İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

SIRA KORUMALI DİZGİ EŞLEŞTİRME TABANLI BAĞLAM MODELLEMESİ İLE KAYIPSIZ VERİ SIKIŞTIRMA

YÜKSEK LİSANS TEZİ

Muhammed Veysel KINGIR

Bilgisayar Bilimleri Anabilim Dalı

Bilgisayar Bilimleri Programı

EYLÜL 2019

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

SIRA KORUMALI DİZGİ EŞLEŞTİRME TABANLI BAĞLAM MODELLEMESİ İLE KAYIPSIZ VERİ SIKIŞTIRMA

YÜKSEK LİSANS TEZİ

Muhammed Veysel KINGIR
(704061023)

Bilgisayar Bilimleri Anabilim Dalı

Bilgisayar Bilimleri Programı

Tez Danışmanı: Prof. Dr. Muhammed Oğuzhan KÜLEKÇİ

EYLÜL 2019

İTÜ, Bilişim Enstitüsü'nün 704061023 numaralı Yüksek Lisans Öğrencisi Muhammed Veysel KINGIR, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "SIRA KORUMALI DİZGİ EŞLEŞTİRME TABANLI BAĞLAM MODELLEMESİ İLE KAYIPSIZ VERİ SIKIŞTIRMA" başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Prof. Dr. Muhammed Oğuzhan KÜLEKÇİ** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Doç. Dr. Gökhan BİLGİN** Yıldız Teknik Üniversitesi	.....
	Dr.Öğr. Üyesi Sefer BADAY İstanbul Teknik Üniversitesi	.....

Teslim Tarihi : **9 Eylül 2019**
Savunma Tarihi : **10 Eylül 2019**

Aileme,

ÖNSÖZ

Yapılan çalışmada dinamik veri sıkıştırma yöntemleri üzerinde iyileştirme hedeflenmiş, yeni bir yaklaşımda bulunulmuş, uygulanmış ve sonuçlar gözlenmiştir. Bu çalışma sırasında ve yüksek lisans eğitim dönemimde desteğini esirgemeyen, fikirlerini ve yaklaşımlarını paylaşarak yol gösteren değerli danışmanım Prof. Dr. Muhammed Oğuzhan Külekçi'ye teşekkürlerimi sunarım.

Eylül 2019

Muhammed Veysel Kıngır
(Bilgisayar Mühendisi)

İÇİNDEKİLER

Sayfa

KISALTMALAR

NYT **:** Not Yet Transmitted (Henüz iletilmemiş)
CDF **:** Cumulative Distribution Function
OPPM **:** Order-Preserve Pattern Matching

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

SIRA KORUMALI DİZGİ EŞLEŞTİRME TABANLI BAĞLAM MODELLEMESİ İLE KAYIPSIZ VERİ SIKIŞTIRMA

ÖZET

Dijital iletişimin çok daha yaygın hale gelmesi ve dijitalleşen veri sahasının genişlemesi ile birlikte veri sıkıştırma tekniklerinde yapılacak iyileştirmeler daha önemli hale gelmiştir. Birçok alanda geçmiş veri istatiksel olarak saklanıp analiz edilmekte, gelecek için yapılan tahmin ve öngörülerde kullanılmaktadır. Bu nedenle veri sıkıştırma oranlarında yapılacak iyileştirmeler bu bağlamda iyi bir katkı sağlayacaktır. Veri sıkıştırmalardaki iyileştirme gereksinimi önümüzdeki dönemde giderek artacağa benziyor.

Buradan yola çıkarak mevcutta hisse senedi analizi, müzik melodi eşleştirmesi gibi farklı alanlarda kullanılan Sıra Korumalı Dizgi Eşleştirme yöntemi, dinamik sıkıştırma algoritmalarına uygulanarak birtakım kazanımlar elde edilmeye çalışıldı. Öncelikle kullanılan dinamik sıkıştırma algoritmalarının nasıl çalıştığı ve modellendiği anlatıldı, verilen örneklerle detaylandırıldı.

Sıra Korumalı Dizgi Eşleştirme yöntemi, verilerin değerlerinin eşleşmesi yerine sıralama ilişkilerinin eşleşmesini gözlemler. Dinamik veri sıkıştırmaları da bağlam modellemesi yaklaşımı ile yapılmaktadır. Bağlam modeli, girdi olan sembolün gerçekleştiği bağlamı kullanarak bu sembolün kaç bitle kodlanacağını belirleyen bir olasılıksal modeldir. Gelen sembolün hangi bağlam modelini kullanacağı, kendisinden önce gelen sabit bir k sayıda karaktere bakılarak belirlenebilir. k-order bağlam modellemesi dediğimiz bu yaklaşımda, model sayısının artması sıkıştırma oranında iyileştirme sağlamakla beraber, kaynak kullanımını ciddi oranda arttırmaktadır. Kullanılan bağlam genişliğinin kaynak kullanımı ve sonuçlar üzerinde etkisi bulunmaktadır. Bu yöntem ile k sayıda karakterin değerlerine bakmak yerine, bu k adet karakterin sıralama ilişkisine bakılarak bağlam değeri belirlendi. Yöntemin uygulanması sonucunda bir takım dosya desenlerinde ciddi iyileştirmeler görülürken, bir kısmında standart yaklaşımla benzer sonuçlar elde edildi. Ancak bunlarda da kaynak kullanımından tasarruf edildi. Bir kısım dosya desenlerinde ise pozitif yönde bir iyileştirme elde edilemedi. Bu yaklaşımda daha iyi sonuçlar elde etmek için ek genişletilmiş yaklaşımlar denendi ve bir takım pozitif sonuçlar alındı. Uygulanan yeni yaklaşımlar kayda değer iyileştirmelerin gözlemlendiği bir sonuç verdi.

LOSSLESS DATA COMPRESS WITH ORDER PRESERVE PATTERN MATCHING BASE CONTEXT MODELLING

SUMMARY

With digital communication becoming more common and expanding the digitalized data field, improvements in data compression techniques have become more important. In many areas, past data is stored and analyzed statistically, and used for future estimations and predictions. Therefore, improvements to data compression rates will make a good contribution in this context. The improvement in data compression is likely to increase in the coming period. As the amount of recorded digital data continues to increase exponentially, the requirements for improving data compression algorithms have begun to increase. In this context, in the context modeling approach used in dynamic compression algorithms, the Order-Preserved Pattern Matching method was used to determine the context modeling. Which context model the input character will use in context modeling is determined by looking at the preceding k characters. This increase in k, that is, context determination over wider length of historical data, can improve the compression ratio, but resource utilization will also increase as the number of models exponentially increases. The aim of this study is to develop new approaches and to obtain better compressed data with less resources.

From this point of view, it has been tried to obtain some gains by applying the Order Preserve Pattern Matching method, which is used in different fields such as stock analysis, music melody matching, and dynamic compression algorithms. Firstly, it was explained how dynamic compression algorithms are used and modeled. Then the context modelling approach explained in detail. Finally, Order-Preserve Pattern Matching implementation explained.

Data compression is a reduction in the number of bits required to represent data. Compressing data can save storage capacity, speed up file transfers, and reduce storage hardware and network bandwidth costs. Compression is performed by a program that uses a formula or algorithm to determine how the size of the data will shrink. For example, an algorithm may represent a bit sequence of smaller 0s and 1s using a dictionary for conversion between them, or a formula may add a reference or pointer to a string of 0s and 1s that the program foresees.

Lossless data compression can be done in two different ways: static and dynamic. In static data compression, a data model is created based on the use of symbols by passing over all data. Then all data is coded according to this model. Resolving data back is the same. First, the data model used for coding is taken, then all data is decoded according to this model. However, in some areas, such as telecommunications, it is not possible to overwrite all the data as all of the data has not yet arrived. They use dynamic data compression algorithms. In dynamic data compression algorithms, a data model is initially determined and the incoming data is encoded with this model. The data model is updated by adding the last encoded data information. During the back-analysis of the data, a data model is also initially determined. The incoming data is

analyzed according to the model, then the data model is updated. The Order-Preserve Pattern Matching Based Context Modeling approach used in this study was applied to Huffman Dynamic Coding and Arithmetic Dynamic Coding methods which are dynamic lossless data compression methods.

In this study, Order-Preserved Pattern Matching method is applied on dynamic compression algorithms. The results were analyzed from different file designs. Additional improvements have been made by increasing the number of contexts used. The results of these were analyzed in detail and some evaluations were obtained. How this method works, development steps, how to implement compression algorithms, the results are explained in detail.

The Order Preserve Pattern Matching method observes the mapping of the sort orders instead of matching the values of the data. Dynamic data compression is also done with the context modeling approach. The context model is a probabilistic model that determines how many bits this symbol is encoded using the context in which the input symbol occurs. Which context model the incoming symbol will use can be determined by looking at a fixed number of characters preceding it. Context modeling is the use of the contents of previously seen characters to determine the encoding of the current character. In this approach, which we call k-order context modeling, the increase in the number of models improves the compression ratio and increases the resource usage significantly. The width of the context used has an impact on resource utilization and results. With this method, instead of looking at the values of k characters, the context value was determined by looking at the sort relationship of these k characters. The context width used in doing so has an impact on resource use and results. Increasing the number of models in context modeling allows the input character to be represented by a lower number of bits. In the case of a single pattern, since each input will use the same pattern, the representation of the string bit length of each character may increase. In case of increasing number of models, it raises the problem of which context model the input will use. In this case, an approach that uses the previous k characters is used to predict which context the input character will use. The Order-Preserve Pattern Matching Based Context Modeling approach tries to determine the context to be used according to the pattern similarity by taking the order of the k data instead of the content of the k characters before the data to be encoded. Thus, although the wider order is used, context length is reduced and resource usage is reduced. This method was applied to the Dynamic Huffman Coding and Adaptive Arithmetic Coding. The results were compared with the standard adaptive algorithm results. Then, to make some further improvements, another parameter was added to increase the context length. This additional parameter is provided by adding any element in the sequence array to the end of the sequence. Which element to be added is determined by comparing the values obtained by adding each element individually. Here, files from various data corpus are used as input. Improvements were observed in a number of file patterns. The same results were obtained in a number of file designs with the standard methods, but a smaller number of contexts were used, saving resource usage. In some file designs, no gain was obtained. A study was performed in which positive results were obtained.

As a result of the implementation of the method, a number of file patterns have been shown to improve significantly, while some have similar results with the standard approach. However, they also saved resources. A positive improvement was not obtained in some of the file patterns. To achieve better results, additional extended

approaches were attempted and a number of positive results were obtained. New approaches have yielded significant improvements.

1. GİRİŞ

Yapılan çalışmada Sıra Korumalı Dizgi Eşleştirmesi yöntemi, dinamik sıkıştırma algoritmaları üzerinde uygulanmıştır. Farklı dosya desenlerinden sonuçlar alınarak analiz edilmiştir. Kullanılan bağlam sayısı arttırılarak ek iyileştirmeler yapılmaya çalışılmıştır. Bunların da sonuçları detaylı olarak karşılaştırılarak analiz edilmiş ve birtakım değerlendirmelere ulaşılmıştır. Bu yöntemin nasıl çalıştığı, geliştirme adımları, sıkıştırma algoritmalarına nasıl implemente edildiği, elde edilen sonuçlar detaylı olarak anlatılmaktadır.

1.1 Amaç

Kaydedilen dijital veri miktarının üstel olarak artmaya devam etmesiyle birlikte veri sıkıştırma algoritmalarının iyileştirilmesi gereksinimleri artmaya başlamıştır. Bu kapsamda dinamik sıkıştırma algoritmalarında kullanılan bağlam modellemesi yaklaşımında bağlam modellemesi belirlenirken Sıra Korumalı Dizgi Eşleştirme yöntemi kullanılmış, böylece verinin içeriği yerine sıralamasının deseni eşleştirildiği için sıkıştırma algoritmamızda daha geniş içerik kullanma imkanı sağlanmıştır. Bağlam modellemesinde girdi olan karakterin hangi bağlam modelini kullanacağı, kendisinden önce gelen k adet karaktere bakılarak belirlenmektedir. Bu k değerinin artması, yani daha geniş uzunlukta geçmiş veriye göre bağlam belirlemesi, sıkıştırma oranında iyileştirme oluşturabilmekle birlikte, model sayısını üstel olarak arttıracağı için kaynak kullanımı da bu oranda artmış olacaktır. Yapılan çalışmayla yeni yaklaşımlar geliştirilerek, daha az kaynak kullanımıyla daha iyi sıkıştırılmış veri elde edilmesi amaçlanmıştır.

1.2 Yöntem

Veri sıkıştırma kabiliyetini arttırmak ve kaynak kullanımını azaltmak amacıyla dinamik sıkıştırma algoritmaları üzerinde yeni bir yaklaşım olarak Sıra Korumalı Dizgi Eşleştirmesi (Order-preserving Pattern Matching – OPPM) yöntemi

uygulanmıştır. Dinamik sıkıştırma algoritmaları bağlam modellemesi yaklaşımı ile uygulanmaktadır. Model, sıkıştırılacak veriyi oluşturacak kaynağın temsilidir. Modelleme, bu temsili oluşturan süreçtir. Kodlama işlemi ise, modelleyicinin temsil ettiği kaynağı sıkıştırılmış olarak sunma işlemidir. Bağlam modellemesi, mevcut karakterin kodlamasını belirlemek için daha önce görülen karakterlerin içeriğinin kullanılmasıdır. Bağlam modellemesinde model sayısının artması girdi olan karakterin daha düşük sayıda bit ile temsil edilmesine olanak sağlamaktadır. Tek model olması durumunda tüm girdiler aynı modeli kullanacağından her karakterin temsil bit katarı uzunluğu artış gösterebilmektedir. Model sayısının artması durumunda, girdinin hangi bağlam modelini kullanacağı problemini ortaya çıkarmaktadır. Bu durumda, girdi olan karakterin hangi bağlamı kullanacağını ön görmek için önceki k adet karakteri kullanan bir yaklaşım uygulanmaktadır. Sıra Korumalı Dizgi Eşleştirme Tabanlı Bağlam Modellemesi yaklaşımı, kodlanacak veriden önceki k adet karakterin içeriği yerine, k adet verinin sıralamasını alarak desen benzerliğine göre kullanılacak bağlamı belirlemeye çalışmaktadır. Böylece daha geniş sıra düzeni kullanılmasına rağmen bağlam uzunluğu azalmakta ve kaynak kullanımı azalmaktadır. Bu yöntem, Dinamik Huffman Sıkıştırma Algoritması (Adaptive Huffman Coding) üzerinde ve Dinamik Aritmetik Sıkıştırma Algoritması (Adaptive Arithmetic Coding) üzerinde uygulanmıştır. Elde edilen sonuçlar standart uyarlanır algoritma sonuçları ile karşılaştırılmıştır. Daha sonra biraz daha iyileştirme yapmak için bir parametre daha eklenerek bağlam uzunluğu arttırılmıştır. Bu ek parametre, sıra dizisindeki herhangi bir elementin dizinin sonuna eklenmesi ile temin edilmiştir. Hangi elementin ekleneceği de, her bir elementin tek tek eklenmesi sonucu elde edilen değerlerin karşılaştırılması ile belirlenmiştir. Burada girdi olarak çeşitli veri külliyatlarından dosyalar kullanılmıştır. Bir takım dosya desenlerinde iyileşmeler gözlenmiştir. Bir takım dosya desenlerinde standart yöntemlerle aynı sonuçlar elde edilmiş, ancak daha düşük sayıda bağlam kullanıldığı için kaynak kullanımından tasarruf elde edilmiştir. Bir takım dosya desenlerinde ise bir kazanım elde edilmemiştir. Pozitif sonuçların elde edildiği bir çalışma gerçekleştirilmiştir.

2. SIRA KORUMALI DİZGİ EŞLEŞTİRMESİ

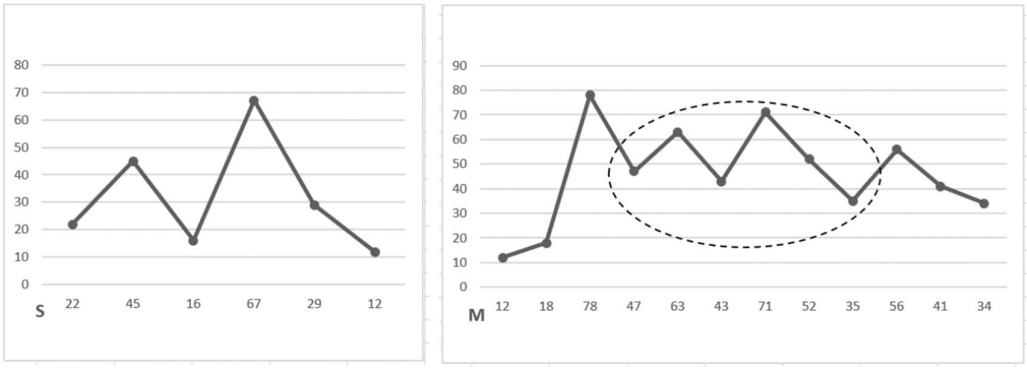
2.1 Amaç

Dizgi (string) eşleştirme işlemi bilgisayar bilimlerinde temel problemlerden bir tanesidir. Bu konuda çok fazla çalışma yapılmış ve yapılmaktadır (Amir ve diğ, 1994). Bazen dizgiler alfabetik karakterlerden değil de, nümerik karakterlerden oluşabilir. Bununla birlikte bu dizgide ilgilendiğimiz durum dizgideki belli kalıplardan ziyade dizgi içerisindeki eğilimler olabilmektedir. Mesela bir borsada analistler bir firmanın hisse senetleri ile ilgili olarak, 10 gün boyunca düşüş gösterip sonraki 5 gün boyunca artış gösteren bir periyot olup olmadığını öğrenmek isterler. Aynı şekilde müzik notalarındaki melodi benzerliği araştırmasında, bir müzisyen yaptığı yeni melodiye benzer tanınmış bir melodi olup olmadığını öğrenmek isteyebilir. Nota değerlerinin değiştiği, ancak sıralamasının aynı olduğu çok fazla çeşitleme mevcuttur. Bu durumda melodi notalarını eşleştirmek yerine, onların sıralamasını eşleştirmek gerçek bir çözüm olacaktır. Sıra Korumalı Desen Eşleştirme yöntemi bu iki duruma da çözüm getirmektedir. Bu yöntemde dizgide belirtilen değerlerden ziyade, bunların sıraları arasındaki ilişkiler analiz edilerek bir istatistik elde edilmektedir. Sıra korumalı eşleştirme, hisse senedi fiyat analizi ve dizi elemanlarının kendilerinin yerine sıra ilişkilerinin eşleştirilmesi gereken müzikal melodi eşleştirmesi gibi birçok senaryo için geçerlidir (Cambouropoulos ve diğ, 2002). Sıra koruma eşleştirmesini çözme, sayısal bir dizgenin düzen ilişkilerinin temsiliyle yakından ilgilidir. Sayısal bir dizgideki her karakteri dizedeki rütbesiyle değiştirirsek, sıra ilişkilerinin (doğal) bir gösterimini elde edebiliriz. Ancak bu doğal temsil, verimli algoritmalar geliştirmeye elverişli değildir, çünkü bir karakterin sırası, rütbenin hesaplandığı alt dizgiye bağlıdır (Aho., 2014).

