakterlerden biri olması mümkün değildir. Bu durumda “e_n” üçlüleri sözlükte yer alamayacak ve yukarıdaki örnekte yer alan “the_compression” katarındaki “compression” kelimesinin kodlaması önündeki karakter grubundan bağımsız olarak standart hale gelecektir.

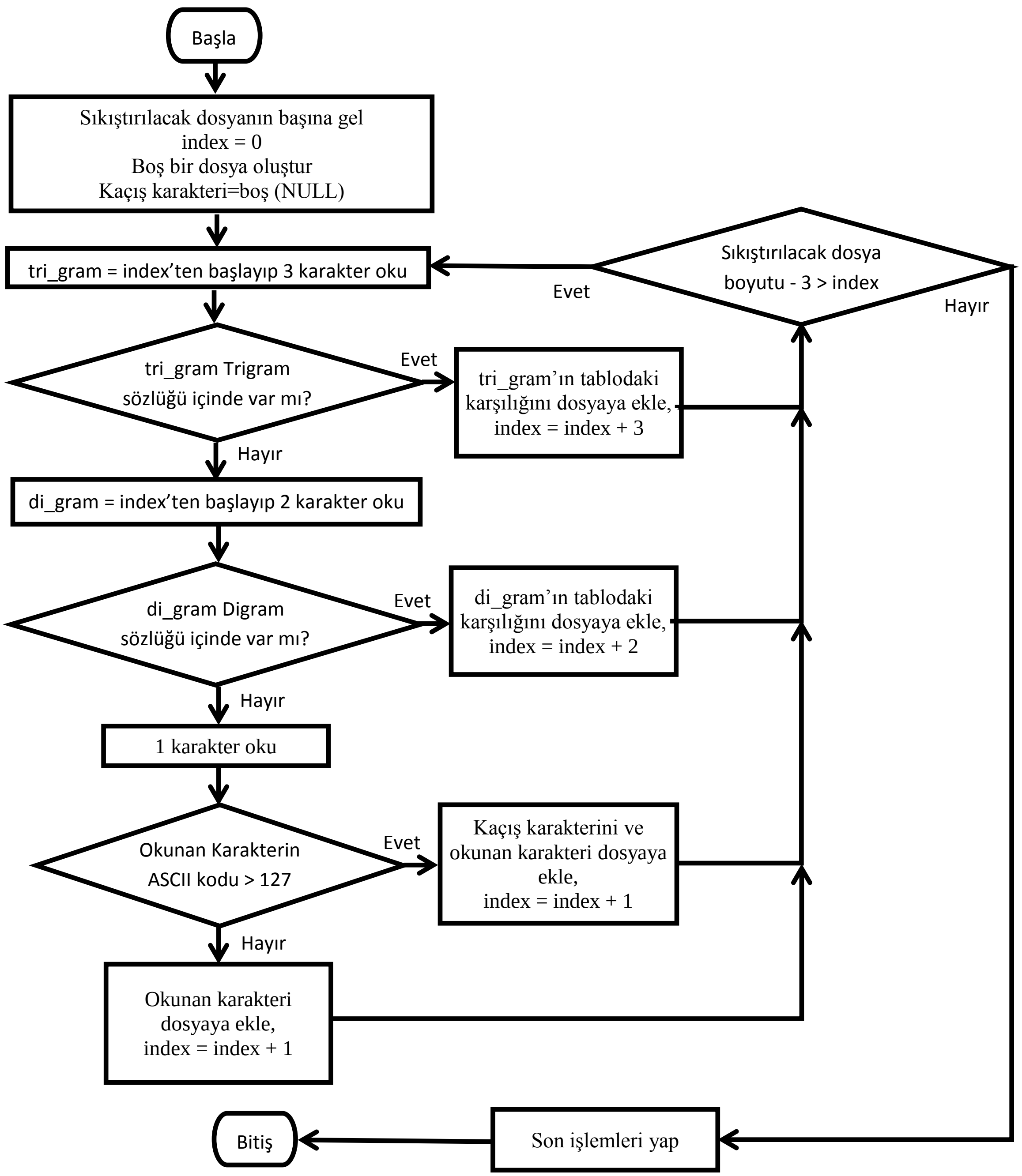
$\boxed{\text{th}}\text{e\_}\boxed{\text{com}}\,\boxed{\text{pre}}\,\boxed{\text{ss}}\,\boxed{\text{ion}}$ → 132, 195, 203, 142, 215 halini alacaktır.

Bu çözümün getirmiş olduğu bazı dezavantajlar da mevcuttur. Bunlardan birincisi sıkıştırılmış dizgi eşlemenin kelime tabanlı olarak sınırlandırılmış olması, ikincisi de kelime sonlarında yer alan ve orijinal metinde oldukça fazla yer kaplayan ayraç karakterlerin dâhil edilememesi ve bununla ortaya çıkan sıkıştırma oranının azlığıdır.



Şekil 1: Sözlük oluşturma safhası akış diyagramı

Sözlük kelimeler baz alınarak oluşturulduğundan, sözlükte yer alan ikililer ve üçlüler de herhangi bir ayraç karakter içermemektedir. Dolayısıyla ikinci geçişte direkt olarak orijinal metin dosyasından okunan karakter grubu sözlükte aranır ve varsa karşılığı kodlanır. Şekil 1'de sıkıştırma algoritmasının birinci, Şekil 2'de de ikinci geçişinin akış şemaları mevcuttur. Sıkıştırma işleminde bir ayrıntı da metin içerisinde ASCII kodu 127'den büyük karakterler varsa bu karakter önlerine kaçış karakteri (boş-NULL) eklenerek sıkıştırılmış dosyaya kodlanır.



Şekil 2: Sıkıştırma safhası akış diyagramı
(Bir önceki adımda oluşturulan Digram ve Trigram sözlükleri kullanılır.)

Algoritmada İyileştirme (v2)

Geliştirilen sıkıştırma algoritması ile 128 adet ikili/üçlü tanımlanabilmekte ve diğer 128 kod, karaktere karşılık gelen ASCII kod olarak kullanılmaktadır. Yapılan denemelerde doğal dil kullanılarak yazılmış bir metin dosyasında 128 karakterden daha az (90-110 arası) karakter kullanıldığı saptanmıştır. Bu durumda üçlü ve ikililere 128 kod kullanmak yerine ASCII kod tablosunda 0-127 arasındaki karakterlerden de kullanılmayanları dâhil ederek daha fazla kod kullanılabileceği düşünülmüştür. Kullanılamayan bu karakterler eşit sayıda ikili ve üçlü için kullanılmaya çalışılmıştır. Ayrıca bu yeni sürümde kaçış karakteri kullanılmadığı için ayrı bir kazanım daha oluşmuştur. Bu yeni durumda;

$\sigma=s_1 \ldots s_u$, $\Sigma=\{a_1,\ldots,a_q\}$ alfabesi elemanlarından oluşmuş *u* uzunluklu bir metin katarında, $\Sigma=\{a_1,\ldots,a_q\}$ alfabesi elemanlarının her biri için yeni kod atamak gereklidir. Atanacak kodlar $q\geq 1$ olan *q* farklı alfabe elemanı için *[1,q]* arasında olacaktır. Dolayısıyla *[q+1,255]* arası kodlar ise ikililer ve üçlüler için kullanılabilecektir.

Bu çalışmada geliştirilen uygulamanın yukarıdaki kod değişimini de içeren ikinci bir sürümü de üretilmiştir. Üretilen bu yeni sürüm de tıpkı ilk sürüm gibi sıkıştırılmış dizgi eşlemenin gerçekleştirilmesine izin verirken, aynı zamanda sıkıştırma oranını da %2-%4 gibi oranlarda iyileştirmiştir. Yapılan testlerin detayları *Deneysel Sonuçlar* bölümünde verilmiştir.

Algoritmada İyileştirme (v3)

Geliştirilen sıkıştırma algoritmasında sıkıştırma oranını arttırmak için en iyi ikililerin ve üçlülerin seçilmesi gerekmektedir. Bu işlemi gerçekleştirmek için seçilecek olan en yüksek sıklığa sahip ikili ve üçlülerin adetlerini parametre olarak alan bir fonksiyon geliştirilmiştir. Fonksiyon ilk aşamada seçilen üçlülerin arasındaki seçilen ikilileri bulup bu ikililerin sıklık değerini içinde geçtiği üçlü sıklığı kadar düşürmektedir. Bu işlemin ardından fonksiyonda ikinci aşamaya geçilerek (1)'deki formüle göre skor hesaplaması yapılmış ve en yüksek skora sahip ikili ve üçlüler alınarak sözlük oluşturulmuştur. Bu sözlük ile algoritmanın sıkıştırma oranı %0,3-%1,2 gibi oranlar arasında iyileşme sağlanmıştır.

Tablo 2: Sözcükler ve sıklık değerlerinin birinci adımdan önceki, sonraki değerleri ve skorları

1. adımdan önceki sıklık değerleri		1. adımdan sonraki sıklık değerleri		Skor değerleri	
Sözcük	Sıklık	Sözcük	Sıklık	Sözcük	Skor
th	5850	the	3000	the	9000
he	3500	th	2850	her	7800
the	3000	her	2600	th	5700
her	2600	he	-2100	he	-4200

T üçlüleri, T_i *i.* üçlüyü, A_{T_i} T_i. üçlünün sıklığını, *D* ikilileri, D_i *i.* ikiliyi, A_{D_i} D_i. ikilinin sıklığını ve S_i *i.* sözcüğün skor bilgisini temsil etmek üzere, skorlar (1)'deki formüle göre hesaplanmıştır.

$$S_i = \begin{cases} sözcük\ ikili\ ise: 2 \times A_{D_i} \\ sözcük\ üçlü\ ise: 3 \ \times A_{T_i} \end{cases} \quad (1)$$

Tablo 2'de geliştirilen fonksiyonun adımları ve bu adımlar sonucu oluşan değişimler örneklenmiştir. Ayrıca, bu aşamada yapılan bazı kodlama iyileştirmeleriyle sıkıştırma süresi %15-%20 oranında iyileştirilmiştir. Yapılan testlerin detayları *Deneysel Sonuçlar* bölümünde verilmiştir.

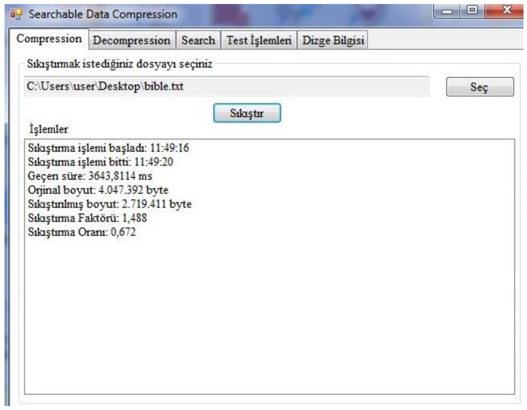
Geliştirilen Uygulama

Bu çalışmada üretilen uygulama C# programlama dili ile geliştirilmiştir. Uygulama 5 farklı işlevi yerine getirmektedir. Bunlar sırasıyla;

1. Sıkıştırma (Şekil 3): *Sıkıştırma Algoritması* bölümünde detayları verilmiş olan sıkıştırma algoritması uygulanarak seçilen bir metin dosyasının sıkıştırılması işlemi gerçekleştirilebilmektedir. Sıkıştırma işlemi gerçekleştirildikten sonra, metinle ilgili sıkıştırma oranı (2), sıkıştırma faktörü (3) ve geçen süre milisaniye bazında program çıktısı olarak üretilmektedir (Salomon, 2006).

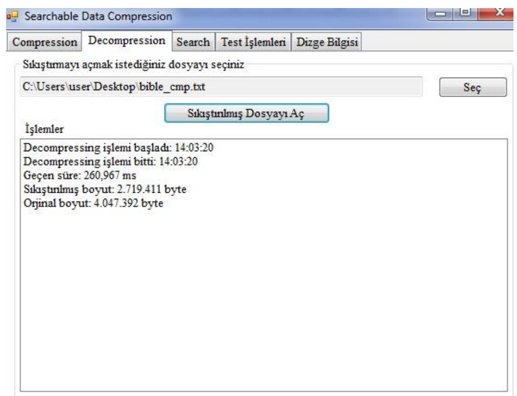
$$Sıkıştırma\ Oranı = \frac{Çıktı\ katarı\ boyutu}{Girdi\ katarı\ boyutu} \quad (2)$$

$$Sıkıştırma\ Faktörü = \frac{Girdi\ katarı\ boyutu}{Çıktı\ katarı\ boyutu} \quad (3)$$



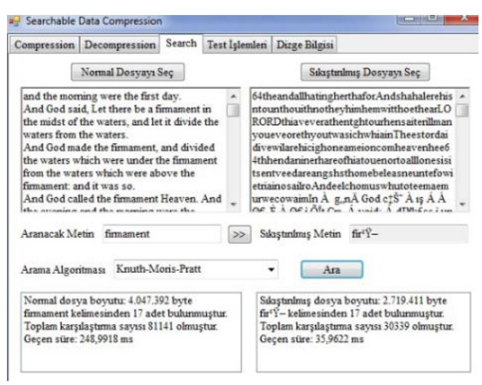
Şekil 3: Sıkıştırma safhası

2. Sıkıştırılmış dosyayı açma (Şekil 4): Üretilen uygulama kullanılarak sıkıştırılmış bir metin dosyası yine bu uygulama ile orijinal hale getirilebilmektedir. Bunun için sıkıştırma algoritması ters sırada gerçekleştirilir. Bu aşamadan sonra çıktı olarak sadece açma işlemi için geçen zaman milisaniye bazında üretilmektedir.