2.2 Yöntemin Uygulanması

Sıra Korumalı Desen Eşleştirme yönteminin nasıl uygulandığı örnek bir veri ile gösterilmektedir. S = { 22, 45, 16, 67, 29, 12 } kalıbı M metninin bir parçası olan {

47, 63, 43, 71, 52, 35 } metin parçası ile aynı sıralama desenine sahip olduğu için eşleşmektedir.



Şekil 2.1 : Sıra korumalı dizgi eşleştirme örneği.

Şekil 2.1 ‘de görüldüğü gibi S kalıbına ait sıra deseni M metni içerisinde yer almaktadır. S dizgisinin sıralama değeri { 3, 5, 2, 6, 4, 1 } şeklindedir. M metnindeki [4-9] karakterleri arasındaki sıralama değeri de { 3, 5, 2, 6, 4, 1 } şeklindedir. Bunların dizgideki değerleri farklı olmasına rağmen sıralama desenlerinin aynı olması bunlar için bir eşleşme durumu ortaya çıkarıyor (Crochemore ve diğ, 2013).

2.3 Dizgi Uzunluğu k-Sıra

Sıra Korumalı Desen Eşleştirme yönteminde desen için belirlenen bir uzunluk bulunmaktadır. Bu uzunluk k-Sıra (k-order) olarak ifade edilmektedir. Desen uzunluğu 5 ise, metin içerisindeki ilk 5 karakterin sırlaması yapılıp desen ile eşleşip eşleşmediğine bakılır. İndeks her defasında bir kaydırılarak metnin sonuna kadar gidilir ve eşleşme sonuçları belirlenir. Şekil 2.1’deki örnekte 6-Sıra (6-order) deseni kullanılmıştır. Desen uzunluğu, yöntemin uygulandığı probleme göre en uygun değer olarak belirlenir (Kim ve diğ, 2013).

3. VERİ SIKIŞTIRMA ALGORİTMALARI

3.1 Veri Sıkıştırma

Veri sıkıştırma, verileri temsil etmek için gereken bit sayısındaki bir azalmadır. Verileri sıkıştırmak depolama kapasitesinden tasarruf sağlayabilir, dosya aktarımını hızlandırabilir ve depolama donanımı ve ağ bant genişliği maliyetlerini düşürebilir. Sıkıştırma, verilerin boyutunun nasıl küçüleceğini belirlemek için bir formül veya algoritma kullanan bir program tarafından gerçekleştirilir. Örneğin, bir algoritma, aralarındaki dönüşüm için bir sözlük kullanarak daha küçük bir 0 lar ve 1 lerden oluşan bir bit dizisini temsil edebilir veya bir formül, programın önceden öngördüğü 0'lar ve 1'lerden oluşan bir dizgeye bir referans veya işaretçi ekleyebilir.

Metin sıkıştırma, gereksiz tüm karakterleri kaldırmak, tekrarlanan karakterlerden oluşan bir dizgiyi belirtmek için tek bir yinelenen karakter eklemek ve sıklıkla oluşan bir bit dizgisi yerine daha küçük bir bit dizgisi kullanmak kadar basit olabilir. Veri sıkıştırma, bir metin dosyasını %50'ye veya orijinal boyutunun çok daha yüksek bir yüzdesine indirebilir.

Veri iletimi için, veri içeriği üzerinde veya başlık verileri dahil tüm iletim biriminde sıkıştırma gerçekleştirilebilir. Bilgi, internet yoluyla gönderildiğinde veya alındığında, tek başına veya bir arşiv dosyasının parçası olarak ZIP, GZIP veya başka bir sıkıştırılmış biçimde iletilebilir.

Veri sıkıştırma, bir dosyanın kapladığı depolama miktarını önemli ölçüde azaltabilir. Örneğin, 2:1 sıkıştırma oranında, 20 megabayt (MB) bir dosya 10 MB alan kaplar. Sıkıştırma sonucunda yöneticiler daha az para harcarlar ve depolamaya daha az zaman harcarlar.

Sıkıştırma, yedekleme depolama performansını optimize eder ve yakın zamanda birincil depolama verisinde azaltmada görüldü. Sıkıştırma, verilerin katlanarak büyümeye devam etmesi için önemli bir veri azaltma yöntemi olmaktadır.

Neredeyse her tür dosya sıkıştırılabilir, ancak hangisinin sıkıştırılacağını seçerken en iyi uygulamaları takip etmek önemlidir. Örneğin bazı dosyalar zaten sıkıştırılmış olabilir, bu nedenle bu dosyaları sıkıştırmanın önemli bir etkisi olmaz.

Verileri sıkıştırmak kayıpsız veya kayıplı bir işlem olabilir. Kayıpsız sıkıştırma, dosyanın sıkıştırılmamış olduğu bir dosyanın tek bir veri kaybı olmadan orijinal durumuna geri yüklenmesini sağlar. Kayıpsız sıkıştırma, sözcüklerin veya sayıların kaybının bilgiyi değiştireceği, çalıştırılabilir yöntemlerin yanı sıra metin ve elektronik tablo dosyalarındaki tipik yaklaşımdır.

Kayıplı sıkıştırma, gereksiz, önemsiz veya algılanamayan veri bitlerini kalıcı olarak ortadan kaldırır. Kayıplı sıkıştırma, bazı veri bitlerinin kaldırılmasının, içeriğin temsili üzerinde çok az veya hiç fark edilmeyen etkisinin olduğu grafik, ses, video ve görüntülerde kullanışlıdır.

Grafik görüntü sıkıştırması kayıplı veya kayıpsız olabilir. Grafik görüntü dosyası formatları, dosyalar büyük olma eğiliminde olduğundan, genellikle bilgileri sıkıştırmak için tasarlanmıştır. JPEG, kayıplı görüntü sıkıştırmayı destekleyen bir görüntü dosyası formatıdır. GIF ve PNG gibi formatlar kayıpsız sıkıştırma kullanır.

Sıkıştırma sıklıkla veri tekilleştirmesiyle karşılaştırılır, ancak iki teknik farklı çalışır. Veri tekilleştirme, bir depolama veya dosya sisteminde yedek veri yığınları arayan ve ardından yinelenen her bir parçanın orijinalini işaretçisi ile değiştiren bir sıkıştırma türüdür. Veri sıkıştırma algoritmaları bir veri akışındaki bit dizelerinin boyutunu azaltır.

Dosya düzeyinde veri tekilleştirme, gereksiz dosyaları ortadan kaldırır ve orijinal dosyaya işaret eden çubuklarla değiştirir. Blok düzeyinde veri tekilleştirme, alt dosya seviyesindeki çift verileri tanımlar. Sistem her bloğun benzersiz örneklerini kaydeder, bunları işlemek için bir karma algoritması kullanır ve onları bir dizinde depolamak için benzersiz bir tanımlayıcı oluşturur. Veri tekilleştirme tipik olarak sıkıştırma işleminden daha büyük çift veri yığınları arar. Sistemler sabit veya değişken boyutta bir yığın kullanarak veri tekilleştirebilir.

Veri tekilleştirme, sanal masaüstü altyapısı veya depolama yedekleme sistemleri gibi çok fazla yedekli veriye sahip ortamlarda en etkilidir. Veri sıkıştırma, resimler, ses, videolar, veri tabanları ve yürütülebilir dosyalar gibi benzersiz bilgilerin boyutunu azaltmada veri tekilleştirmeden daha etkili olma eğilimindedir. Çoğu depolama sistemi hem sıkıştırmayı hem de veri tekilleştirmeyi destekler.

Sıkıştırma çoğu zaman, erişilemeyen veriler için kullanılır, çünkü işlem yoğun olabilir ve sistemleri yavaşlatır. Ancak yöneticiler sıkıştırmayı yedekleme sistemlerine sorunsuz bir şekilde entegre edebilirler.

İşlem sırasında sistem aynı dosyaları sık sık yakaladığı için yedekleme, gereksiz bir iş yükü türüdür. Tam yedekleme yapan bir kuruluşta genellikle yedeklemeden yedeklemeye hemen hemen aynı verilere veya yakın verilere rastlanır.

Yedeklemeden önce verileri sıkıştırmanın büyük yararları vardır:

- Sıkıştırma oranı 100:1'e ulaşabileceği için veriler daha az yer kaplar, ancak 2:1 ile 5:1 arasında bir oran daha sık görülür.
- İletimden önce bir sunucuda sıkıştırma yapılırsa, verileri iletmek için gereken süre ve toplam ağ bant genişliği önemli ölçüde azaltılır.
- Teypte, sıkıştırılmış, daha küçük dosya sistemi görüntüsü, belirli bir dosyaya erişmek için daha hızlı taranarak geri yükleme gecikmesini azaltır.
- Sıkıştırma, yedekleme yazılımı ve teyp kitaplıkları tarafından desteklenir, bu nedenle bir dizi veri sıkıştırma tekniği vardır.

Sıkıştırmanın ana avantajları, depolama donanımındaki, veri iletim zamanındaki ve iletişim bant genişliğindeki bir azalma ve bunun sonucunda maliyet tasarrufudur. Sıkıştırılmış bir dosya, sıkıştırılmamış bir dosyaya göre daha az depolama kapasitesi gerektirir ve sıkıştırma kullanımı, disk ve / veya yarıiletken sürücüler için harcamalarda önemli bir düşüşe yol açabilir. Sıkıştırılmış bir dosya aynı zamanda aktarım için daha az zaman gerektirir ve sıkıştırılmamış bir dosyadan daha az ağ bant genişliği kullanır.

Veri sıkıştırmanın en büyük dezavantajı, verileri sıkıştırmak ve sıkıştırmayı açmak için CPU ve bellek kaynaklarının kullanılmasından kaynaklanan performans etkisidir. Pek çok satıcı, sistemlerini sıkıştırma ile ilgili işlemci yoğun hesaplamalarının etkisini en aza indirmeye çalışmak için tasarladı. Sıkıştırma satır içi çalışıyorsa, veriler diske yazılmadan önce, sistem kaynaklarını korumak için sistem sıkıştırmayı yapacağı alanı boşaltabilir. Örneğin, IBM, bazı kurumsal depolama sistemlerinde sıkıştırmayı yönetmek için ayrı bir donanım hızlandırma kartı kullanır.

Veriler diske yazıldıktan sonra veya işlem sonrası sıkıştırılırsa, performans etkisini azaltmak için sıkıştırma arka planda çalışabilir. İşlem sonrası sıkıştırma her giriş / çıkış için yanıt süresini azaltabilse de (G / Ç), yine de bellek ve işlemci döngülerini tüketir ve bir depolama sisteminin kaldırabileceği genel G / Ç sayısını etkileyebilir. Ayrıca, başlangıçta verilerin diske veya flash

sürücülere sıkıştırılmamış bir biçimde yazılması gerektiğinden, fiziksel depolama tasarrufu satır içi sıkıştırmada olduğu kadar büyük değildir.

Dosya sistemi sıkıştırması, her bir dosyayı yazıldığı gibi saydam bir şekilde sıkıştırarak verilerin depolama alanını azaltmak için oldukça basit bir yaklaşım izler. Popüler Linux dosya sistemlerinin çoğu - Reiser4, ZFS ve btrfs dahil – ve de Microsoft NTFS, bir sıkıştırma seçeneğine sahiptir. Sunucu bir dosyadaki veri parçalarını sıkıştırır ve daha sonra küçük parçaları depoya yazar. Geri okuma, her bir parçayı genişletmek için nispeten küçük bir gecikme süresi içerirken, yazarken sunucuya önemli bir yük eklenir, bu nedenle sıkıştırma genellikle geçici olan veriler için önerilmez. Dosya sistemi sıkıştırması performansı zayıflatabilir, bu yüzden sık sık erişilmeyen dosyalara seçici olarak dağıtılmalıdır. Tarihsel olarak, eski bilgisayarların pahalı sabit diskleriyle, DiskDoubler ve SuperStor Pro gibi veri sıkıştırma yazılımlarının popülerliği ana dosya sistemi sıkıştırmasının kurulmasına yardımcı oldu. Depolama yöneticileri, daha iyi veri azaltma için sıkıştırma ve veri tekilleştirme tekniğini de uygulayabilir.

Sıkıştırma, depolama sistemleri, veri tabanları, işletim sistemleri ve işletmeler ve kurumsal kuruluşlar tarafından kullanılan yazılım uygulamaları dahil olmak üzere çok çeşitli teknolojilerde yerleşiktir. Veri sıkıştırma, dizüstü bilgisayarlar, PC'ler ve cep telefonları gibi tüketici cihazlarında da yaygındır.

Birçok sistem ve cihaz şeffaf şekilde sıkıştırma yapar, ancak bazıları kullanıcılara sıkıştırmayı açma veya kapatma seçeneği sunar. Sıkıştırma işlemi aynı dosyada veya veri parçası üzerinde bir kereden fazla gerçekleştirilebilir, ancak daha sonraki sıkıştırmalar çok az ek maliyet getirir ve dosya sıkıştırma boyutunu veri sıkıştırma algoritmalarına bağlı olarak biraz daha iyileştirebilir.

WinZip, bir arşive paketlendiğinde dosyaları sıkıştıran popüler bir Windows programıdır. Sıkıştırmayı destekleyen arşiv dosya formatları ZIP ve RAR' dır. BZIP2 ve GZIP formatlarının, dosyaları tek tek sıkıştırmak için yaygın olarak kullanıldığı görülür. Sıkıştırma sunan diğer satıcılar arasında, XtremIO all-flash dizili Dell EMC, K2 all-flash dizili Kaminario ve veri sıkıştırma yazılımı ile RainStor yer alıyor.

Veri farklılığı, iki veri nesnesinin içeriğini karşılaştırmak için kullanılan genel bir terimdir. Sıkıştırma bağlamında, benzer blokları bulmak için hedef dosyada tekrar tekrar aranmayı ve

bunların bir kütüphane nesnesine bir referansla değiştirilmesini içerir. Bu işlem, ek yinelenen nesne bulana kadar tekrar eder. Veri farklılaşması, kütüphanede çoğaltılan her nesneyi temsil eden yalnızca bir öğeli birçok sıkıştırılmış dosyaya neden olabilir.

Sanal masaüstlerinde, bu teknik 100: 1 kadar sıkıştırma oranına sahip olabilir. İşlem genellikle her bir nesnenin içeriğinden ziyade aynı dosyaları veya nesneleri arayan veri tekilleştirme ile daha uyumludur. Veri farklılığı bazen veri tekilleştirme olarak adlandırılır.

3.2 Kayıpsız Veri Sıkıştırma Yaklaşımları

Kayıpsız veri sıkıştırmaları statik ve dinamik olarak iki farklı şekilde yapılabilmektedir. Statik veri sıkıştırmalarında tüm veri üzerinden geçilerek sembollerin kullanıma göre bir veri modeli oluşturulur. Sonrasında tüm veri bu modele göre kodlanır. Verinin geri çözümlenmesi de aynı şekilde olur. İlk olarak kodlanırken kullanılan veri modeli alınır, sonrasında tüm veri bu modele göre çözülür. Ancak telekomünikasyon gibi bazı alanlarda verinin tamamı henüz ulaşmamış olduğundan tüm verinin üzerinden geçmek mümkün değildir. Bunlar için dinamik veri sıkıştırma algoritmaları kullanılır.

Dinamik veri sıkıştırma algoritmalarında başlangıç olarak bir veri modeli belirlenir, gelen veri bu model ile kodlanır. Veri modeli son kodlanan veri bilgisi eklenerek güncellenir. Verinin geri çözümlenmesi sırasında yine başlangıç olarak bir veri modeli belirlenir. Gelen veri modele göre çözümlenir, sonrasında veri modeli güncellenir. Yapılan çalışmada kullanılan Sıra Korumalı Dizgi Eşleştirme Tabanlı Bağlam Modellemesi yaklaşımı, dinamik kayıpsız veri sıkıştırma yöntemlerinden Huffman Dinamik Kodlama ve Aritmetik Dinamik Kodlama yöntemleri üzerine uygulanmıştır.

3.3 Statik Veri Sıkıştırma

Statik veri sıkıştırma, bir veri içindeki birden fazla değeri kapsayan tekrar eden modellerin daha kısa sembol dizeleriyle değiştirilmesini içerir. Tekrarlanan verilerin örneklerini bulmak için bir veri örneklemesi taranır. Bu taramadan, dosya veya tüm veri düzeyinde bir sıkıştırma sözlüğü oluşturulur. Bu sözlük tekrarlayan verileri daha kısa sembol dizeleriyle değiştirmek için kullanılır. Dosya düzeyinde sıkıştırma sözlükleri statiktir; İlk oluşturulduktan sonra, klasik bir tablo düzenleme sırasında bunları yeniden oluşturmadıkça değişmezler. Klasik veri

sıkıştırması, verileri karakter karakter sıkıştırmak için dosya düzeyinde bir sıkıştırma sözlüğü kullanır. Sözlük, tekrarlanan byte desenlerini metin içerisindeki temsilinden çok daha küçük sembollere eşlemek için kullanılır; bu semboller daha sonra tüm veri içerisindeki daha uzun byte desenlerini değiştirir. Sıkıştırma sözlüğü, tüm veri nesnesi ile beraber sıkıştırılmış veri ile birlikte depolanır (Amir ve diğ, 2009).