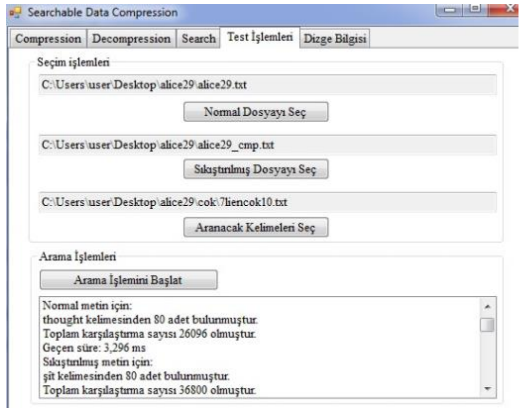
Şekil 4: Açma safhası

3. Dizgi eşleme işlemleri (Şekil 5): Dizgi eşleme işlemi sıkıştırılmış metin dosyası üzerinde yapılabilirken, geliştirilen uygulamada hem açık metin dosyası hem de sıkıştırılmış metin dosyası aynı anda seçilerek, sıkıştırma performanslarının tek bir dizgi için görüntülenmesi gerçekleştirilmiştir. Ayrıca, dizgi eşleme algoritmalarından da hangisinin kullanılacağının seçimini yapmak mümkün kılınmıştır. Geliştirilen uygulama çıktı olarak (hem açık hem de sıkıştırılmış metin için) dizginin metin içerisinde kaç kere geçtiği, bu geçişler bulunurken kaçar adet karakter eşlemesi yapıldığını ve milisaniye bazında eşleme sırasında geçen süreyi üretmektedir.



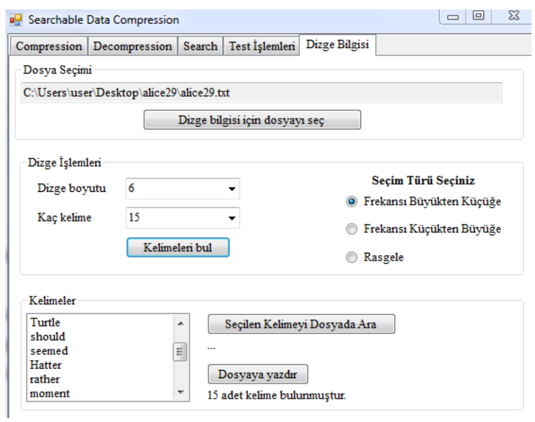
Şekil 5: Tek dizgi arama safhası

4. Test işlemleri (Şekil 6): Bu aşamada sıkıştırılmış ve orijinal metin dosyasının yanı sıra aranacak dizgilerin de yer aldığı dosyanın yolu uygulamadan girilerek, dizgi eşleme gerçekleştirilmiştir. Çıktı olarak; normal ve sıkıştırılmış metin için dizgi eşleme sayıları ve milisaniye bazında süreleri üretilmiştir.



Şekil 6: Çok dizgi arama safhası

5. Dizge bilgisi (Şekil 7): Bu aşama sıkıştırma algoritmasından bağımsız bir aşamadır. Bu aşamada herhangi bir sıkıştırılmamış metin dosyasından, sıkıştırılmış dizgi eşlemede kullanılacak olan arama dosyasının oluşturulması işlemleri yer almaktadır. Orijinal metin dosyasının yolu verildikten sonra, seçilecek dizgilerin karakter bazında boyutu ve kelime sayısı belirlenmektedir. Bu aşamada kullanıcının karşısına üç alternatif çıkar. Bunlar en sık geçen kelimeler, en az geçen kelimeler ya da rastgele seçilmiş kelimeler olabilmektedir. Uygun seçim yapıldıktan sonra seçime uygun kelimeler metinden seçilerek yeni bir dosyaya yazdırılır. Daha sonra bu yeni dosya 4. adımda kullanılabilecektir. Tüm bu aşamalar geliştirilen algoritmanın ikinci sürümü için de birebir aynı olarak gerçekleştirilebilmektedir.



Şekil 7: Çok dizgi dosyası oluşturma safhası

Deneysel Sonuçlar

Geliştirilen uygulama kullanılarak yapılan test işlemleri, özellikleri Tablo 3'te verilen bir sistemde gerçekleştirilmiştir.

Tablo 3: Sistem Özellikleri

Birim	Özellik
İşlemci	Intel(R) Core(TM) i5-3470 CPU 3.20GHz
Bellek (RAM)	8,00 GB
Grafikler	AMD Radeon HD 6570
Birincil sabit disk	38.9 GB Boş (Toplam 111GB) SSD
İşletim Sistemi	Windows 10 Pro (64 bit)

Yapılan testlerde kullanılan metin dosyaları ve alındıkları külliyatlar Tablo 4’te verilmiştir.

Tablo 4: Kullanılan veri seti dosyaları ve ait oldukları külliyatlar

Dosya Adı	Boyut (bayt)	Tanım	Alındığı Külliyat
alphabet.txt	100.000	İngilizce alfabenin 100.000 karakteri dolduracak kadar tekrarlanması	Canterbury
bib	111.261	UNIX'de ASCII metni "refer" biçiminde - 725 bibliyografik referans.	Calgary
asyoulik.txt	125.179	Shakespeare oyunu	Canterbury
alice29.txt	152.089	Alice'in Harikalar Diyarında Maceraları - Lewis Carroll.	Canterbury
news	377.109	ASCII metni - çeşitli konularda USENET toplu iş dosyası.	Calgary
lcet10.txt	426.754	Teknik yazı	Canterbury
plrabn12.txt	481.861	Şiir	Canterbury
book2	610.856	UNIX "troff" formatında ASCII metni - Witten: Bilgisayar Konuşma İlkeleri.	Calgary
book1	768.771	Biçimlendirilmemiş ASCII metni - Thomas Hardy: Far from the Madding Crowd.	Calgary
pi.txt	1.000.000	Pi'nin ilk bir milyon basamağı	Canterbury
world192.txt	2.473.400	CIA Dünyası kitabı	Canterbury
bible.txt	4.047.392	İncil'in King James versiyonu	Canterbury
E.coli	4.638.690	E. coli bakterisinin tüm genomu	Canterbury
dickens.txt	10.192.446	Charles Dickens'ın topladığı eserler	Silesia

Tablo 5 geliştirilen algoritmanın farklı sürümleriyle ve metnin tamamının değil de değişik yüzdelerinin okunması suretiyle ikili/üçlü kodları elde ederek yapılan sıkıştırma işleminin sıkıştırma oranları arasındaki farkı göstermektedir.

Daha fazla ikili ve üçlü kodlamaya imkân veren ikinci sürüm %2-%4 gibi, üçüncü sürüm %0,3-%1,2 gibi bir sıkıştırma oranı iyileştirmesi sağlamaktadır. Ayrıca, ikinci ve üçüncü sürümde metnin sadece %20’si ve %50’si okunarak sonuçlar elde edilmiştir.