3.4 Dinamik Veri Sıkıştırma

Dinamik sıkıştırma, klasik statik sıkıştırma özelliğini kullanarak elde edilebilecek sıkıştırma oranlarını geliştirir. Uyarlanabilir sıkıştırma da denilen bu yöntem, klasik sıra sıkıştırmayı içerir; ancak, verileri daha da sıkıştırmak için sayfa-sayfa esasına göre çalışır. Veri sıkıştırma tekniklerinden uyarlamalı sıkıştırma, depolama tasarrufu için en çarpıcı olanakları sunar. Uyarlanabilir sıkıştırmada aslında iki sıkıştırma yaklaşımı kullanır. Birincisi, klasik statik veri sıkıştırmasında kullanılan, aynı dosyadaki verilerin bir örneklemesinde tekrarlamaya dayalı verileri sıkıştırmak için kullanılan aynı sıkıştırma sözlüğünü kullanır.

İkinci yaklaşım, her veri sayfasındaki veri tekrarına dayanarak verileri sıkıştırmak için sayfa düzeyinde bir sözlük tabanlı sıkıştırma algoritması kullanır. Sözlükler tekrarlanan bayt desenlerini çok daha küçük sembollere eşler; bu semboller daha sonra dosyadaki daha uzun bayt modellerini değiştirir. Dosya düzeyinde sıkıştırma sözlüğü, oluşturulduğu dosyanın içinde depolanır ve tablodaki verileri sıkıştırmak için kullanılır. Sayfa düzeyinde sıkıştırma sözlüğü, veri sayfasındaki verilerle birlikte depolanır ve yalnızca o sayfadaki verileri sıkıştırmak için kullanılır.

3.5 Huffman Kodlama

Huffman algoritması, bir veri kümesinde daha çok rastlanan sembolü daha düşük uzunluktaki kodla, daha az rastlanan sembolleri daha yüksek uzunluktaki kodlarla temsil etme mantığı üzerine kurulmuştur. Bir örnekten yola çıkacak olursak: Bilgisayar sistemlerinde her bir karakter 1 byte yani 8 bit uzunluğunda yer kaplar. Yani 10 karakterden oluşan bir dosya 10 byte büyüklüğündedir. Çünkü her bir karakter 1 byte büyüklüğündedir. Örneğimizdeki 10 karakterlik veri kümesi “aaaaaaaccs” olsun. “a” karakteri çok fazla sayıda olmasına rağmen “s” karakteri tektir. Eğer bütün karakterleri 8 bit değil de veri kümesindeki sıklıklarına göre

kodlarsak veriyi sembolize etmek için gereken bitlerin sayısı daha az olacaktır. Söz gelimi "a" karakteri için "0" kodunu "s" karakteri için "10" kodunu, "c" karakteri için "11" kodunu kullanabiliriz. Bu durumda 10 karakterlik verimizi temsil etmek için;

(a kodundaki bit sayısı)x(verideki a sayısı) + (c kodundaki bit sayısı)x(verideki c sayısı) + (s kodundaki bit sayısı)x(verideki s sayısı) = 1x7 + 2x2 + 2x1 = 12 bit

gerekecektir. Halbuki bütün karakterleri 8 bit ile temsil etseydik 8x10 = 80 bite ihtiyacımız olacaktı. Dolayısıyla %80 'in üzerinde bir sıkıştırma oranı elde etmiş olduk.

Huffman Kodlama algoritması sıkıştırılacak veriyi tarayarak, veri içerisindeki sembollerin kullanım sıklığını hesaplar. Daha sonra bunlardan dengeli bir ağaç oluşturur. Sağ taraftaki dal 0 biti, sol taraftaki dal da 1 biti ile işaretlenir. Ağaç üzerindeki tüm yapraklara inilir. Her bir yaprak bir sembolü temsil eder. Böylelikle her sembolün bir bit katarı oluşmuş olur. Kullanımı en fazla olan semboller en düşük uzunluklu bit katarı ile temsil edilir (Larmore ve Hirschberg, 1990).

Dinamik Huffman Kodlamada ise tüm veri taranmaz. Gelen veriye göre Huffman kodlama ağacı sürekli güncellenir. Sıkıştırılan veri tekrar açık hale dönüştürülürken de aynı şekilde sembol açık veriye dönüştürüldükçe kodlama ağacı güncellenir. Sıkıştırılan veri çözümlenirken bir ön bilgiye ihtiyaç duyulmaz. Bu yöntemin avantajı gerçek zamanlı veri iletişimini sağlamasıdır.

Huffman Kodlamada ağaç oluşturulurken, en düşük frekanslı son 2 sembolün aynı seviyede olması gerekmektedir. Önce Huffman Kodlamanın nasıl çalıştığını gösteren aşağıdaki örneği inceleyelim.

Sıkıştırılacak veri katarımız $S = \{ S_1, S_2, S_3, S_4, S_5 \}$ olsun. Bunların frekanslarından elde edilen olasılık değerleri de;

$P_1 = P_3 = 0.2 \qquad P_2 = 0.4 \quad P_4 = P_5 = 0.1 \quad , \; P_i = P(S_i)$

şeklinde bulunmuş olsun.

En düşük 2 sembol S_4 ve S_5 aynı seviyede olacağı için bunlara S_4' olarak gruplayabiliriz.

$P(S_4') = 0.1 + 01 = 0.2$ olur.

$P(S_1) = 0.2 \quad P(S_3) = 0.2 \quad P(S_2) = 0.4 \quad P(S_4') = 0.2$

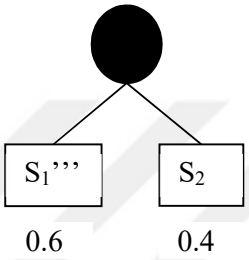
Yeni durumda en düşük frekanslı iki sembol S_3 ve S_4' alınır. Bunu S_3'' olarak gruplayabiliriz.

$P(S_3'') = 0.2 + 02 = 0.4$ olur.

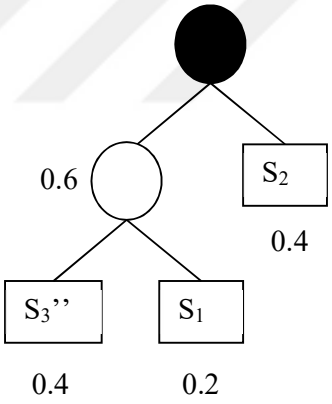
$P(S_1) = 0.2 \quad P(S_3'') = 0.4 \quad P(S_2) = 0.4$

Tekrar en düşük frekanslı iki sembol S_3'' ve S_1 alınır. S_1''' olarak gruplanır.

$P(S_1''') = 0.6 \quad P(S_2) = 0.4$ değerleri için ikili ağaç oluşturulur. İkili ağaç oluşturulurken yüksek değer sol yaprağa, düşük değer sağ yaprağa bağlanır.



Şekil 3.1 : Huffman ikili kod ağacı-1.



Şekil 3.2 : Huffman ikili kod ağacı-2.

Şekil 3.4'de gösterilen ikili ağaç oluşturulduktan sonra sol dala 0, sağ dala 1 değeri verilerek tüm yapraklara inilir. Böylece her bir yaprağın (sembolün) bir kodu temin edilmiş olur.

Son durumda Şekil 3.5'de görüldüğü gibi;

S2 : 1

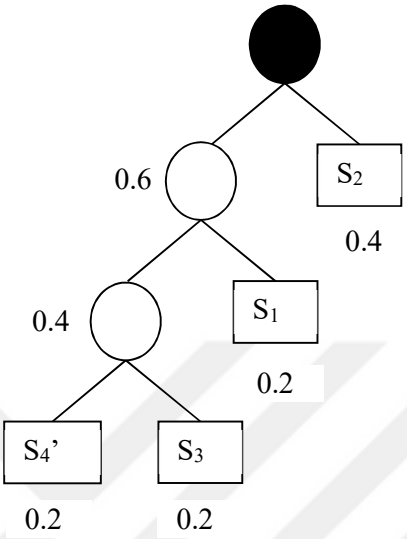
S1 : 01

S3 : 001

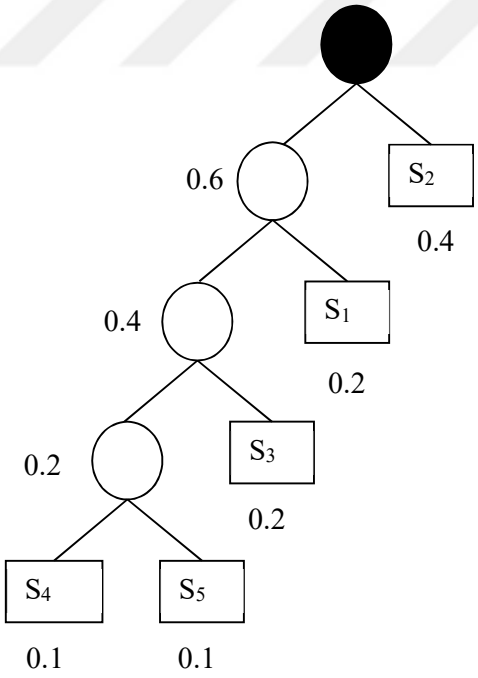
S4 : 0010

S5 : 0011

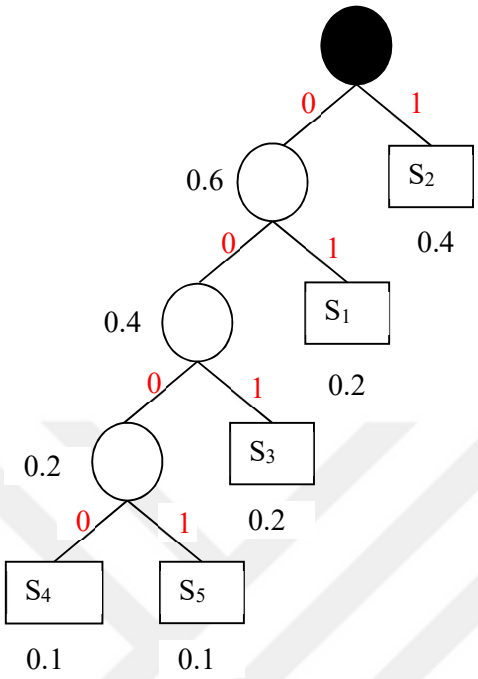
şeklinde elde edilmiş olur. Bu bilgi saklanır ve veri çözümlemede kullanılır.



Şekil 3.3 : Huffman ikili kod ağacı-3.



Şekil 3.4 : Huffman ikili kod ağacı-4.



Şekil 3.5 : Huffman ikili kod ağacı-5.

Dinamik Huffman yönteminde ise elimizde henüz istatiksel bilgi bulunmamaktadır. Bir başlangıç algoritmasına göre başlangıç ağacı oluşturulur. Gerçek zamanlı iletişim örneği üzerinden gittiğimizde gönderici de (kodlayan), alıcı da (çözümleyen) aynı başlangıç ağacını oluşturur. Daha sonra veri geldikçe ikili ağaç güncellenir ve buna göre her gelen sembol güncel ağaç üzerindeki değerlere göre kodlanır ve çözümlenir. Dinamik Huffman kodlaması 3 temel işlem üzerine kuruludur (Pigeon ve Bengio, 1997).

- GÜNCELLEME (UPDATE)
- KODLAMA (ENCODING)
- ÇÖZÜMLEME (ÇÖZÜMLEME)

Güncelleme işlemi hem kodlama ve hem de çözümleme için aynıdır.

Bir örnek üzerinden bu adımlar uygulanarak açıklanmaktadır. İlk olarak başlangıç kodlama değerleri oluşturulmaktadır. Örneğimizde alfabe boyutu m = 26 şeklinde olsun.

$S = \{ S_1\ S_2 \ldots S_m \}$ $m = 26$

Daha sonra aşağıdaki kurala uyacak e ve r değerleri seçilir.

$m = 2^e + r$ ve $0 \leq r < 2^e$ olacak şekilde e ve r değerleri belirlenir.

S_k nın kod katarı şu şekilde belirlenir:

- Eğer $1 \leq k \leq 2r$ ise, k-1 değerini ikili tabanda (e + 1) bit uzunluklu olarak belirle.
- k belirtilen aralıkta değilse k-r-1 değerini e bit uzunluklu olarak belirle

Mesela; m = 26 olduğunda, e = 4 ve r = 10 oluyor.

$S_1 \rightarrow$ 00000 oluyor, k = 1 değeri $1 \leq k \leq 2r$ aralığında olduğu için k-1 = 0 değeri ikili tabanda e + 1 = 5 bit uzunluğunda kodlanıyor.

$S_{20} \rightarrow$ 10011 oluyor, k = 20 değeri $1 \leq k \leq 2r$ aralığında olduğu için k-1 = 19 değeri ikili tabanda e + 1 = 5 bit uzunluğunda kodlanıyor.

$S_{21} \rightarrow$ 1010 oluyor, k = 20 değeri $1 \leq k \leq 2r$ aralığında olmadığı için k-r-1 = 10 değeri ikili tabanda e = 4 bit uzunluğunda kodlanıyor.

Bu hesaplamaya göre Dinamik Huffman Kodlama metodunda hem kodlama aşamasında hem çözümleme aşamasında elimizde olacak kod bilgileri Çizelge 3.1' de gösterildiği şekildedir.

Gönderici ve alıcı tarafta bu kod değerleri başlangıç olarak belirlenir. Bu kod değerlerinin olduğu listeye 'Henüz İletilmemişler' (NYT : Not Yet Transmitted) listesi denilir. Bir sembole ilk defa rastlandığında, bu sembol NYT listesinden alınır ve ikili ağaçtaki yerine yerleştirilir. Böylece ağaç güncellenmiş olur ve bu sembole bir sonraki rastlamada ikili ağaçtaki yeni kod değeri kullanılır. Bu akış gönderici ve alıcı tarafında, yani kodlama ve çözümleme akışlarında, aynı şekilde uygulanmaktadır. Böylece alıcı ve gönderici tarafı eş zamanlı olmuş olur (Siemifiski, 1998).

GÜNCELLEME (UPDATE) işleminde akış aşağıdaki şekilde olur:

- Her düğüm ağırlığına göre sağa ve sola düzgün bir sırada yerleştirilir.
- Burada ağırlığı sembolün frekansı oluşturur. Düşük ağırlık sola, yüksek ağırlık sağa yazılır.
- Yapılan numaralandırma ile bu sıralama korunmuş (sbling property) olur.
- En yüksek Numara kök (root) düğüme verilir.

- En düşük Numara ise NYT düğümüne verilir.
- Soldan sağa ve aşağıdan yukarıya artan şekilde numaralandırma yapılır.
- Aynı ağırlıktaki düğümler bir blok oluşturur.
- Gönderici tarafda kodlama işlemi yapıldıktan sonra, alıcı tarafta da çözümleme işlemi yapıldıktan sonra t anındaki kodlama ağacı güncellenir.
    - Gelen sembol ilk defa geliyorsa ağaçta uygun yere yeni düğüm eklenir ve NYT listesinden bu sembol çıkarılır.
    - İlk defa gelmiyorsa ilgili düğüme gidilir ve ağırlığı (frekansı) güncellenir.
    - Eğer bu sembolün düğümü, bloktaki en yüksek numaraya sahip değilse, bu düğüm bloktaki diğer düğüm ile yer değiştirir. Aksi durumda sadece ağılığı güncellenir.

Kök düğüme ulaşıncaya kadar bu adımlar tekrarlanır ve böylece ağaç güncellenmiş olur.

KODLAMA (ENCODING) işlemi ise aşağıdaki gibidir:

- NYT düğümüne 2m-1 uzunluk değeri atanır (m = alfabe boyu)
- Bir sembol geldiğinde NYT listesine bakılır, eğer sembol burda ise başlangıçta kendisine atanan kodlama değeri alınarak gönderilecek veri katarına eklenir.
- Sembol NYT listesinde yok ise, kodlama ağacından değeri bulunur.
- Daha sonra güncelleme işlemi çağırılır.

Bir örnek ile yukarıdaki adımları uygulayalım. Alfabe uzunluğunun m = 26 olduğu bir iletişimde gelen veri sırasıyla “aardva” olsun. Burada $m = 2^e + r$ ve $0 \leq r < 2^e$ kuralından yola çıkarsak, e = 4, r = 10 değerlerini elde ederiz. Yukarıda bşlangıç değerlerinin nasıl oluştuğunun anlatıldığı örnekte de alfabe uzunluğu 26 olduğu için Çizelge 3.1 deki başlangıç değerlerini kullanabiliriz.

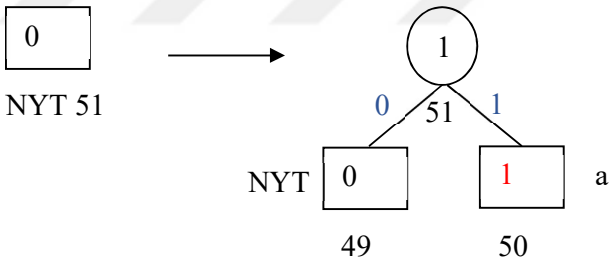
aardva , a sembolüne daha önce rastlanmadığı için NYT listesindeki değerine bakıyoruz.

k = 1 için $S_{k-1} = S_0 = 00000$ olur. Transfer edilen veri T = 00000 olur.

Daha sonra Huffman ağacı Şekil 3.6’da olduğu gibi güncellenir.

Çizelge 3.1 : Alfabe uzunluğu m = 26 olan bir veri kümesinde sembollerin başlangıç için kod değerleri.

1-7	8-14	15-21	22-26
$S_1 \rightarrow 00000$	$S_8 \rightarrow 00111$	$S_{15} \rightarrow 01110$	$S_{22} \rightarrow 1011$
$S_2 \rightarrow 00001$	$S_9 \rightarrow 01000$	$S_{16} \rightarrow 01111$	$S_{23} \rightarrow 1100$
$S_3 \rightarrow 00010$	$S_{10} \rightarrow 01001$	$S_{17} \rightarrow 10000$	$S_{24} \rightarrow 1101$
$S_4 \rightarrow 00011$	$S_{11} \rightarrow 01010$	$S_{18} \rightarrow 10001$	$S_{24} \rightarrow 1101$
$S_5 \rightarrow 00100$	$S_{12} \rightarrow 01011$	$S_{19} \rightarrow 10010$	$S_{25} \rightarrow 1110$
$S_6 \rightarrow 00101$	$S_{13} \rightarrow 01100$	$S_{20} \rightarrow 10011$	$S_{26} \rightarrow 1111$
$S_7 \rightarrow 00110$	$S_{14} \rightarrow 01101$	$S_{21} \rightarrow 1010$	



Şekil 3.6 : Gelen veri "a" durumunda Huffman ağacı : "a" NYT listesinden çıkarıldı, ağaca eklendi.

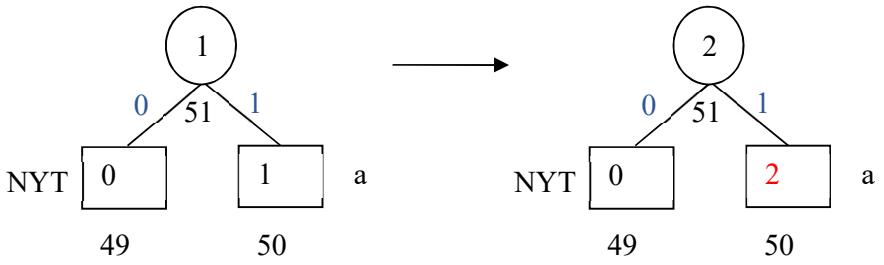
aardva , gelen ikinci "a" Huffman ağacında olan bir kod olduğu için, bunun kod değeri transfer edilen veriye T = 000001 şeklinde eklenir.

Daha sonra Huffman ağacı Şekil 3.7'de olduğu gibi güncellenir.

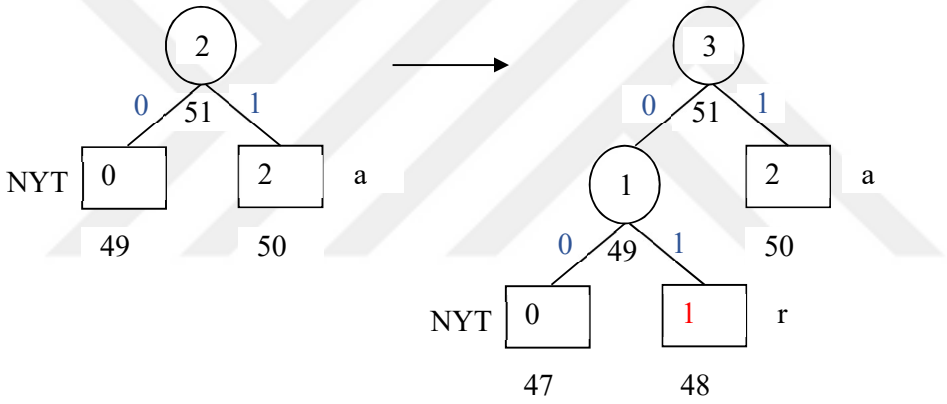
aardva , gelen r sembolüne daha önce rastlanmadığı için NYT listesindeki değerine bakıyoruz. NYT' ye gidip bakıldığı için NYT düğümünün kod değeri (0) transfer edilen veriye eklenir, T = 00000 1 0 olur.

k = 18 için $S_{k-1} = S_{17} = 10001$ olur. Transfer edilen veri T = 00000 1 0 10001 olur.

Daha sonra Huffman ağacı Şekil 3.8'de olduğu gibi güncellenir.



Şekil 3.7 : Gelen veri "aa" durumunda Huffman ağacı : "a" sembolünün ağırlığı (frekenası) güncellendi.

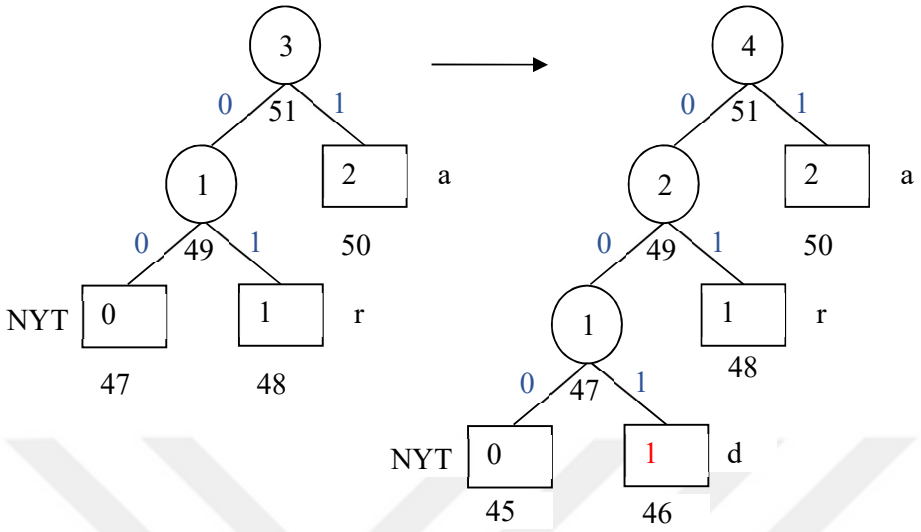


Şekil 3.8 : Gelen veri "aar" durumunda Huffman ağacı : "r" NYT listesinden çıkarıldı, ağaca eklendi.

aardva , gelen d sembolüne daha önce rastlanmadığı için NYT listesindeki değerine bakıyoruz. NYT' ye gidip bakıldığı için NYT düğümünün kod değeri (00) transfer edilen veriye eklenir, T = 00000 1 0 10001 00 olur.

k = 4 için $S_{k-1} = S_3$ = 00011 olur. Transfer edilen veri T = 00000 1 0 10001 00 00011 olur.

Daha sonra Huffman ağacı Şekil 3.9'da olduğu gibi güncellenir.



Şekil 3.9 : Gelen veri "aard" durumunda Huffman ağacı : "d" NYT listesinden çıkarıldı, ağaca eklendi.

aardva , gelen v sembolüne daha önce rastlanmadığı için NYT listesindeki değerine bakıyoruz. NYT' ye gidip bakıldığı için NYT düğümünün kod değeri (000) transfer edilen veriye eklenir, T = 00000 1 0 10001 00 00011 000 olur.

k = 22 için $S_{k-1} = S_{21}$ = 1011 olur. Transfer edilen veri T = 00000 1 0 10001 00 00011 000 1011 olur.

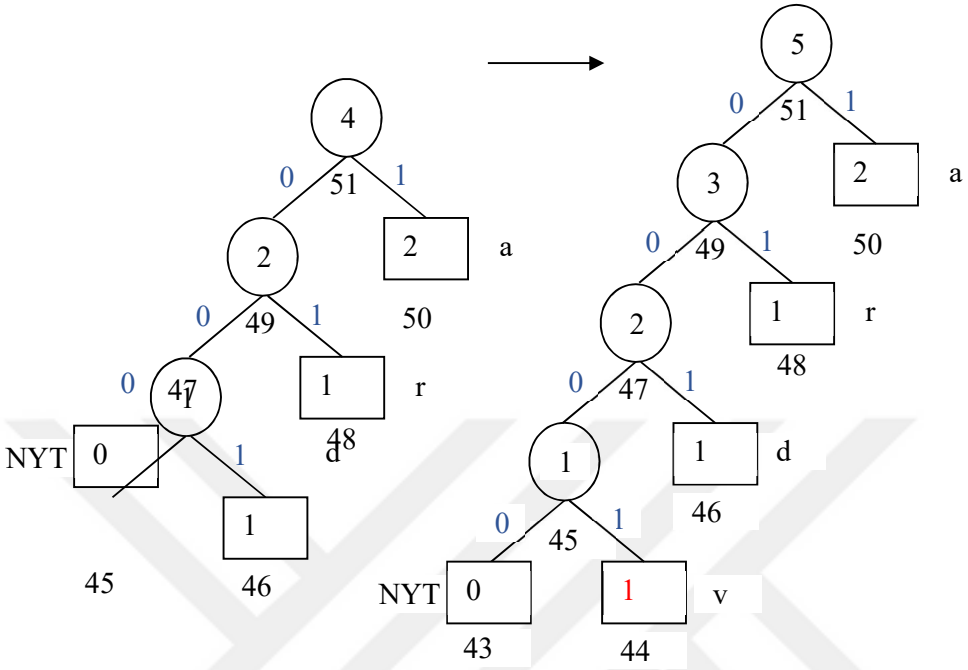
Daha sonra Huffman ağacı Şekil 3.10'da olduğu gibi güncellenir.

Güncelleme adımında bahsedildiği gibi "sibling" özelliğini korumak için her düğüme bir numara verilmektedir. Bu numaralar soldan-sağa ve aşağıdan-yukarı artan özellikte ilerler. Aynı seviyedeki en yüksek ağırlıklı düğüm daha büyük uzunluğa sahip olmalıdır. Burada "v" sembolünün eklenmesiyle birlikte bu durum iki yerde bozulmuştur. "r" sembolünün bulunduğu seviyede (2 > 1 fakat 47 < 48) ve "a" sembolünün bulunduğu seviyede (3 > 2 fakat 49 < 50). Ağaç üzerinde güncelleme yapılarak Huffman ağacının özelliği korunmuş olur. Bu durum Şekil 3.11'de gösterilmiştir.

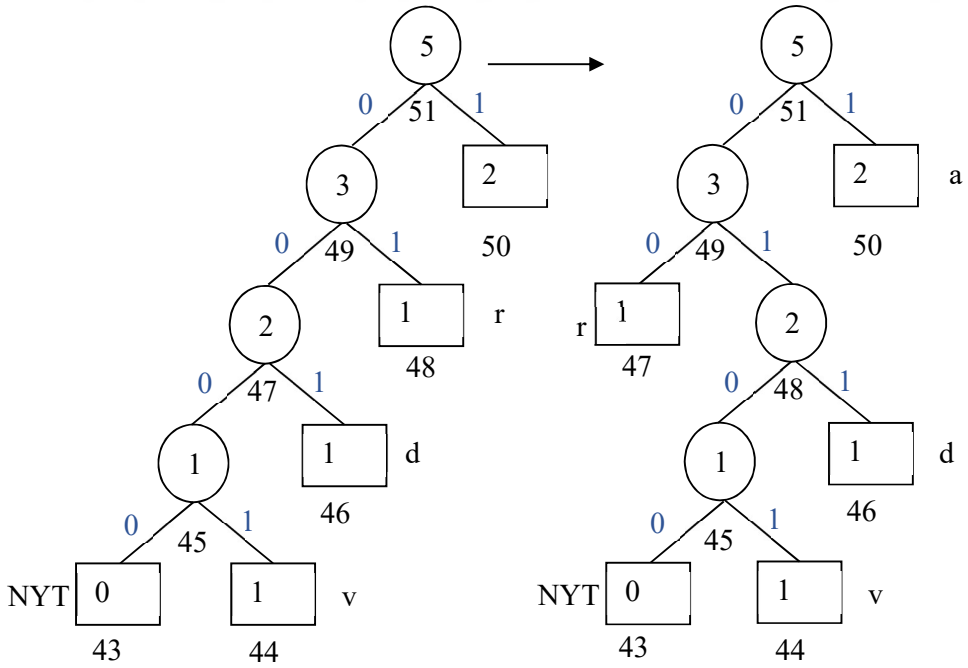
Sonrasında Şekil 3.12'de görüldüğü gibi kök düğüme kadar aynı işlem uygulanır. Diğer sorunlu olan "a" seviyesindeki düzeltme yapılır.

aardva , gelen üçüncü "a" Huffman ağacında olan bir kod olduğu için, bunun kod değeri transfer edilen veriye T = 00000 1 0 10001 00 00011 000 1011 0 olarak eklenir.

Daha sonra Şekil 3.13'de gösterildiği gibi Huffman ağacı güncellenir.



Şekil 3.10 : Gelen veri "aardv" durumunda Huffman ağacı : "v" NYT listesinden çıkarıldı, ağaca eklendi, ancak "sibling" özelliği bozuldu.



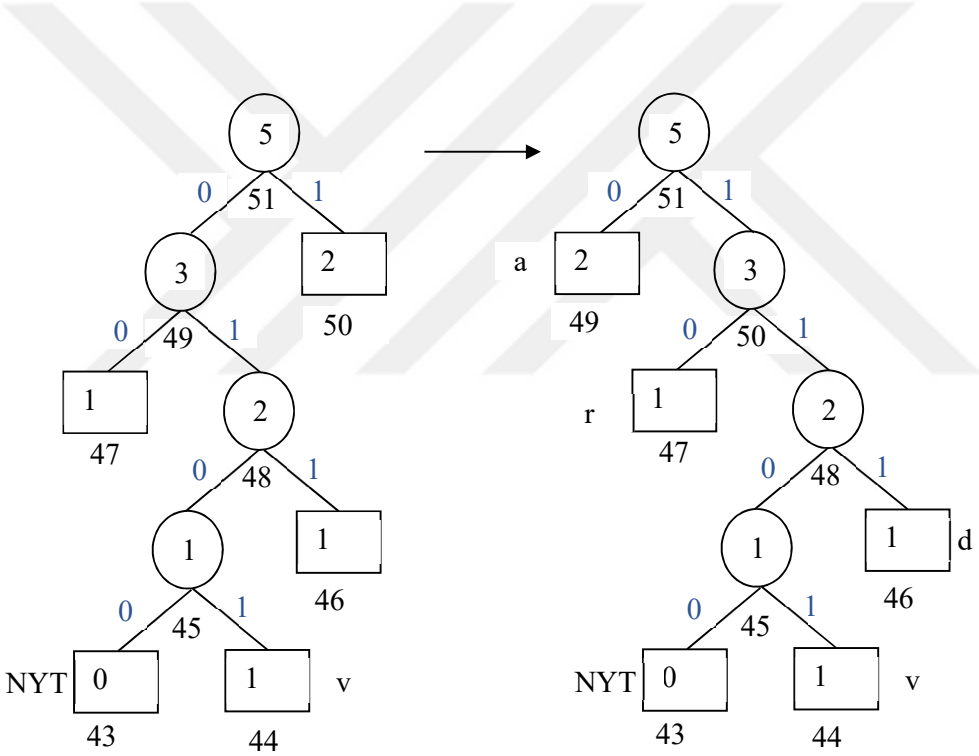
Şekil 3.11 : Gelen veri "aardr" durumunda Huffman ağacı : "d" seviyesindeki en yüksek uzunluk sorunu giderildi.

Çözümleme işlemi de kodlama işleminde olduğu gibi yapılır. Aynı Huffman ağacı elde edileceği için, gönderilen veri doğru bir şekilde çözümlenir.

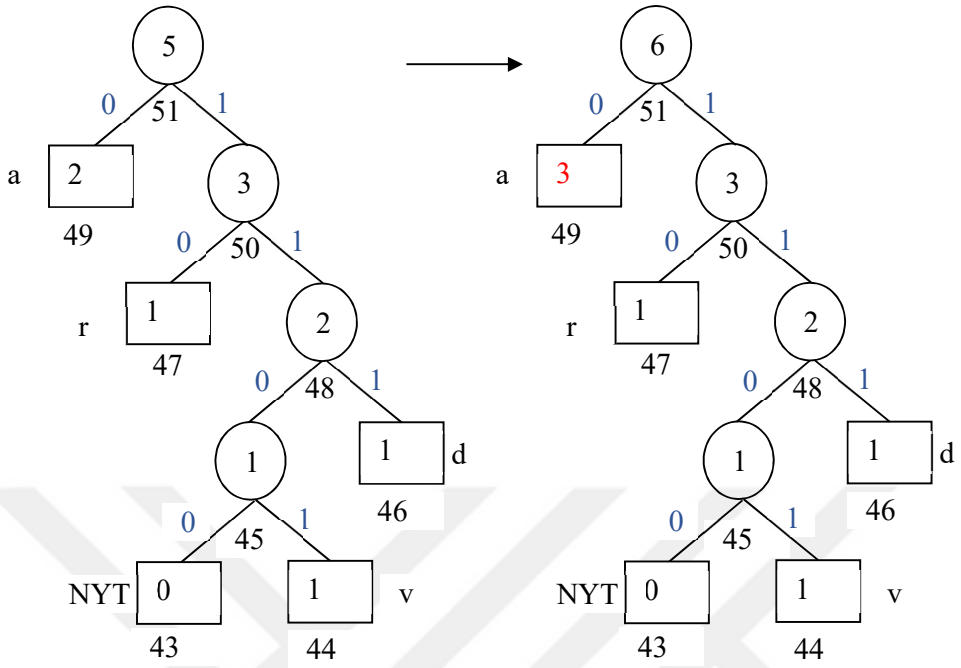
ÇÖZÜMLEME (DECODING) işleminin adımları aşağıdaki gibidir:

- NYT düğümüne 2m-1 uzunluk değeri atanır (m = alfabe boyu)
- Veri NYT listesinde bir değer bulunana kadar veya Huffman Ağacında bir değer bulunana kadar okunur.
- Elde edilen sembol çözümlenen veriye eklenir.

Daha sonra güncelleme işlemi çağırılır.



Şekil 3.12 : Gelen veri "aardr" durumunda Huffman ağacı : "a" seviyesindeki en yüksek uzunluk sorunu giderildi.