Tablo 5: Geliştirilen Algoritmanın Farklı Sürümlerinin Sıkıştırma Oranları (output/input)

	Sıkıştırma Oranları				
Dosya Adı	v1	v2 %50	v2 %20	v3 %50	v3 %20
alphabet.txt	0,335	0,335	0,335	0,335	0,335
bib	0,768	0,745	0,748	0,74	0,74
asyoulik.txt	0,718	0,686	0,687	0,68	0,682
alice29.txt	0,708	0,681	0,684	0,678	0,678
news	0,764	0,741	0,743	0,736	0,738
lcet10.txt	0,692	0,659	0,667	0,653	0,655
plrabn12.txt	0,707	0,676	0,678	0,673	0,672
book2	0,704	0,679	0,681	0,671	0,674
book1	0,691	0,661	0,661	0,653	0,653
pi.txt	0,586	0,466	0,466	0,466	0,466
world192.txt	0,762	0,737	0,736	0,731	0,731
bible.txt	0,672	0,64	0,641	0,633	0,634
E.coli	0,333	0,333	0,333	0,333	0,333
dickens	0,701	0,678	0,678	0,673	0,675

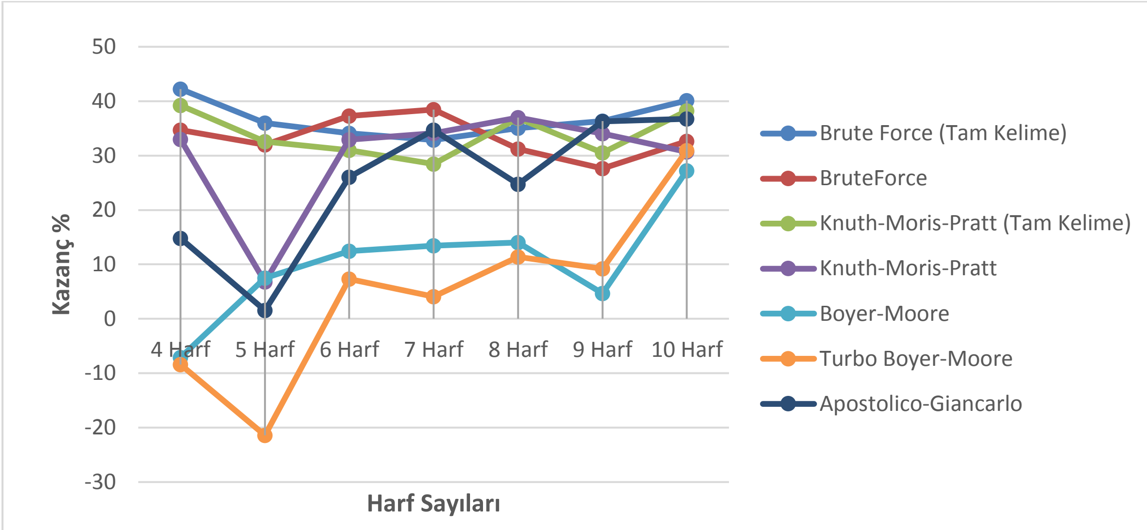
Tablo 5’te de görüldüğü gibi metnin %50’si ve %20’si okunmak suretiyle elde edilen sıkıştırma oranları arasındaki fark oldukça düşüktür. Bu durumda metnin tamamından elde edilecek olan istatistiklerin, çok büyük oranda yaklaşık metnin %20’sinden de elde edilebileceği anlaşılmaktadır. Orijinal metni okuma işlemini %20 oranında sabitleyerek, sözlük elde ederek yapılacak bir sıkıştırma işleminin algoritma hızını oldukça arttıracağı kanısı oluşmaktadır. Buna uygun olarak yapılan testlerde bu fikre paralel sonuçlar elde edilmiştir. Bu sonuçlar Tablo 6’da verilmiştir.

Tablo 6: Geliştirilen Algoritmanın Farklı Sürümlerinin Sıkıştırma ve Açma Süreleri (ms)

	Sıkıştırma ve Açma Süreleri									
Corpus	v1	v2 %50	v2 %20	v3 %50	v3 %20	v1	v2 %50	v2 %20	v3 %50	v3 %20
alphabet.txt	79,31	47,73	35,07	39,18	27,07	8,80	5,95	6,76	5,50	5,51
bib	81,60	68,61	54,86	54,26	41,59	11,06	9,28	9,70	7,77	8,69
asyoulik.txt	85,79	72,99	60,24	59,20	45,86	9,57	9,56	10,78	9,31	8,88
alice29.txt	103,87	113,86	95,74	73,64	54,98	14,91	14,99	15,34	10,78	10,79
news	252,57	199,39	160,61	169,26	126,22	26,01	25,97	26,19	21,16	21,88
lcet10.txt	269,61	216,54	165,30	185,74	137,20	23,25	27,84	28,69	22,17	23,10
plrabn12.txt	322,82	239,20	184,91	205,62	155,99	31,00	30,80	30,89	25,38	25,62
book2	349,96	319,31	237,77	260,21	193,57	33,12	38,11	39,29	31,26	32,05
book1	466,35	406,06	302,23	323,51	244,30	46,87	48,04	49,08	37,53	39,24
pi.txt	676,22	452,50	316,21	406,94	289,36	64,68	43,44	44,74	36,16	35,29
world192.txt	1.515,74	1.210,61	945,28	1.029,24	782,35	130,24	153,48	151,37	121,69	122,36
bible.txt	2.120,26	1.919,72	1.504,75	1.640,33	1.238,44	215,78	218,67	221,82	180,91	184,31
E.coli	2.564,72	1.986,91	1.204,22	1.764,85	1.137,86	162,09	139,38	138,59	130,99	124,96
dickens	5.328,98	4.790,17	3.745,67	4.103,99	3.101,56	560,35	577,66	587,69	462,08	476,08