Şekil 3.13 : Gelen veri "aardra" durumunda Huffman ağacı : "a" sembolünün ağırlığı (frekenası) güncellendi.

3.6 Aritmetik Kodlama

Aritmetik Kodlama algoritması Huffman Kodlamada olduğu gibi sıkıştırılacak veriyi tarayarak, veri içerisindeki sembollerin kullanım sıklığını hesaplar. Daha sonra bunların olasılıklarını hesaplayarak her bir sembol için [0, 1) aralığında eşsiz bir aralık elde eder. Eşsiz aralık elde edildikten sonra bu aralıktan herhangi bir değer alınabilir ve o sembolün ondalıklı değeri bu olur. Gelen sembol ne ise o aralık seçilir, sonraki her sembolün aralığı kendi sıklığı oranında seçilen aralık üzerinde yeniden belirlenir. Böylelikle veri geldikçe daha dar aralıklı eşsiz değerler oluşur. Oluşan bu eşsiz ondalıklı değer ikili tabana çevrilir. Böylece kod katarı temin edilmiş olur. Çözümleme yapılırken de tersi yönde aynı yöntem izlenir. Bit katarı ondalıklı sayıya çevrilerek eşsiz etiket değeri elde edilir. Her karakterin aralığı ve frekansı (oranı) bulunur. Eşsiz etiket değeri hangi aralıkta ise, kontrol edilen sembol bulunur ve eşsiz etiket değeri geriye doğru güncellenir (Amir, 2003).

Dinamik Aritmetik Kodlamada ise tüm veri taranmaz. Gelen veriye göre sembollerin kod değerleri sürekli güncellenir. Başlangıç olarak alfabedeki her bir sembolün eşit olasılıkta olduğu varsayılır. Veri geldikçe sembollerin frekansı güncellenir ve yeni veriye göre bit

katarları yeniden oluşturulur. Sıkıştırılan veri tekrar açık hale dönüştürülürken de aynı şekilde açık veriye dönüşen sembolün frekansı güncellenir ve bit katarları yeniden oluşturulur.

Aritmetik Kodlamada sembollerin bit katarlarını belirlemek için iki temel adım bulunmaktadır:

- Eşsiz bir etiket değeri bulmak, [0,1) aralığında
- Bu etiket değerini ikili tabanda sunarak kullanmak

İlk adım olarak etiket değerinin nasıl oluşturulduğu gösterilecek, sonrasında ikili tabanda eşsiz olan kod katarları oluşturulacaktır.

Eşsiz bir etiket oluşturmak için [0, 1) aralığı alt aralıklara bölünecektir. Bunu sağlamak için kümülatif dağılım fonksiyonu olan F_x kullanılmaktadır. Çünkü kümülatif dağılım fonksiyonunun (cdf) minimum değeri 0, maksimum değeri 1'dir. Alfabe uzunluğunun m olduğu durumda, başlangıçta m adet eşsiz aralık oluşturmak gerekmektedir. Yani [0, 1) aralığı m adet alt aralığa bölünecektir.

$$[F_x(i - 1), F_x(i)), \quad i = 1, 2, 3,\ldots,m \qquad \textbf{(3.1)}$$

şeklinde birbirini takip eden alt aralıklar oluşturulacaktır. Bu aralıklar da sembolün frekansı oranında belirlenmektir. Bu durumda S_i aşağıdaki gibi ifade edilmektedir.

$[F_x(i - 1), F_x(i)) \rightarrow S_i$

$S_k \rightarrow [F_x(k - 1), F_x(k))$ şeklinde olur.

S_j 'ye karşılık gelen j. aralık ise,

$$[F_x(k-1) + F_x(j-1)/(F_x(k) - F_x(k-1), F_x(k-1) + F_x(j) / (F_x(k) - F_x(k-1))) \qquad \textbf{(3.2)}$$

formülü ile hesaplanır.

Bir örnek ile bu işlemi daha detaylı açıklayabiliriz.

Sıkıştırılacak veri katarımız $S = \{ S_1, S_2, S_3 \}$ olsun. Bunların frekanslarından elde edilen olasılık değerleri de;

$P_1 = 0.7 \qquad P_2 = 0.1 \quad P_3 = 0.2 \quad , \; P_i = P(S_i)$

şeklinde belirlenmiş olsun.

X rastgele değişken olmak üzere, $X(S_i) = i \qquad s_i \in S$

F(1) = 0.7 F(2) = 0.8 F(3) = 1

Mesela ilk gelen karakter S_1 olsun. Bu durumda eşsiz aralığımız [0, 0.7) şeklinde olacaktı. İlk gelen karakter S_2 olsaydı [0.7, 0.8), ilk gelen karakter S_3 olsaydı [0.8, 1) olacaktı. Örneğimizde ilk karakterimiz S_1 olduğu için aralığımızı [0, 0.7) şeklinde belirleyerek devam ediyoruz.

İlk karakterden sonra aralığımız [0, 0.7) olarak belirlenir.

İkinci karakterden sonra S_2 için aralığımız [0, 0.7) aralığının aynı oranlar ile yeniden güncellenmesi ile 0.7x0.7 = 0.49 , 0.7x0.8 = 0.56 şeklinde [0.49, 0.56) olur.

Üçüncü karakterden sonra S3 için aralığımız 0.49x0.8 = 0.546 , 1x0.56 = 0.560 şeklinde [0.546, 0.560) olur.

Son durumda aralığımız [0.546, 0.560) değerine ulaşmış olur. Bundan sonra S_1, S_2 ve S_3 ün aralıkları bulunur ve bir sonraki gelen sembol hangisi ise ilgili aralıkta bir değer seçilerek etiketleri belirlenmiş olur.

S_1 için, 0.560x0.7 = 0.5558 sonucunda [0.5460, 0.5558)

S_2 için, 0.560x0.8 = 0.5572 sonucunda [0.5558, 0.5572)

S_3 için, 0.560x0.1 = 0.5600 sonucunda [0.5572, 0.5600)

değerleri elde edilmiş olur.

Şekil 3.14 ‘te de görüldüğü üzere tüm aralıklar birbirinden ayrıktırlar. Bu da eşsiz bir etiket değeri almalarını sağlıyor. Her adımda; bir sonraki adım için kullanılacak aralık, o adımda gelen sembolün aralığı üzerinden belirleniyor.

Elde edilen aralıklardan sonra her transfer edilecek verinin eşsiz etikti bulunur. Bunun için orta nokta yaklaşımı kullanılabilir.

Yukarıdaki örneğimizde sırasıyla $S_1S_2S_3$ girdilerinin olduğu durumda

$F_x\tilde{}(i) = (F_x(i-1) + F_x(i)) /2$ ile hesaplandığında; **(3.3)**

$F_x\tilde{}(S_1S_2S_3) = (0.546 + 0.560) / 2 = 0.5509$

İkinci adım olarak bulunan bu değer bit katarına çevrilir. Bunun için ondalıklı sayı, ikili tabana çevrilir.

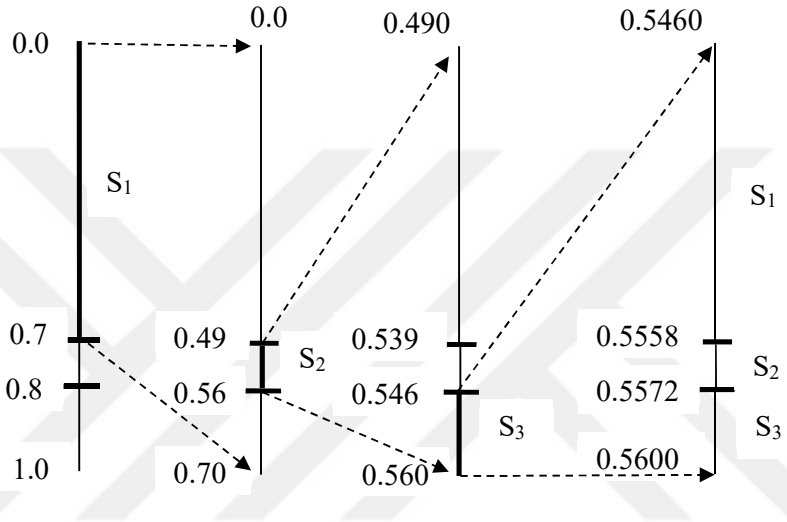
$0.5509 = (0.10001101)_2$ olduğundan, $S_1S_2S_3$ girişinin kod katarı 10001101 olur.

Ondalıklı sayılarda elde edilen bit katarı çok uzun olabilmektedir. Bu değer kesilerek belli bir uzunluk kadarı alınabilir (Fano, 1944).

Uzunluk değeri

$$L_i = -\log_2(p_i) \quad \textbf{(3.4)}$$

formülü ile hesaplanır.



Şekil 3.14 : $S_1S_2S_3$ sıralamasında gelen verinin etiket değer aralıklarının sınırlandırılması.

Dinamik Aritmetik Kodlama algoritması, Dinamik Huffman Kodlama algoritmasında olduğu gibi Kodlama (Encoding) / Çözümleme (Decoding) ve Güncelleme (Update) adımlarından oluşmaktadır. Bit katarı oluşturma/çözümleme yöntemleri dışında ana akışları değişmemektedir.

4. VERİ SIKIŞTIRMADA SIRA KORUMALI DİZGİ EŞLEŞTİRME YAKLAŞIMI

Dinamik kayıpsız veri sıkıştırma yöntemlerinde istatistik sürekli değiştiği için her gelen veri ile birlikte hafızada tutulan istatistik güncellenir, sembollerin kodlama ağacı veya kümülatif değerleri güncel veriler ile yeniden belirlenir. Bu akış bir veri modeli ile temsil edilmiştir. Yani; işleme başlarken modelin alfabe uzunluğu belirlenir, tüm istatistikler başlangıç konumuna getirilir. Veri geldiğinde ise model üzerinde gelen sembolün frekansı arttırılır, modelden kodlama ağacı veya kümülatif değerleri oluşturması istenir. Sembol model tarafından kodlanır ve kod katarına eklenir. Özetle, dinamik sıkıştırma sırasında veri yapısını tutan, akışı yöneten yapımız 'veri modelimiz' olmaktadır. Standard olarak işlem sırasında tek bir veri modeli olur. Eğer gelen veriye bağlı olarak bir desen belirlenip, birden fazla model kullanılırsa, gelen her veri her zaman aynı veri modelini kullanmak yerine farklı veri modelleri kullanacak, böylece her modelin kendi kod katarları olacak. Bu sayede kod katarlarının boyu kısalmakta ve veri sıkıştırmada daha iyi sonuçlar elde edilmektedir. Ancak veri modeli sayısının arttırılması bellek kullanım gereksinimini ciddi oranda arttırmaktadır. Bu nedenle belli bir sayının üzerine çıkmak zorlaşmaktadır, bu da yaygın kullanıma engel olmaktadır. Bu noktada sıra korumalı dizgi eşleştirme yaklaşımımız ile bellek kullanımı daha düşük seviyelerde tutarak aynı sonuçları elde etmeye çalıştık.

4.1 Model

Veri sıkıştırma çalışmasındaki en önemli gelişmelerden biri, Rissanen ve Langdon [RL81] tarafından sunulan modern paradigmadır. Bu paradigma, sıkıştırma işlemini iki ayrı bileşene ayırır: modelleme ve kodlama. Model, sıkıştırılmakta olan verileri üreten kaynağın temsilidir. Modelleme, bu gösterimi oluşturma sürecidir. Kodlama, modelleyicinin kaynağın gösterimini sıkıştırılmış bir gösterimle eşleştirmesini gerektirir. Kodlama bileşeni, modelleyici tarafından sağlanan bilgiyi alır ve bu bilgiyi bir bit dizisine çevirir. Sıkıştırmanın bu ikili yapısını tanımak, dikkatimizi iki süreçten yalnızca birine odaklamamızı sağlar.

Bir istatistik modelleyici tarafından sağlanan kodlama, giriş sembollerinin sıkıştırma gereksinimine iyi bir çözüm olmuştur. Aritmetik kodlama, istatistikleri üretmek için kullanılan modele göre en uygun sıkıştırma sağlar. Yani, kodlayıcıya bilgi sağlayan bir model verildiğinde, aritmetik kodlama minimal uzunlukta bir sıkıştırılmış gösterim sağlar. Witten' nin aritmetik kodlamasının bir tanımını ve uygulaması detaylı olarak incelenebilir. Aynı şekilde Huffman'ın iyi bilinen bir algoritması, başka bir istatistiksel kodlayıcıdır. Huffman kodlaması iki önemli açıdan aritmetik kodlamanın gerisindedir. İlk olarak, Huffman kodlaması, bütünleşik bir bit sayısı kullanarak her olayı (örneğin karakter) temsil etmekle sınırlıdır. Bilgi teorisi bize 4/5 olasılıkla bir olay olduğunu söylüyor (Jinil ve diğ, 2014).

lg 5/4 bit bilgi uzunluğu içerir ve lg 5/4 = 32 bit olarak kodlanmalıdır, Huffman kodlaması bu olayı temsil etmek için 1 bit atayacaktır. Aritmetik kodlamanın doğruluğu, yalnızca üzerinde uygulandığı makinenin kesinliği ile sınırlıdır. Aritmetik kodlamanın ikinci avantajı, değişen modelleri daha etkili bir şekilde gösterebilmesidir. Bir Huffman ağacını güncellemek çok daha fazla zaman alıcı. Araştırmacılar verimlilik amacıyla aritmetik kodlama yapmaya devam ediyorlar.

En uygun kodlama yönteminin varlığı göz önüne alındığında, modelleme etkin veri sıkıştırmanın anahtarı haline gelir. Bir modelleme paradigmasının seçimi ve uygulanması sistemin kaynak gereksinimlerini ve sıkıştırma performansını belirler. Bağlam modelleme, metin sıkıştırma için istatistiksel modellemeye çok umut verici bir yaklaşım olmuştur. Bağlam modelleme, özel durumlu bir Markov modellemesidir ve aslında Markov modelleme terimi, sıklıksal modellemeye atıfta bulunmak için sık sık kullanılır. Bu bölümde bağlam modellemesi stratejisini ve bağlam modellerinin uygulanmasında yer alan parametreleri açıklıyoruz.

4.2 Bağlam Modellemesi

Sonlu uzunluklu bağlam modeli, mevcut karakterin kodlamasını belirlemek için daha önce görülen karakterlerin sağladığı bağlamı kullanır. Birkaç önceki karakterden oluşan bir bağlam fikri, sıkıştırılan veriler doğal bir dil olduğunda çok mantıklıdır. Hepimiz biliyoruz ki, düz bir metinde q harfini izleyen karakterin hepsi garanti edilmekle kalmaz, bu bağlamda kelimenin veya cümlenin devamında gelebilecek harfler öngörülebilmektedir. Bu türden bir bilgiyi kullanmanın, bilgi kaynağının daha doğru bir şekilde modellenmesine yol açacağı beklenmektedir.

Bağlam modelleme tekniği iyi bir düzeyde gelişmiş ve özellikle de doğal dili sıkıştırmak için açıkça uygun olsa da, bağlam modelleri doğal dil dışında farklı çeşitlilikte bir dizi alanda çok iyi sıkıştırma sağlar.

Bir bağlam modelinin, daha önce görülen karakterlerin sağladığı bağlamı dikkate alarak başarılı karakterleri öngördüğünü söylüyoruz. Burada öngörülen ile kastedilen, mevcut karakteri kodlarken kullanılan frekans değerlerinin hangi bağlamı kullanacağının belirlendiğidir. Bir karakteri kodlamak için kullanılan frekans dağılımı, sıkıştırılmış gösterime katkıda bulunur ve bit sayısını belirler. x karakterine c bağlamında bakıldığında c, x için bir frekans içermez, bağlam c'nin x'i tahmin edemediğini söyleriz.

Bir bağlam modeli tahminlerinde sabit sayıda önceki karakter kullanabilir veya birkaç uzunluktaki bağlamlara dayanan tahminleri içeren karma bir model olabilir. Geçerli karakteri tahmin etmek için her zaman önceki karakterleri kullanan bir model saf bir order-i bağlam modelidir. i = 0 olduğunda, bağlam kullanılmaz ve metin bir seferde bir karakterle kodlanır. i = 1 olduğunda, önceki karakter mevcut karakteri kodlarken kullanılır; i = 2 olduğunda, önceki iki karakter kullanılır vb.. Bir harmanlanmış model, önceki üç karakteri, üç karakter bağlamı tahmin edemediğinde önceki iki karakteri ve hem order-3 hem de order-2 bağlamları başarısız olduğunda bir öncekini kullanabilir. Karışımlı bir model iki veya daha fazla alt modelden oluşur. Bir order-i bağlam modeli, giriş akışında meydana gelen her bir i karakter dizisi için bir frekans dağılımından oluşur. Order-1 durumunda, bu, q bağlamındaki frekans dağılımının u ve diğer herhangi bir harfe az ağırlık, bağlam t dağılımı ise diğerleri arasında a, e, i, o, u ve h için yüksek frekanslara ve q, n ve g gibi harfler için çok az ağırlığa sahip olacaktır.

Karışımlı bir model, maksimum uzunluk bağlamı ve tüm düşük dereceli bağlamlar için alt modeller içeriyorsa, tamamen harmanlanmıştır. Yani, tamamen harmanlanmış bir order-3 bağlamı modeli, tahminlerini 3, 2, 1, 0 order modellerine dayandırır (order 1 modeli, tüm karakterleri eşit ağırlıkta bir frekans dağılımından oluşur). Kısmen harmanlanmış bir model, daha düşük dereceli içeriğin bir kısmını kullanır ancak tümünü kullanmaz.

Bağlam modellemesi de statik veya dinamik olabilir. Kodlama işlemi boyunca bilgileri değişmeden kalırsa, bir model statiktir. Dinamik veya uyarlamalı bir model, kodlama

ilerledikçe girdi olan değerin gösterimini değiştirir ve kaynağın özellikleri hakkında bilgi toplar. Çoğu statik veri sıkıştırma modelinde adaptif eşdeğerler bulunur ve adaptif eşdeğerler genellikle daha etkili sıkıştırma sağlar. Aslında, Bell çok çeşitli şartlar altında, herhangi bir statik modelden sadece biraz daha kötü olacak bir adaptif model olduğunu kanıtlarken, statik bir modelin adaptif bir modelden sıkıştırma olarak daha kötü olabildiği durumlar olmuştur. Tartışmamızı uyarlamalı bağlam modeli ile sınırlayacağız.

Bir bağlam modeli genellikle bir veri sıkıştırma sistemi oluşturmak için aritmetik kodlamayla birleştirilir. Model, her bağlam için bir frekans dağılımı sağlar (order-1 deki her karakter ve order-2 deki her karakter çifti). Her frekans dağılımı, bir aritmetik kodun temelini oluşturur ve bunlar olayları kod bitlerine eşlemek için kullanılır. Huffman kodlaması yukarıda belirtilen nedenlerden dolayı uyarlamalı bağlam modelleriyle kullanım için uygun değildir.

4.3 Modelleri Harmanlama Yöntemleri

Harmanlama arzu edilen ve esasen kodlama ilerledikçe modelin sıfırdan yapıldığı adaptif bir ortamda kaçınılmaz olan bir durumdur. Bir dosyanın ilk karakteri okunduğunda, modelin tahminlemeleri temel alacak bir geçmişi yoktur. Sıkıştırma ilerledikçe daha büyük bağlamlar daha anlamlı hale gelir. Genel harmanlama, ağırlıklı harmanlama mekanizması, çeşitli alt modeller tarafından sağlanan olasılıkları (veya daha doğru bir şekilde frekansları) ağırlıklandırarak ve bu olasılıkların ağırlıklı toplamını hesaplayarak her bir karaktere bir olasılık verir. Bu harmanlama yöntemi pratik olamayacak kadar yavaştır ve ek olarak çeşitli dezavantajlara sahiptir, ayrıca çeşitli order-i modellerine ağırlık atamak için teorik bir temeli yoktur. Daha basit ve daha pratik harmanlanmış bir order-i modelinde, c karakterini kodlamak için kullanılan bit sayısı, eğer daha önce bu özel bağlamda c olmuşsa, önceki i karakterleri tarafından belirlenir. Bu durumda, yalnızca order-i frekans dağılımı kullanılır. Aksi takdirde, birinden bir tahmin belirleyene kadar düşük order (sıra) modellerine gidilir. Order-i bağlamı, mevcut karakteri tahmin edemediğinde, kodlayıcı, bir kod çözücüye, düşük dereceli modele başvurulduğuna dair bir işaret olan bir değer gönderir. Girdi alfabesindeki her karakter için bir tahmin sağlamak için bazı düşük seviye kodlayıcı modellerin garanti edilmesi gerekir.

Çözümleyicinin çıktı olacak kodu iletmesi için her frekansın harmanlanmış modeldeki dağılımına ve bu dağılımda çıktı için tahsis edilmiş bir frekansa sahip olmalıdır. Basit bir strateji, escape olayını girdi alfabesinde ek bir sembol gibi ele almaktır. Başka herhangi bir karakter gibi, escape olayının frekansı, meydana gelme sayısıdır. Diğer stratejiler, çıktı kodunun sıklığının bağlamın toplam sıklığına ve bağlamda meydana gelen farklı karakterlerin sayısına ilişkilendirilmesini içerir. Bir yandan, farklı karakterlerin sayısı arttıkça, bağımlılık olasılığı artar ve çıkış kodunun kullanılması daha az olası hale gelir. Öte yandan, bir bağlam sık sık ortaya çıktıysa ve her seferinde aynı karakteri (veya az sayıda karakteri) öngördüyse, yeni bir karakterin ortaya çıkması pek mümkün görünmüyordur. Bu kaçış stratejilerinden birini diğerinden seçmek için teorik bir temel yoktur. Neyse ki, deneysel deneyler sıkıştırma performansının kaçış stratejisinin seçimine büyük ölçüde duyarsız olduğunu göstermektedir.

Yukarıda açıklanan harmanlama stratejisi, hariç tutma etkisine sahiptir. Bir üst düzey modelde bir karakter meydana geldiğinde ilgili modeldeki frekans değerleri güncellenir. Ancak, olabildiğince düşük dereceli bilgileri dışlamaz. Örneğin, x karakteri bağlamı abc'de ilk kez gerçekleştiğinde, order-2 bağlamı bc'ye danışılır. Eğer abc bağlamında y karakteri olmuşsa, order-2 tahmininden çıkarılabilir. Yani, order-3 bağlamından abc çıkmamız gerçeği, kod çözücüye kodlanan karakterin y olmadığını bildirir. Bu nedenle, bc modelinin bu öngörüyü yaparken y'ye herhangi bir frekans ataması gerekmez. y'yi order-2 tahmininden çıkartarak, x daha doğru tahmin edilebilir. Yüksek dereceli modellerde öngörülen karakterlerin dışlanması yürütme süresini iki katına çıkarabilir. Sıkıştırma performansındaki artış,% 5 düzeyindedir; bu, artan yürütme zamanını neredeyse hiç haklı çıkarmaz. Başka bir tür olan ve çok daha basit olan ve yürütme süresini azaltma etkisine sahip dışlama, güncelleme dışlamasıdır. Güncelleme dışlaması, yalnızca geçerli tahmine katkıda bulunan modelleri güncellemek anlamına gelir. Dolayısıyla, yukarıdaki örnekte, bağlam bc x'i öngörüyorsa, sadece abc için order-3 modeli ve bc için order-2 modeli güncellenecektir. Düşük dereceli modeller değişmeden kalır.

4.4 Kaynak (Memory) Sınırlamaları

c bağlamında meydana gelen her x karakteri için, model her x için c bağlamında bir frekans dağılımı içeriyorsa, buna bir tamamlanmış (complete) order-i bağlam modeli

diyoruz. Yani model öğrendiği her şeyi elinde tutar. Order-3 için bile tam bağlam tamamlanmış modeller nadirdir, çünkü büyük bir dosyadan toplanan tüm bağlam bilgisini saklamak için gereken alan çok fazladır. Sınırlı bir bağlam modeline bir hafıza limiti koymanın iki açık yolu vardır. Birincisi, boyutunu izlemek ve boyut maksimum seviyeye ulaştığında modeli dondurmaktır. Model donduğunda, yeni bağlamlarda ortaya çıkan karakterleri artık temsil edemez, ancak modelde zaten kayıtlı olan frekans değerlerini güncellemeye devam edebiliriz. İkinci yaklaşım, modeli dondurmak yerine yeniden inşa etmektir. Model sıfırdan veya yakın geçmişini temsil eden bir tampondan yeniden oluşturulabilir. Tampon kullanımı, yeniden yapılanma nedeniyle sıkıştırma performansındaki bozulmayı azaltabilir. Öte yandan, arabellek için ayrılan bellek daha önce yeniden oluşturulmasına neden olur. Sınırlı bellek sorununa kesin olarak bir çözüm olmayan üçüncü bir yaklaşım, sıkıştırma yapısının yanı sıra veri yapısının boyutunun izlenmesidir. Sıkıştırma bozulmaya başladığında yeniden yapılanma, gerekli olana kadar beklemekten daha elverişli olabilir. Modelin temsili, içerebileceği bilgi miktarını ve kullanım kolaylığını açıkça etkiler. Bu temsile erişilebilir ve güncellenebilir.

4.5 Sıkıştırma Performansının Ölçülmesi

Sıkıştırma performansı genellikle iki formdan birinde sunulur: Sıkıştırma oranı ve sıkıştırılmış gösterimde karakter başına bit sayısı. Sıkıştırma oranı, oran (sıkıştırılmış gösterimin boyutu) / (orijinal girişin boyutu) 'dur ve sıkıştırma sonrasında kalan orijinal girişin yüzdesi olarak ifade edilebilir. Bu tanımların her biri, giriş akımı, model ya da algoritmanın kodlama bileşeni hakkında herhangi bir varsayımda bulunmadığı (örneğin, ergodiklik) olduğu varsayımı ile yapılmaktadır. Sıkıştırma, karakter başına bit sayısı cinsinden rapor edildiğinde, aynı zamanda orijinal girdideki sembollerin temsilini de etkiler (Ziv ve Lempel, 1978).

Bir veri sıkıştırma tekniğinin performansı net olarak uygulandığı verinin türüne ve ayrıca belirli bir veri dizisinin özelliklerine bağlıdır. Bir veri sıkıştırma sisteminin etkinliğini ölçmek için, çeşitli tür ve boyutları temsil eden geniş bir dosya grubuna uygulanmalıdır. Tekniklerin güvenilir bir karşılaştırması ancak aynı verilere uygulandıkları takdirde yapılabilir.

4.6 Uygulanan Bağlam Modellemesi Yaklaşımı ve Etkisi

Kodlama/Çözümleme sırasında gelen veriye göre model oluşturma, gelen sembolden önceki k tane sembole bakılarak k-order yaklaşımı ile yapılmaktadır. Bu sayede birden fazla sayıda model olabilmektedir. Hangi modelin hangi durumda kullanılacağı konusunda bir yapı oluşturulur. Modelleri tutan ve kurgulayan bu yapıya bağlam denilmektedir. Bağlam yapısında her modelin bir indeksi olur. Bu indeks, tek modelli yapıda 0 olmaktadır.

Örnek olarak; alfabe uzunluğu m = 3 olsun, S = { a, b, c}

Kendisinden önceki herhangi bir veriye bakmadığımız durumda k-order yaklaşımına göre değerimiz 0-order olur. Kodlanmak için gelen sembol 'a' olması durumunda bunun frekans değeri 1 yapılır. Sonra model üzerinde güncelleme yapılıp yeni kod ağacı veya kümülatif sembol değerler elde edilir.

Çizelge 4.1 : Bağlam modeli : 0-order.

Bağlam İndeksi	a	b	c
0	0 → 1	0	0

0-order durumunda model sayısı $m^k = 3^0 = 1$ dir.

Çok modelli yapıda ise, model sayısı k-order yaklaşımı ile belirlenmektedir. Yani kendisinden önce gelen k tane karakter alınarak bir bağlam indeksi oluşturulur ve indeksteki model üzerinde güncelleme yapılır.

Örnek olarak 1-order için;

İlk sembol kodlanmaz, sıkıştırmadan iletilir. İlk sembolün 'a' geldiğini varsayalım.

'a' sembolünden sonra 'b' gelmesi durumunda, aşağıdaki gibi 0 indeksli model bulunup güncellenir.

Çizelge 4.2'de görüldüğü gibi k = 1 olduğu durumda t_0 anında gelen sembol 'a', t_1 anında gelen sembol 'b' olması durumunda t_0 'a, yani t_{k-1}'e, ait model bulunup kullanılır.

Bu şekilde k-order yaklaşımında k değeri arttıkça model sayısı artmakta, kod katarları küçülebilmekte ancak özellikle alfabe uzunluğu arttıkça bellek kullanımı giderek artmaktadır.

Çizelge 4.2 : Bağlam modeli: 1-order.

Bağlam İndeksi	a	b	c
0 (a)	0	0 → 1	0
1 (b)	0	0	0
2 (c)	0	0	0

1-order durumunda model sayısı $m^k = 3^1 = 3$ tür.

Örneğin, alfabe uzunluğunun (kullanılabilen karakter sayısının) m = 256 olduğu durumda, 1-order için model sayısı 256^1, 2-order için 256^2, 3-order için 256^3, … şeklinde devam etmekte. 2-order dan sonrası ciddi anlamda bellek kullanımı gereksinimi oluşturmaktadır.

Çizelge 4.3 : Bağlam modeli: m = 256, 3-order.

Bağlam	a	b	c	d	…
aaa	0	0	0	0	0
aab	0	0	0	0	0
aac	0	0	0	0	0
…					

Çizelge 4.3'te görüldüğü gibi alfabe uzunluğunun arttığı durumda model boyutu ciddi anlamda yüksek değerlere ulaşmaktadır.

Bu nedenle k-order = 2-order 'dan daha yüksek k değerleri kullanılmamaktadır.

4.7 Sıra Korumalı Dizgi Eşleştirme Tabanlı Bağlam Modellemesi

Bağlam modellemesinde k-order yaklaşımında k değerinin artması durumunda sıkıştırma oranında daha iyi sonuçlar alınmaktadır. Ancak yukarıda anlatıldığı gibi kaynak gereksinimi yüksek olduğundan pek kullanılamamaktadır. Burada Sıra Korumalı Dizgi Eşleştirme yöntemi kullanarak kaynak kullanımını azaltıp, yüksek k değerlerini kullanabilme imkanı bulmaktayız.

Öngördüğümüz bu yaklaşım ile gelen k tane sembolün değerlerini tutmak yerine, aralarındaki sıralama ilişkisini gösteren bir numara tutulabilir.

Mesela, 'karanlıkta' ifadesinde k = 3-order alındığını ve 't' sembolünün geldiğini düşünelim.

k = 3 olduğu için, bağlam için kullanılacak veri 'lık' kısmıdır. 'lık' ifadesinde geliş sırasına göre 'l' = 1, 'ı' = 2, 'k' = 3 indekslerine sahip olduğundan, bunların sıralaması 'ı' < 'k' < 'l' şeklinde olacaktır, yani 2 < 3 < 1 sıralamasına sahiptir.

k = 3 için, olabilecek değerler Çizelge 4.4'te gösterilmiştir. Bunları eşsiz, sıralı olarak numaralandıralım;

Çizelge 4.4 : k = 3 için permütasyon değerleri ve numaralandırması.

Sıralama	Numerik Değer
1 < 2 < 3	1
1 < 3 < 2	2
2 < 1 < 3	3
2 < 3 < 1	4
3 < 1 < 2	5
3 < 2 < 1	6

Her bir sıralama değerine karşılık gelen bir numara bulunmakta. Gelen sembol ne olursa olsun, k = 3 olan bütün verilerde bu 6 değerden birisi elde edilir.

Dolayısı ile bu yöntem ile 3-order için oluşabilecek model sayısı k! = 3! = 6 ‘dır. Standard yaklaşım ile $m^k = 256^3$ olan bir model sayısı, sıra korumalı yaklaşım ile

k!=3! = 6 değerine düşmektedir.

Buna karşılık ‘abc’ değeri ile ‘cde’ değeri aynı sıralamaya sahip olduğundan, ‘abct’ ile ‘cdet’ dizgileri geçiyorsa aynı bağlam numarasına sahip olacakları için aynı modeli kullanacaklar. Bu yüzden bazı durumlarda daha kötü sıkıştırma sonuçlarına rastlanabilir. Ancak k-order değeri arttıkça bu ihtimal azalacaktır.

4.7.1 Sıra korumalı dizgi eşleştirme yöntemine ek indeks eklenmesi

İçeriği farklı olan, sıralaması aynı olan dizgilerde aynı model kullanılmaktadır. Bu yüzden sıra korumalı dizgi eşleştirmesinde her zaman çok iyi sonuç elde edilemeyebilir. Keza bazı dosya desenlerinde daha iyi sonuç elde edilmediği görüldü. Bu duruma alternatif olabilecek bir çözüm olarak, sıralamaya karşılık gelen permütasyon numarasının yanına bir de k-order’a karşılık gelen dizgideki herhangi bir karakter eklenebilir.

Örneğin, ‘acb’ dizgisinde sonraki karakter ‘c’ olsun. ‘abc’ dizgisinde 1 < 3 < 2 sıralaması vardır. Çizelge 4.4’e göre bunun permütasyondan gelen desen numarası 2 olur. Ek indeks değerimiz 3 olsun. Bu durumda bağlam tablomuz Çizelge 4.5’te gösterildiği gibi olur.

Çizelge 4.5’te görüldüğü gibi alfabe uzunluğu = m olduğu bir işlemde (k! x m) tane veri modeli kullanmış olur. Standard yaklaşımdan daha az kaynak kullanan bu yaklaşımın etkili olduğu dosya desenleri bulundu. k-order’daki hangi elemanın daha iyi sonuç verdiğine ilişkin de detaylı bir inceleme ve çıkarım yapıldı. Genellikle düşük sıradaki indeks alındığında daha iyi sonuç alındı. En ideal sonuçlar ek indeks değeri 1 veya 2 olduğu durumda elde edildi. Kullanılan külliyatlardaki tüm dosyalar işlenip sonuçları detaylı bir şekilde son bölümde listelenmiştir.

Çizelge 4.5 : Bağlam modeli: m = 3, 3-order, ek indeks = 2.

Bağlam	a	b	c
1a	0	0	0
2a	0	0	0
3a	0	0	0
4a	0	0	0
5a	0	0	0
6a	0	0	0
1b	0	0	0
2b	0	0	0 → 1
3b	0	0	0
4b	0	0	0
5b	0	0	0
6b	0	0	0
1c	0	0	0
2c	0	0	0
3c	0	0	0
4c	0	0	0
5c	0	0	0
6c	0	0	0

4.8 Yöntemlerin Bağlam Uzunlukları

Standard yönteminin, Sıra Korumalı Dizgi Eşleştirme yönteminin ve Sıra Korumalı Dizgi Eşleştirmede Ek İndeks kullanılması yöntemlerinin kullandığı veri modeli sayısı Çizelge 4.6'da belirtilmiştir. Bu çizelgedeki değerler bize kaynak kullanımındaki kazançları göstermektedir. Alfabe uzunluğu m = 256 olarak alınmıştır.