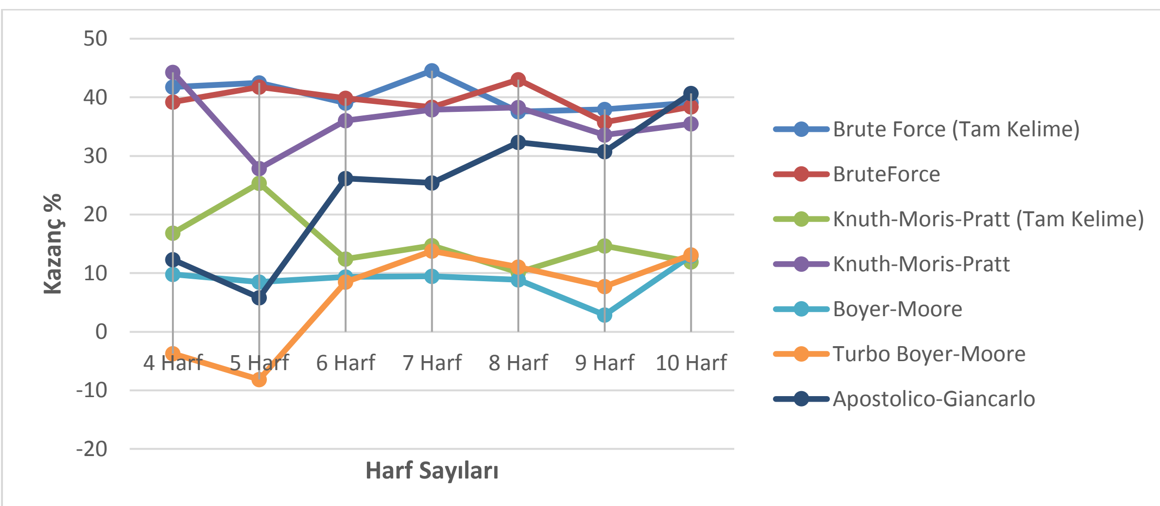
Yapılan çalışma sonucunda v1'in iyileştirilmesi için *Algoritmada İyileştirme (v2)*'de anlatıldığı gibi metinde geçen karakterler yeniden kodlanmıştır ve böylece v2 oluşturulmuştur. Yeni sürümde sıkıştırma oranında %2 ile %4 arasında bir kazanım elde edilmiştir. Bunun yanı sıra sözlük oluşturulurken sıkıştırılacak dosyanın tamamını okumaktansa %20'si veya %50'si okunmuş ve böylece sıkıştırma zamanından kazanılmıştır. Ayrıca *Algoritmada İyileştirme (v3)*'de anlatılan v3 ile sıkıştırma oranında %0,3 ile %1,2 oranında kazanım elde edilmiştir. Bunun yanı sıra bu sürümde yapılan iyileştirilmelerle %15 ile %20 oranında sıkıştırma süresinde kazanımlar elde edilmiştir. Bu kazanımlar Tablo 5'te açıkça görülebilmektedir. Metnin %20 ile %50 gibi oranlarının okunması işleminin tek dezavantajı sıkıştırma oranında %0,1 ile %0,5 arasında oluşan kayıptır. Bu algoritmanın kullanılabileceği ve sıkıştırma hızının sıkıştırma oranından daha önemli olduğu uygulamalarda bu farkın göz ardı edilebilecek bir fark olabileceği kanısı oluşmuştur.

Şekil 8 ve Şekil 9'da çeşitli harf uzunluklarına sahip kelimelerin sıkıştırılmış metinde eşlenme hızı ile normal metinde eşlenme hızları arasındaki fark v1 ve v2 için yüzde cinsinden verilmiştir. Yapılan çalışmada sıkıştırılmış veri üzerinde gerçekleştirilen arama işleminin normal metin üzerinde gerçekleştirilen arama işlemine göre daha hızlı olduğu görülmüştür. Sayıların pozitif olduğu harf sayıları sıkıştırılmış dizgi eşlemenin daha hızlı olduğunu, negatif olan harf sayılarının ise sıkıştırılmış dizgi eşlemenin daha yavaş olduğunu göstermektedir.



Şekil 8: Algoritmalara göre kelime uzunluğunun sıkıştırılmış veri üzerindeki arama kazancına etkisi v1

Aranan kelimenin harf uzunluğu arttıkça arama süresindeki kazanımın da arttığı görülmüştür (Şekil 8 ve Şekil 9). 4 ve 5 harfli kelimelerde yapılan aramalarda Turbo Boyer-Moore algoritmasının normal metne göre daha kötü sonuç verdiği görülmektedir. Bunun sebebi karşılaştırma sayısının doğrusal arama yapan algoritmalara göre daha çok olması ve sıkıştırılmış kelimenin uzunluğunun ortalama 2-3 karakter olmasıdır. Karakter sayısı az olduğu için karşılaştırma sonucu atlanacak karakter sayısı da düşmekte ve algoritma daha kötü çalışmaktadır. Bunun yanı sıra 10 harfli kelimelerde v1'de tüm arama algoritmaların makul bir kazanım oranı verdiği görülmekle birlikte v2'de tüm algoritmalar aynı kazanç oranını elde edememişlerdir. Buradan şu söylenebilir ki v1'de aranacak kelimenin karakter sayısı arttıkça algoritmaların kazançları genel olarak artmakla birlikte v2'de bu durum sadece belirli algoritmalarda geçerli olmuştur.



Şekil 9: Algoritmalara göre kelime uzunluğunun sıkıştırılmış veri üzerindeki arama kazancına etkisi v2

Tablo 7 ve Tablo 8 geliştirilen sıkıştırma algoritmasının farklı iki sürümü ile sıkıştırılmış olan ve "The Catenbury Corpus"ta yer almakta olan "bible.txt" metin dosyası üzerinde yapılmış olan dizgi eşleme işlemlerinin çeşitli istatistiki bilgilerini göstermektedir.

Buna göre yapılan denemeler farklı harf sayılarına göre gruplandırılmış olup, birinci sütunda eşleme yapılacak kelime gruplarının harf sayıları verilmiştir. İkinci sütunda ise, normal ve sıkıştırılmış dizgi eşleme için kullanılmış olan bazı önemli dizgi eşleme algoritmalarının isimleri verilmiştir. Burada dikkat edilmesi gereken nokta, "Tam Kelime" ibaresi ile anlatılmak istenendir. "Tam kelime" ile sadece ilgili kelimenin arandığı anlamına gelmektedir. Bunun sebebi de (Tablo 1) kullanılarak yapılmış olan örnekle paralel olarak bazı kelimelerin, diğer bazı kelimeler içerisinde geçiyor olması durumunda tamamının bulunamaması durumudur. Örneğin, “compress” kelimesi “compression” ve “decompress” gibi diğer kelimelerin içerisinde geçmesine rağmen, farklı şekilde sıkıştırılabilmektedir. Ayrıca, kelimenin sıkıştırılmasına göre farklı sonuçlar üretmeyebilir ve arama yapılabilir. Elde edilen sonuçlardaki farklılıklar bunlardan kaynaklanmaktadır. Gelecekte yapılması planlanan çalışmada bu problemin de üzerine gidilmesi ve optimum sonuca ulaşılması planlanmaktadır.

Tablo 7 ve Tablo 8’deki 1, 3, 4 ve 5’inci sütunlar sırasıyla, orijinal metinden belirtilen harf sayısına uygun olarak rastgele seçilen 10’ar tane kelimeden oluşan kelime gruplarını, normal metinde ortalama dizgi eşleme sürelerini, sıkıştırılmış metinde ortalama dizgi eşleme sürelerini ve bu işlemler sonucunda oluşan süre farkının getirdiği kazanımın yüzde olarak ifade edilmiş olan bilgilerini tutmaktadır.

Tablo 7 ve Tablo 8’deki 6 ve 7’inci sütunlar sırasıyla normal ve sıkıştırılmış metin üzerinde seçilen kelime grupları için dizgi eşleme yapılırken oluşan toplam karakter karşılaştırma sayılarını vermektedir. 6 ve daha fazla karakterden oluşan kelimeler için sıkıştırılmış dizgi eşleme yapmanın daha az karşılaştırma yaptığı açıkça görülmekte iken, 6’dan az karakterden oluşan kelimeler için böyle kesin bir yargıya varmanın mümkün olmayacağı açıkça görülmektedir.

Tablo 7 ve Tablo 8’deki 8 ve 9. sütunlar sırasıyla normal ve sıkıştırılmış metinde bulunan ortalama eşleşme sayılarını vermektedir. Yukarıda da anlatıldığı gibi dizgi eşleme “Tam Kelime” olarak yapıldığı takdirde, sonuçlar normal metin ve sıkıştırılmış metin için birebir olarak örtüşmektedir.

Tablo 7: Algoritmaların harf sayılarına göre normal ve birinci sürüm (v1) ile sıkıştırılmış metin üzerindeki arama performansları

	Arama Algoritması	Normal Metin Arama Süresi (ms)	Sıkıştırılmış Metin Arama Süresi (ms)	Kazanç %	Normal Metin Karşılaştırma Sayısı	Sıkıştırılmış Metin Karşılaştırma Sayısı	Normal Metin Bulunan Eşleşme Sayısı	Sıkıştırılmış Metin Bulunan Eşleşme Sayısı
4 harfliler için ortalama değerler	Brute Force (Tam Kelime)	100.904	**58.294**	**42**	215,949	**29,051**	**3,453**	**3,453**
	BruteForce	76.440	**49.920**	**35**	215,949	**29,051**	3,588	3,521
	Knuth-Moris-Pratt (Tam Kelime)	46.765	**28.423**	**39**	213,400	**28,990**	**3,453**	**3,453**
	Knuth-Moris-Pratt	41.582	**27.873**	**33**	213,400	**28,990**	3,588	3,521
	Boyer-Moore	**30.144**	32.294	**-7**	**1,190,583**	1,250,114	3,588	3,521
	Turbo Boyer-Moore	**43.359**	47.001	**-8**	**1,190,327**	1,250,114	3,588	3,521
	Apostolico-Giancarlo	63.461	**54.089**	**15**	**1,190,583**	1,250,114	3,588	3,521
5 harfliler için ortalama değerler	Brute Force (Tam Kelime)	79.834	**51.122**	**36**	197,190	**25,584**	**2,122**	**2,122**
	BruteForce	78.000	**53.040**	**32**	197,190	**25,584**	2,359	2,328
	Knuth-Moris-Pratt (Tam Kelime)	40.514	**27.309**	**33**	195,285	**25,584**	**2,122**	**2,122**
	Knuth-Moris-Pratt	39.128	**36.478**	**7**	195,285	**25,584**	2,359	2,328
	Boyer-Moore	26.916	**24.893**	**8**	**953,593**	1,018,274	2,359	2,328
	Turbo Boyer-Moore	**34.240**	41.578	**-21**	**953,101**	1,018,274	2,359	2,328
	Apostolico-Giancarlo	50.944	**50.160**	**2**	**953,593**	1,018,274	2,359	2,328
6 harfliler için ortalama değerler	Brute Force (Tam Kelime)	72.968	**48.079**	**34**	118,778	**19,416**	**814**	**814**
	BruteForce	72.465	**45.452**	**37**	118,778	**19,416**	832	818
	Knuth-Moris-Pratt (Tam Kelime)	37.940	**26.191**	**31**	117,080	**19,078**	**814**	**814**
	Knuth-Moris-Pratt	36.657	**24.586**	**33**	117,080	**19,078**	832	818
	Boyer-Moore	19.520	**17.096**	**12**	817,015	**740,462**	832	818
	Turbo Boyer-Moore	30.729	**28.477**	**7**	816,938	**740,462**	832	818
	Apostolico-Giancarlo	46.603	**34.472**	**26**	817,015	**740,462**	832	818
7 harfliler için ortalama değerler	Brute Force (Tam Kelime)	76.933	**51.630**	**33**	196,710	**24,890**	**476**	**476**
	BruteForce	81.120	**49.920**	**38**	196,710	**24,890**	482	481
	Knuth-Moris-Pratt (Tam Kelime)	38.190	**27.327**	**28**	195,287	**24,829**	**476**	**476**
	Knuth-Moris-Pratt	37.487	**24.694**	**34**	195,287	**24,829**	482	481
	Boyer-Moore	20.959	**18.140**	**13**	734,015	**697,449**	482	481
	Turbo Boyer-Moore	28.041	**26.892**	**4**	733,987	**697,438**	482	481
	Apostolico-Giancarlo	52.517	**34.283**	**35**	734,015	**697,449**	482	481
8 harfliler için ortalama değerler	Brute Force (Tam Kelime)	73.616	**47.819**	**35**	131,025	**12,554**	**373**	**373**
	BruteForce	74.880	**51.480**	**31**	131,025	**12,554**	406	406
	Knuth-Moris-Pratt (Tam Kelime)	40.397	**25.509**	**37**	129,970	**12,538**	**373**	**373**
	Knuth-Moris-Pratt	40.216	**25.333**	**37**	129,970	**12,538**	406	406
	Boyer-Moore	16.389	**14.086**	**14**	644,043	**594,122**	406	406
	Turbo Boyer-Moore	24.889	**22.062**	**11**	644,041	**594,122**	406	406
	Apostolico-Giancarlo	43.221	**32.520**	**25**	644,043	**594,122**	406	406
9 harfliler için ortalama değerler	Brute Force (Tam Kelime)	73.927	**47.037**	**36**	151,350	**20,770**	**264**	**264**
	BruteForce	73.320	**53.040**	**28**	151,350	**20,770**	267	265
	Knuth-Moris-Pratt (Tam Kelime)	39.956	**27.768**	**31**	150,291	**20,751**	**264**	**264**
	Knuth-Moris-Pratt	40.300	**26.588**	**34**	150,291	**20,751**	267	265
	Boyer-Moore	15.367	**14.650**	**5**	574,902	**553,020**	267	265
	Turbo Boyer-Moore	22.393	**20.334**	**9**	574,686	**553,020**	267	265
	Apostolico-Giancarlo	46.393	**29.564**	**36**	574,836	**553,020**	267	265
10 harfliler için ortalama değerler	Brute Force (Tam Kelime)	79.430	**47.549**	**40**	228,238	**31,972**	**125**	**125**
	BruteForce	76.440	**51.480**	**33**	228,238	**31,972**	128	128
	Knuth-Moris-Pratt (Tam Kelime)	40.822	**25.218**	**38**	226,155	**31,647**	**125**	**125**
	Knuth-Moris-Pratt	40.627	**28.147**	**31**	226,155	**31,647**	128	128
	Boyer-Moore	14.180	**10.320**	**27**	530,978	**442,155**	128	128
	Turbo Boyer-Moore	23.805	**16.444**	**31**	530,917	**442,155**	128	128
	Apostolico-Giancarlo	42.436	**26.839**	**37**	530,978	**442,155**	128	128