(OPPM: Sıra Korumalı Dizgi Eşleştirme, Order-preserve pattern matching)

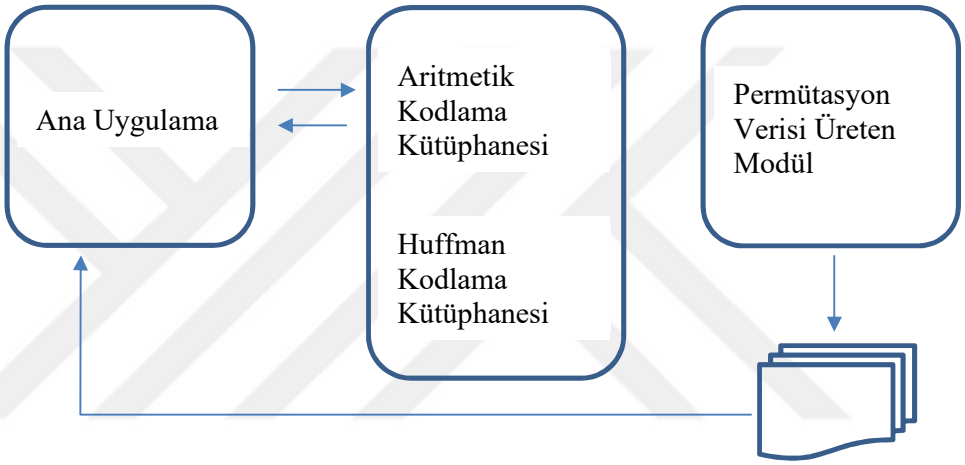
(OPPM + Add.: Sıra Korumalı Dizgi Eşleştirme + Ek indeks)

Çizelge 4.6 : Bağlam modelleme yöntemi-model sayısı listesi.

	k=0	k=1	k=2	k=3	k=4	k=5	k= 6	k=7	k=8
Standart	1	256	256^2	256^3	256^4	256^5	256^6	256^7	256^8
OPPM	1	1	2	6	24	120	720	5040	40320
+Add.	1	-	2x256	6x256	4!x256	5!x25	6!x256	7!x256	8!x256

5. UYGULAMA

Geliştirilen uygulama üç temel bileşenden oluşmaktadır. Permütasyon verisi üreten bir modül, sıkıştırma algoritmalarını gerçekleyen bir kütüphane ve örnek veriyi alıp akışları işleten ana uygulama.



Şekil 5.1 : Uygulama mimarisi.

5.1 Permütasyon Verisi Üreten Modül

Permütasyon verisini işlem anında üretmek çok maliyetli bir işlemdir. Bu nedenle permütasyon verisini üreten ayrı bir modül geliştirildi. Bu modül n = 1'den başlayarak, n = 10'a kadar, permütasyon datalarını üretip numaralandırmaktadır. Her bir n değeri için bir dosya oluşturulmuştur. Ana işlemi yapan uygulama bu dosyaları kullanmaktadır. Dosya içeriğinin bir örneği aşağıdadır.

3=1;2;3;=1 3'lü permütasyon, = ayracı, 1;2;3; durumu, eşsiz numara değeri 1

3=1;3;2;=2

3=2;1;3;=3

3=2;3;1;=4

3=3;1;2;=5

3=3;2;1;=6

5.2 Aritmetik ve Huffman Kodlama Kütüphaneleri

Veri kodlama ve çözümleme işlemleri sırasında gereksinim olan Dinamik Huffman Kodlama ve Dinamik Aritmetik Kodlama işlemleri hazır kütüphaneler kullanılarak yapılmıştır. Bu kütüphaneler anlatılan akışa uygun veri kodlama/çözümleme ve modelin güncellenmesi işlemlerini yapabilmektedirler. Alfabe uzunluğu dışardan verilmektedir.

5.3 Kodlama ve Çözümleme İşlemini Gerçekleştiren Ana Uygulama

Uygulamaya aşağıdaki parametreler iletilir:

- Yöntem: Standard / OPPM / OPPM + Additional Index
- Varsa Additional Index (Ek indeks)
- İşlem: Code / Decode
- Kaynak Dosya Adı
- Hedef Dosya Adı
- k-order değeri

Uygulama aldığı parametrelere göre ilgili yordamı çağırır. Yordamların akışları aşağıda anlatılmıştır.

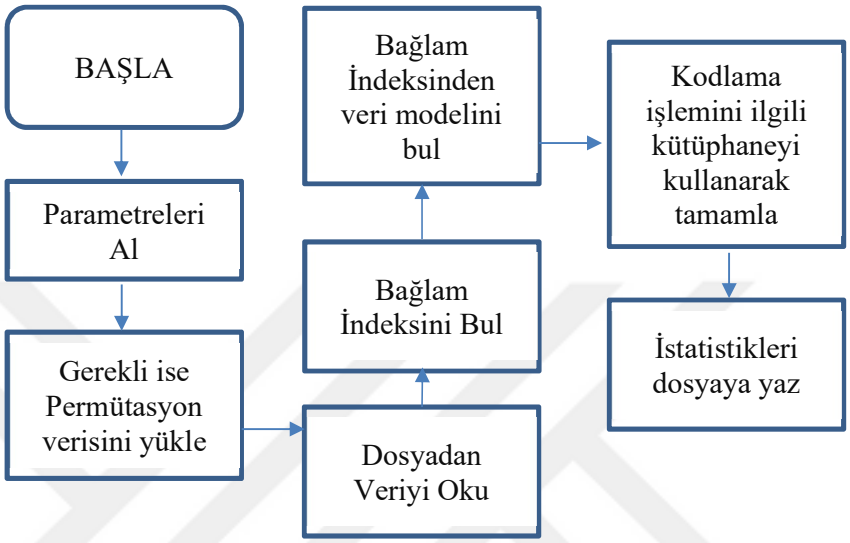
5.3.1 Sıra numarası bulan yordam

- Uygulama ayağa kalktığında k-order değerindeki permütasyon dosyasını ön belleğine yükler.
- Gelen veriyi sıralar
- Ön bellekten permütasyon verisini okuyarak, sıralamaya uygun olan veriyi bulur ve bunun eşsiz sıra numarasını alır.
- Eğer Ek İndeks yöntemi seçilmişse, alfabe uzunluğu = m,

  (Sıra Numarası x m + Ek indeks) değerini döner.
- Ek İndeks yöntemi seçilmemişse Sıra Numarası bilgisi dönülür

5.3.2 Kodlama akışı

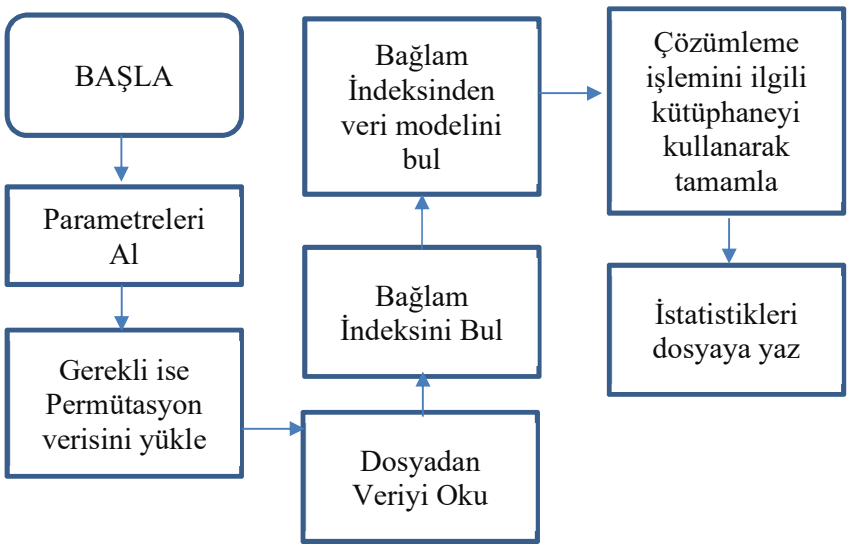
Kodlama akışı giriş parametrelerine göre veriyi kodlar. İstatistiklerini kaydeder. Akış aşağıdaki şekildedir:



Şekil 5.2 : Verinin kodlanması akışı.

5.3.3 Çözümleme akışı

Çözümleme akışı giriş parametrelerine göre veriyi çözümler. İstatistiklerini kaydeder. Akış aşağıdaki şekildedir:



Şekil 5.3 : Verinin çözümlenmesi akışı.

6. SONUÇ VE ÖNERİLER

Geliştirilen yeni yaklaşım modelinin etkisini görmek için dosya külliyatlarından (corpus) birçok dosya uygulama tarafından işlenmiştir. Dosyalar işlenirken her bir dosya için bütün yöntemler ayrı ayrı uygulanmış, aynı zamanda her bir yöntem k = 0'dan 8'e bütün k-order değerleri ile işlenmiş ve sonuçlar gözlemlenmiştir. Ayrıca Sıra Korumalı Dizgi Eşleştirme + Ek indeks yönteminde hangi indeksin optimum olduğunu gözlemlemek için tüm değerler için süreçler çalışmış, detaylı olarak istatistiklere dökülmüştür.

Dosya külliyatlarından 'canterbury' ve 'silesia' külliyatları işlenmiştir. Genel gözlemler aşağıdaki gibidir:

- Ek indeksli işlemelerde (Additional Index) Canterbury külliyatı için Ek İndeks 1 iken hemen hemen en ideal sonuç yakalandı.
- Ek indeksli işlemelerde (Additional Index) Silesia külliyatı için Ek İndeks 2 iken hemen hemen en ideal sonuç yakalandı.
- Excel dosyalarında ciddi iyileşmeler gözlemlendi.
- Text, roman, html ve c dosyalarında hemen hemen aynı değerler yakalandı, daha düşük bağlam sayısı ile.
- Tıbbi ilaç yapı ve tıbbi x-ray dosyalarında daha iyi sonuç yakalandı.
- Tıbbi manyetik resim ve yıldız kümesi yapıları dosyalarında hemen hemen aynı değerler yakalandı, daha düşük bağlam sayısı ile.
- Open Office dll'i ve doğal text dosyalarında daha kötü sonuçlar elde edildi.

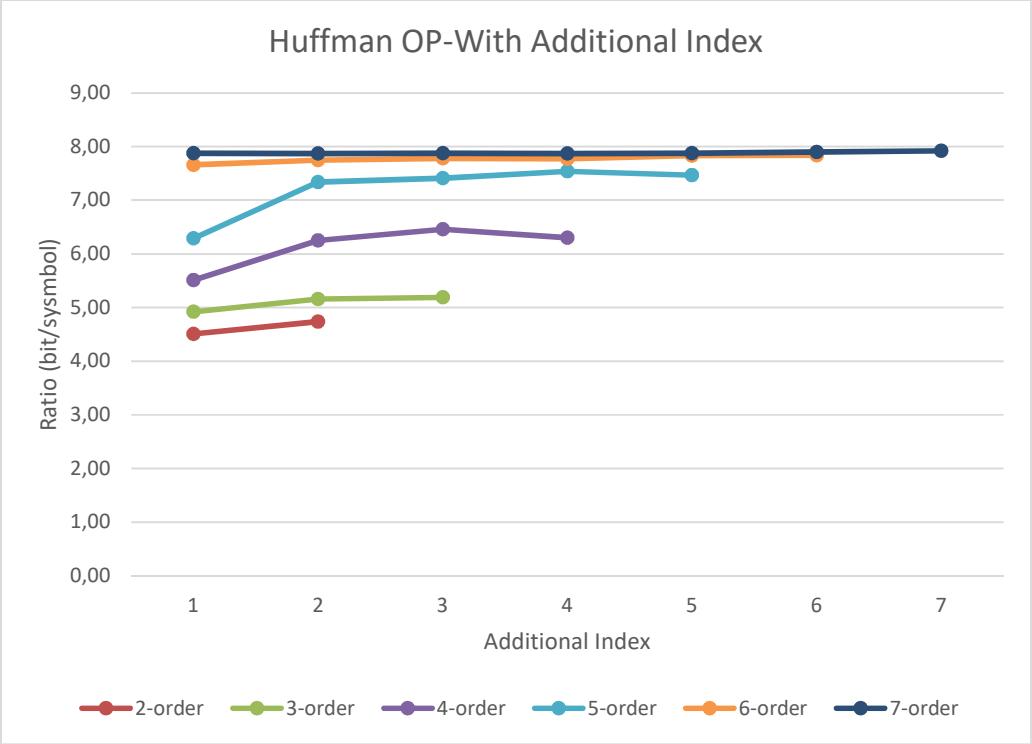
6.1 Sıra Korumalı Dizgi Eşleştirme Yönteminde Ek İndeks Sonuçları

İşlenen dosya : alice29-0.15MB , Canterbury corpus, text roman dosyası Şekil 6.1 ve Şekil 6.2 grafiklerinde görüleceği üzere, Huffman/Arithmetic Order Preserve Pattern Matching With Additional Index yöntemi ile işlenmiş bir dosya var.

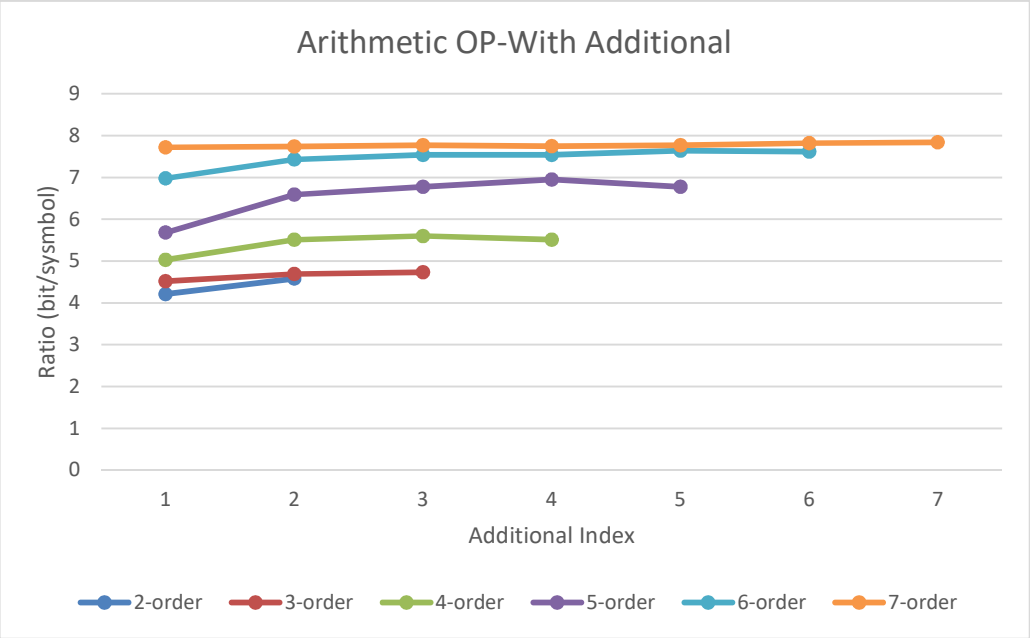
Dosya sırasıyla:

- k = 2 (Addit. Ind. = 1)
- k = 2 (Addit. Ind. = 2)
- k = 3 (Addit. Ind. = 1)
- k = 3 (Addit. Ind. = 2)
- k = 3 (Addit. Ind. =3)
- k = 4 (Addit. Ind. = 1)
- …

şeklinde tüm sıralar için işlenmiş, görüldüğü gibi en iyi sıkıştırma oranı hep Additional Index (Ek İndeks) = 1 iken elde edilmiştir.



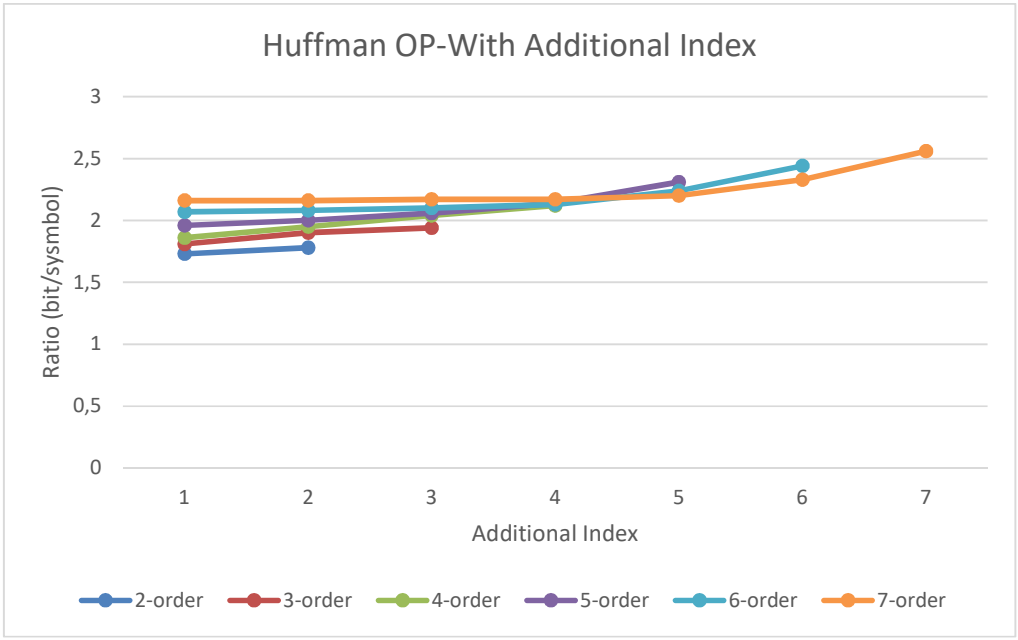
Şekil 6.1 : Dosya : alice29 – Huffman OP with additional.