Tablo 8: Algoritmaların harf sayılarına göre normal ve ikinci sürüm (v2) ile sıkıştırılmış metin üzerindeki arama performansları

	Arama Algoritması	Normal Metin Arama Süresi (ms)	Sıkıştırılmış Metin Arama Süresi (ms)	Kazanç %	Normal Metin Karşılaştırma Sayısı	Sıkıştırılmış Metin Karşılaştırma Sayısı	Normal Metin Bulunan Eşleşme Sayısı	Sıkıştırılmış Metin Bulunan Eşleşme Sayısı
4 harfliler için ortalama değerler	Brute Force (Tam Kelime)	80.563	**46.902**	**42**	215,949	**28,094**	**3,453**	**3,453**
	BruteForce	74.520	**45.308**	**39**	215,949	**28,094**	3,588	3,521
	Knuth-Moris-Pratt (Tam Kelime)	42.735	**35.542**	**17**	213,400	**28,065**	**3,453**	**3,453**
	Knuth-Moris-Pratt	41.839	**23.305**	**44**	213,400	**28,065**	3,588	3,521
	Boyer-Moore	31.568	**28.472**	**10**	1,190,583	**1,188,495**	3,588	3,521
	Turbo Boyer-Moore	**43.894**	45.523	**-4**	1,190,327	**1,188,495**	3,588	3,521
	Apostolico-Giancarlo	58.238	**51.071**	**12**	1,190,583	**1,188,495**	3,588	3,521
5 harfliler için ortalama değerler	Brute Force (Tam Kelime)	79.207	**45.579**	**42**	197,190	**25,040**	**2,122**	**2,122**
	BruteForce	74.794	**43.567**	**42**	197,190	**25,040**	2,359	2,327
	Knuth-Moris-Pratt (Tam Kelime)	44.973	**33.561**	**25**	195,285	**25,040**	**2,122**	**2,122**
	Knuth-Moris-Pratt	35.723	**25.769**	**28**	195,285	**25,040**	2,359	2,327
	Boyer-Moore	26.150	**23.926**	**9**	**953,593**	1,051,164	2,359	2,327
	Turbo Boyer-Moore	**36.297**	39.244	**-8**	**953,101**	1,051,164	2,359	2,327
	Apostolico-Giancarlo	49.984	**47.070**	**6**	**953,593**	1,051,164	2,359	2,327
6 harfliler için ortalama değerler	Brute Force (Tam Kelime)	72.185	**43.990**	**39**	118,778	**11,531**	**814**	**814**
	BruteForce	73.143	**43.993**	**40**	118,778	**11,531**	832	831
	Knuth-Moris-Pratt (Tam Kelime)	38.357	**33.590**	**12**	117,080	**11,512**	**814**	**814**
	Knuth-Moris-Pratt	36.853	**23.573**	**36**	117,080	**11,512**	832	831
	Boyer-Moore	20.616	**18.686**	**9**	817,015	**756,539**	832	831
	Turbo Boyer-Moore	29.688	**27.171**	**8**	816,938	**756,539**	832	831
	Apostolico-Giancarlo	45.747	**33.771**	**26**	817,015	**756,539**	832	831
7 harfliler için ortalama değerler	Brute Force (Tam Kelime)	80.289	**44.523**	**45**	196,710	**14,261**	**476**	**476**
	BruteForce	72.488	**44.694**	**38**	196,710	**14,261**	482	481
	Knuth-Moris-Pratt (Tam Kelime)	37.398	**31.901**	**15**	195,287	**14,237**	**476**	**476**
	Knuth-Moris-Pratt	37.864	**23.506**	**38**	195,287	**14,237**	482	481
	Boyer-Moore	17.641	**15.966**	**9**	734,015	**697,416**	482	481
	Turbo Boyer-Moore	28.842	**24.864**	**14**	733,987	**697,416**	482	481
	Apostolico-Giancarlo	47.033	**35.077**	**25**	734,015	**697,416**	482	481
8 harfliler için ortalama değerler	Brute Force (Tam Kelime)	78.460	**48.968**	**38**	131,025	**8,069**	**373**	**373**
	BruteForce	74.484	**42.455**	**43**	131,025	**8,069**	406	406
	Knuth-Moris-Pratt (Tam Kelime)	37.182	**33.419**	**10**	129,970	**8,052**	**373**	**373**
	Knuth-Moris-Pratt	34.817	**21.489**	**38**	129,970	**8,052**	406	406
	Boyer-Moore	15.881	**14.472**	**9**	644,043	**626,758**	406	406
	Turbo Boyer-Moore	25.819	**22.958**	**11**	644,041	**626,758**	406	406
	Apostolico-Giancarlo	45.370	**30.702**	**32**	644,043	**626,758**	406	406
9 harfliler için ortalama değerler	Brute Force (Tam Kelime)	74.809	**46.437**	**38**	151,350	**16,325**	**264**	**264**
	BruteForce	71.979	**46.228**	**36**	151,350	**16,325**	267	265
	Knuth-Moris-Pratt (Tam Kelime)	38.777	**33.085**	**15**	150,291	**16,324**	**264**	**264**
	Knuth-Moris-Pratt	36.977	**24.548**	**34**	150,291	**16,324**	267	265
	Boyer-Moore	14.078	**13.673**	**3**	574,902	**563,368**	267	265
	Turbo Boyer-Moore	22.060	**20.357**	**8**	574,686	**563,353**	267	265
	Apostolico-Giancarlo	41.781	**28.936**	**31**	574,836	**563,368**	267	265
10 harfliler için ortalama değerler	Brute Force (Tam Kelime)	79.055	**48.142**	**39**	228,238	**25,061**	**125**	**125**
	BruteForce	75.690	**46.625**	**38**	228,238	**25,061**	128	128
	Knuth-Moris-Pratt (Tam Kelime)	39.702	**34.956**	**12**	226,155	**25,044**	**125**	**125**
	Knuth-Moris-Pratt	37.794	**24.382**	**35**	226,155	**25,044**	128	128
	Boyer-Moore	13.032	**11.346**	**13**	530,978	**476,231**	128	128
	Turbo Boyer-Moore	19.207	**16.681**	**13**	530,917	**476,231**	128	128
	Apostolico-Giancarlo	43.407	**25.741**	**41**	530,978	**476,231**	128	128

Sonuçlar

Yapılan testler sonucunda elde edilen değerlere göre geliştirilen sıkıştırma algoritmasının ilk sürümü ortalama %30-%35 sıkıştırma faktörü sağlarken, kelime bazlı tam sıkıştırılmış dizgi eşlemeye de izin vermektedir. Benzer şekilde geliştirilen ikinci sürüm birinciye oranla %2-%4 sıkıştırma faktörü iyileştirmesi sunarken, geliştirilen üçüncü sürüm de ikinciye oranla %1-%2 sıkıştırma faktörü iyileşmesi sunmuştur. Yeni geliştirilen her iki sürümde de sıkıştırılmış dizgi eşlemeye izin verilmektedir.

Bununla beraber metinde en sık geçen ikili ve üçlüleri bulmak için metnin tamamını okumak yerine değişik yüzdelerde bir örnekleme uzayı seçilerek metnin bir kısmı okunmuştur. Elde edilen veriler, gerçekleştirilen sıkıştırma işleminin süresini büyük oranda düşürürken, sıkıştırma oranını da göz ardı edilebilecek bir biçimde %0,1-%0,05 değerinde kötüleştirmektedir.

Bunlara ilave olarak bu çalışmanın merkezinde yer alan araştırma fikrine dayanarak, normal metin ve sıkıştırılmış metin üzerinde dizgi eşleme yapılmış ve bazı sonuçlar elde edilmiştir. Bu sonuçlara göre üretilen algoritma ile sıkıştırılmış metin üzerinde çalışmak uzun katarlarda gerek karakter karşılaştırma sayısını azaltması, gerekse de işlem süresini düşürmesi açısından bir avantaj sağlamıştır. Kısa karakter katarlarında ise dizgi eşleme algoritmasına bağlı olarak bazı dizgi eşleme algoritmalarında bu avantajlar korunmuş, bazılarında ise olumsuz sonuçlar elde edilmiştir.

Ayrıca, yapılan test sonuçlarına göre dizge eşleme algoritmalarından BruteForce, Knuth-Moris-Pratt ve türevlerinin Şekil 8 ve Şekil 9'da görüldüğü üzere harf sayısındaki değişime rağmen kazanç değerlerinde fazla bir değişim yaşanmamıştır. Buna ek olarak sıkıştırılmış metinde yapılan aramalarda bu algoritmalar diğerlerine göre daha başarılı olmuştur. Apostolico-Giancarlo algoritması ise harf sayısı arttıkça kazanç değerini arttırmıştır. Boyer-Moore ve türevleri ise kazanç değerleri baz alındığında diğer algoritmalara göre biraz zayıf kalmıştır.

Sonuç olarak yapılan bu çalışmada görülmektedir ki, dizgi eşleme yöntemlerini kullanan çeşitli uygulamaların, normal metin yerine sıkıştırılmış dizgi eşlemeye izin veren sıkıştırma algoritmaları ile oluşturulmuş sıkıştırılmış metin kullanmasının bazı kazanımları olacağı açıktır.

Bu çalışmada amaç, bu kazanımların ayrıntılı bir şekilde işlenerek ortaya konmasıdır. Bu sebeple üretilen sıkıştırma algoritması sıkıştırma oranı açısından çok etkin değildir ancak amaçlanan hedefleri istenilen bir şekilde gerçekleştirmektedir.

Sonuçlardan da anlaşıldığı üzere daha da etkin bir sıkıştırma algoritması kullanılması durumunda elde edilen karşılaştırma sayıları, sıkıştırma oranı, arama süresi vb. değerler daha da iyi olabilecektir.

Ayrıca şunu da belirtmek gerekir ki kelime bazlı sıkıştırılmış dizgi eşleme işleminde BruteForce, Knuth-Moris-Pratt ve türevleri daha fazla kazanç sağlarken Boyer-Moore algoritması da en iyi arama süresini vermiştir.

Kaynakça

Amir, A., & Benson, C. (1992). Efficient two-dimensional compressed matching. In *Data Compression Conference, 1992. DCC '92.* (pp. 279–288). http://doi.org/10.1109/DCC.1992.227453

Amir, A., Benson, G., & Farach, M. (1996). Let Sleeping Files Lie: Pattern Matching in Z-Compressed Files. *Journal of Computer and System Sciences*, *52*(23), 299–307. http://doi.org/DOI: 10.1006/jcss.1996.0023

Boyer, R. S., & Moore, J. S. (1977). A Fast String Searching Algorithm. *Commun. ACM*, *20*(10), 762–772. http://doi.org/10.1145/359842.359859

Crochemore, M., & Rytter, W. (2002). *Jewels of Stringology: Text Algorithms*. Hackensack, NJ, USA: World Scientific. Retrieved from http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/9810248970

Farach, M., & Thorup, M. (1998). String Matching in Lempel—Ziv Compressed Strings. *Algorithmica*, *20*(4), 388–404. http://doi.org/10.1007/PL00009202

Gasieniec, L., & Rytter, W. (1999). Almost-optimal fully LZW-compressed pattern matching. In *Data Compression Conference, 1999. Proceedings. DCC '99* (pp. 316–325). http://doi.org/10.1109/DCC.1999.755681

Kärkkäinen, J., Navarro, G., & Ukkonen, E. (2003). Approximate string matching on Ziv-Lempel compressed text. *Journal of Discrete Algorithms*, *1*(3–4), 313–338. http://doi.org/10.1016/S1570-8667(03)00032-7

Kida, T., Takeda, M., Shinohara, A., & Arikawa, S. (1999). Shift-and approach to pattern matching in LZW compressed text. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *1645*, 1–13. http://doi.org/10.1007/3-540-48452-3_1

Klein, S. T., & Shapira, D. (2002). A new compression method for compressed matching. *Data Compression Conference, 2000. Proceedings. DCC 2000*, 400–409. http://doi.org/10.1109/DCC.2000.838180

Manber, U. (1997). A Text Compression Scheme That Allows Fast Searching Directly in the Compressed File. *ACM Trans. Inf. Syst.*, *15*(2), 124–136. http://doi.org/10.1145/248625.248639

Moura, E. S. De, Navarro, G., Ziviani, N., & Baeza-Yates, R. (1998). Direct pattern matching on compressed text. *Proceedings. String Processing and Information Retrieval: A South American Symposium (Cat. No.98EX207)*. http://doi.org/10.1109/SPIRE.1998.712987

Navarro, G., & Raffinot, M. (1999). A general practical approach to pattern matching over ziv-lempel compressed text. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *1645*, 14–36. http://doi.org/10.1007/3-540-48452-3_2

Salomon, D. (2006). *Data Compression: The Complete Reference*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*(July 1928), 379–423. http://doi.org/10.1145/584091.584093

Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., & Arikawa, S. (2000). Speeding up pattern matching by text compression. *Algorithms and Complexity*, 306–315. http://doi.org/10.1007/3-540-46521-9