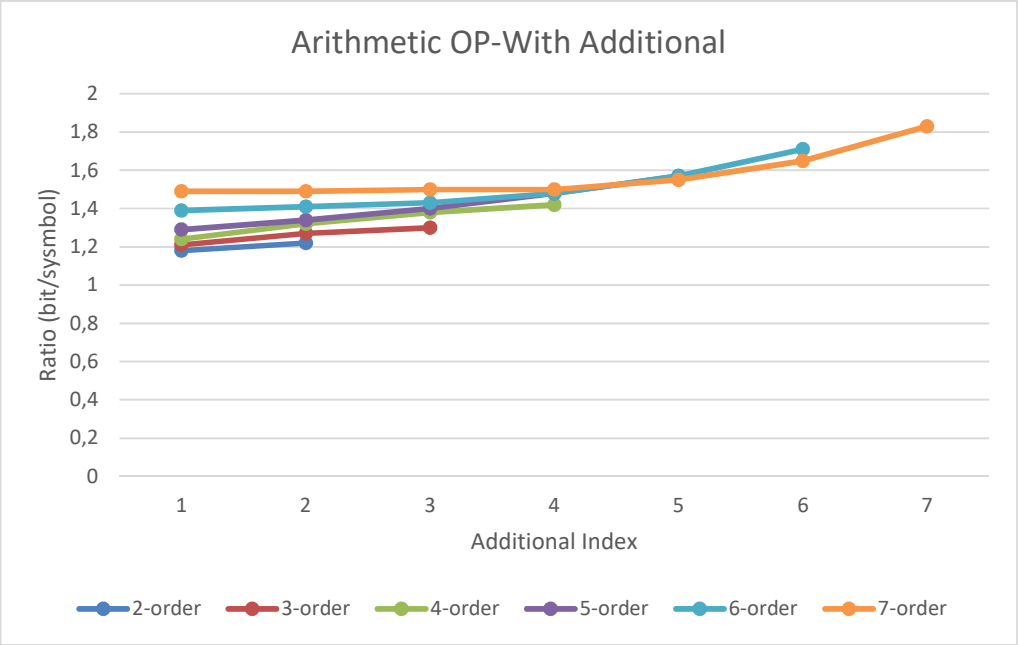
Şekil 6.2 : Dosya : alice29 – Arithmetic OP with additional.

İşlenen dosya: ptt5-0.49MB, Canterbury corpus, CCITT test kümesi

Bu dosya da aynı şekilde, en iyi sonuçlar Additional Index (Ek İndeks) = 1 iken alınmış.



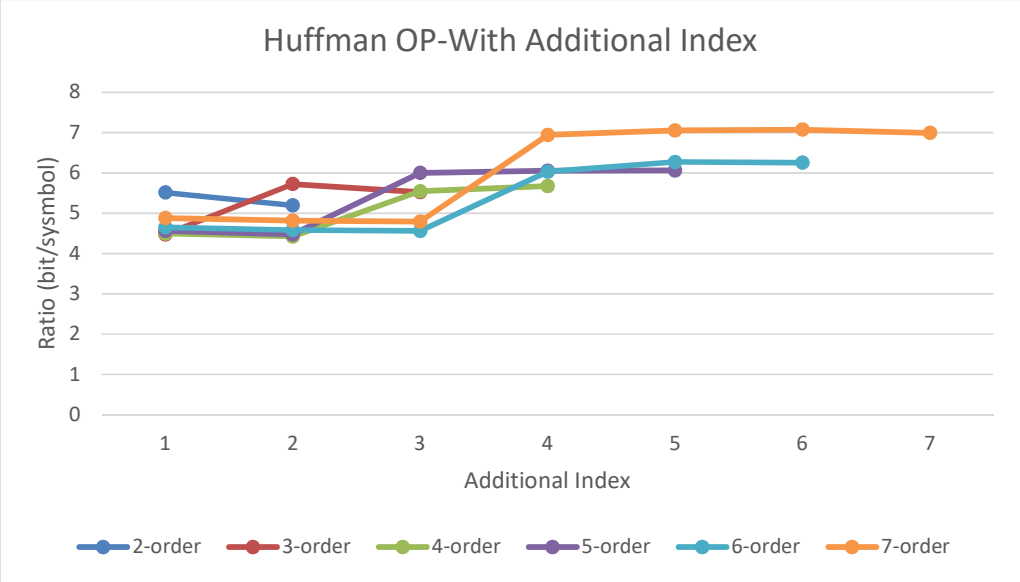
Şekil 6.3 : Dosya: ptt5 – Huffman OP with additional.



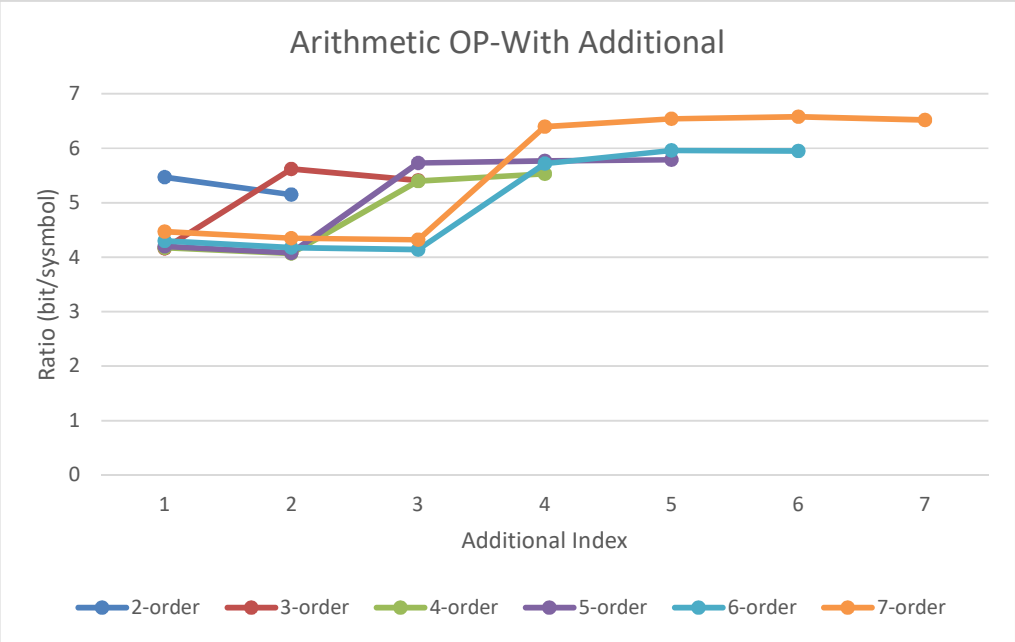
Şekil 6.4 : Dosya: ptt5 – Arithmetic OP with additional.

İşlenen dosya : x-ray-8.08MB, silesia corpus, x-ray tıbbi imaj dosyası

Şekil 6.5 ve 6.6 'dan görüleceği üzere bu dosyada en iyi sonuç ağırlıklı olarak Additional Index (Ek İndeks) = 2 olduğu zaman alınmış.



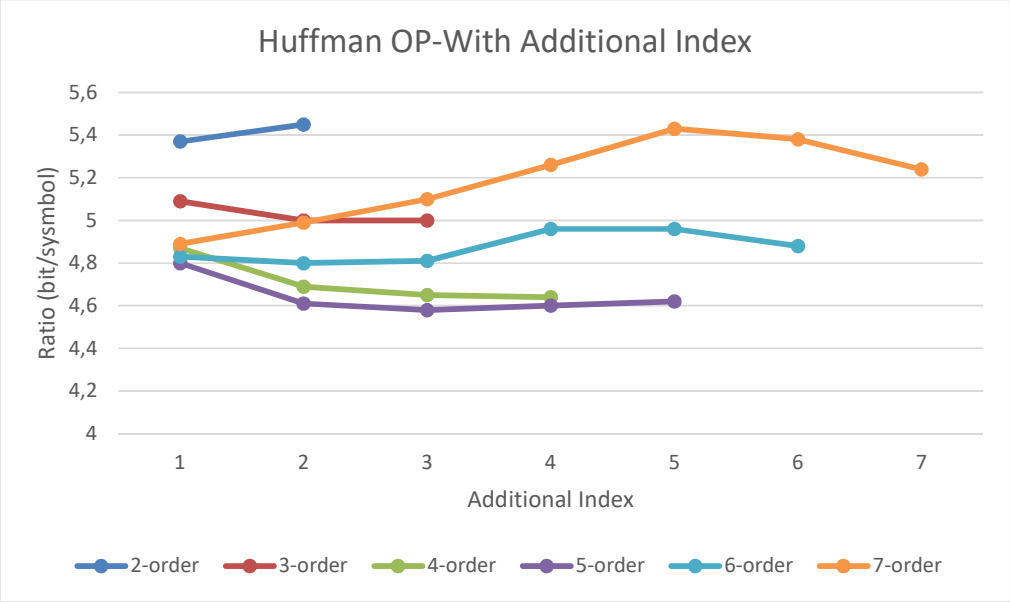
Şekil 6.5 : Dosya: x-ray – Huffman OP with additional.



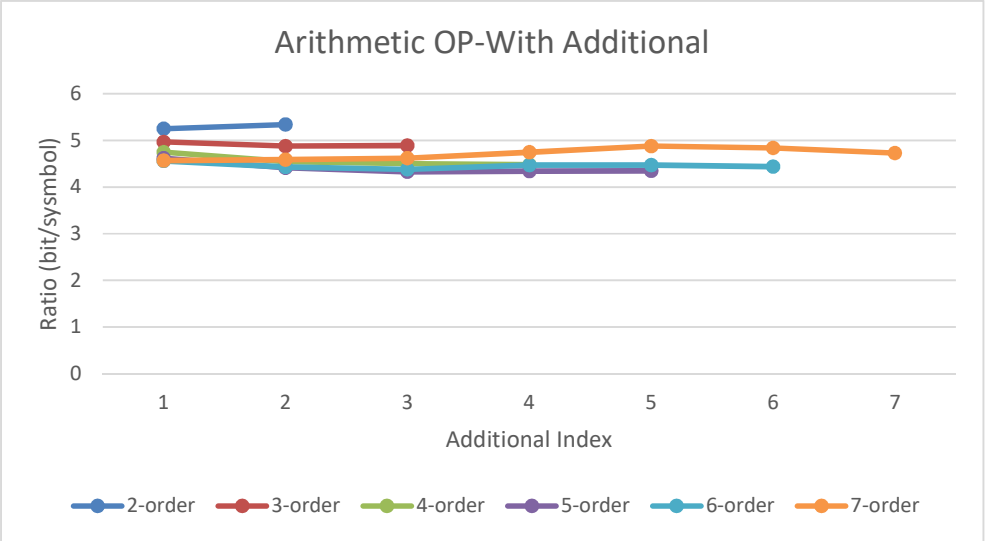
Şekil 6.6 : Dosya : x-ray – Arithmetic OP with additional.

İşlenen dosya: mozilla-48.85MB, Silesia corpus, web sayfası dosyası

Aynı şekilde Silesia corpus' undaki başka bir dosyada da en iyi sonuçlar Additional Index (Ek İndeks) = 2 iken alınmış.



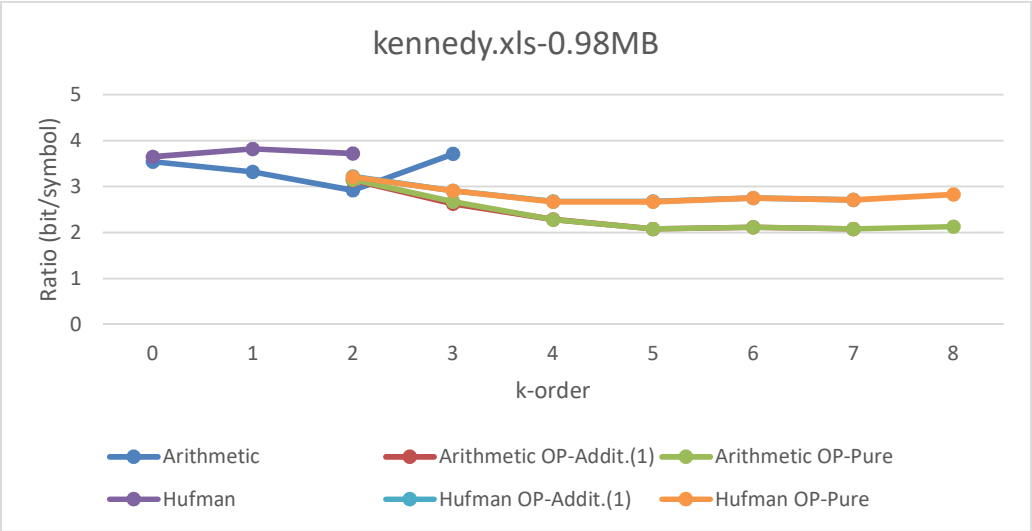
Şekil 6.7 : Dosya : mozilla – Huffman OP with additional.



Şekil 6.8 : Dosya : mozilla – Arithmetic OP with additional.

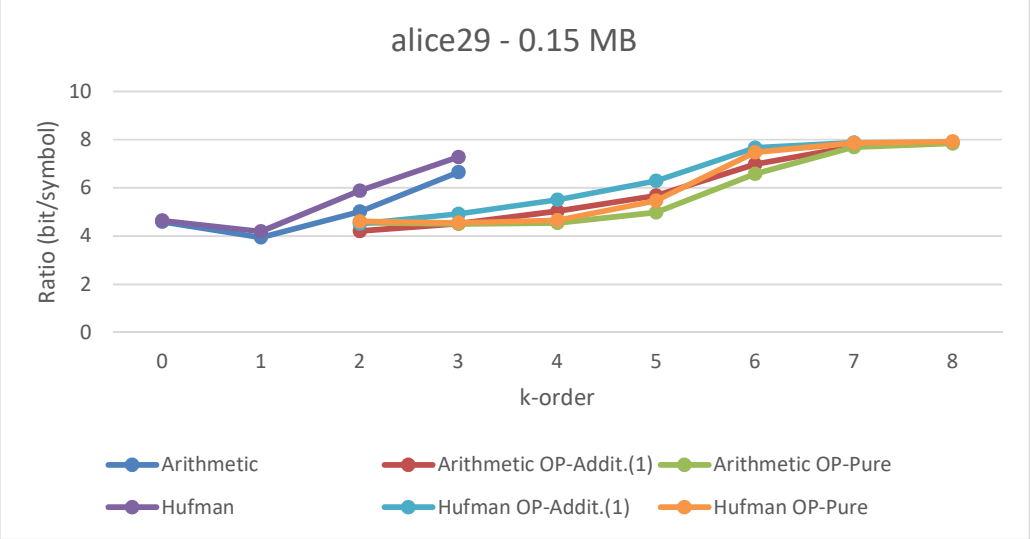
6.2 İşlenen Dosyalar ve Tüm Yöntemlerin Karşılaştırılması

Uygulama tarafından işlenen dosyaların detayları grafiklerde gösterilmiştir. Grafiklerde aynı dosya üzerinde tüm yöntemlerin uygulandığı görülebilmektedir. Yöntemler uygulanırken Standard yaklaşımda k = 0, 1, 2 ve 3 için uygulanmış, sıra korumalı (OP) yöntemlerde ise k = 2, 3,..,8 için uygulanmıştır. Aynı grafik üzerinde tüm sonuçlar kıyaslanabilmektedir.



Şekil 6.9 : Dosya : kennedy.xls – Tüm yöntemler.

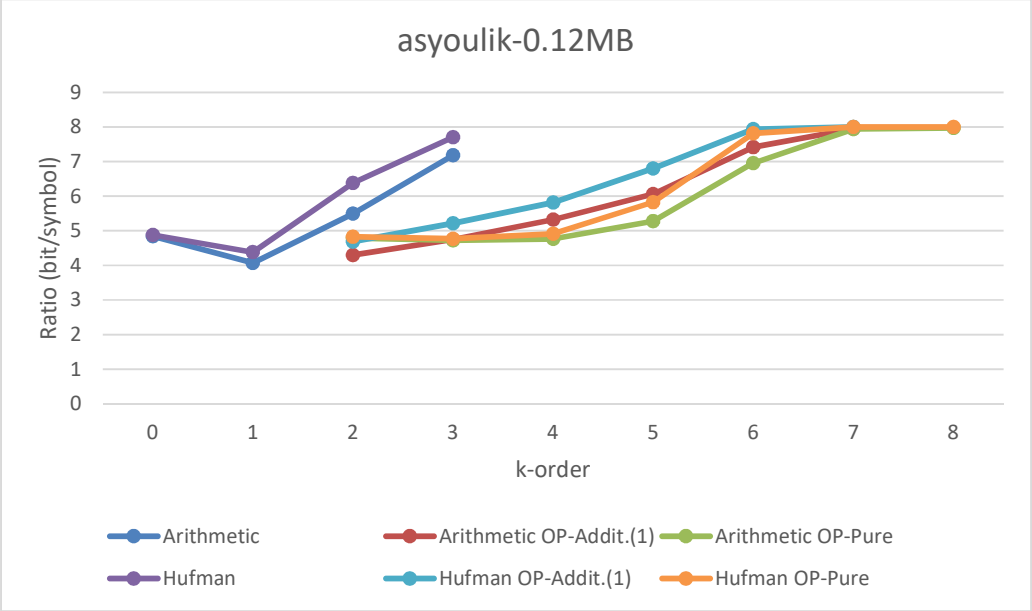
Şekil 6.9'da bir excel dosyası işlenmiş ve sonuçları gösterilmiştir. Bu işlemde OPPM yaklaşımının klasik yaklaşıma (Arithmetic/Hufman) göre daha iyi sonuç verdiği görülmektedir. 5-order'dan sonra bir iyileşme görülmemiştir. Burda Ek indeks yaklaşımlı OPPM yönteminin (OP-Addit.(1)), standard OPPM (OP-Pure) yaklaşamından bir farkı olmadığı gözlenmiştir. En ideal sonucun Arithmetic OP-Pure 5-order olduğu gözlemlenmektedir. Kullanılan model sayısı 5! = 120'dir. Standard yaklaşımda en iyi sonuç k = 2-order değerinde görülürken, bu değerde kullanılan model sayısının 256^2 = 65.536 olduğu görülmektedir. Bu değerler ciddi bir kaynak kazanımı sağlandığını göstermektedir. Ek indeks yaklaşımı da standard OPPM yaklaşımına benzer sonuçlar vermis, ancak buradaki model sayısı k!xAlfabe uzunluğu = 5!x256 = 30.720 dir. Klasik yaklaşıma göre model sayısı yarı yarıya azalmış olsa da, standard OPPM yaklaşımı ciddi bir kaynak katkısı sağlamıştır.



Şekil 6.10 : Dosya : alice29 – Tüm yöntemler.

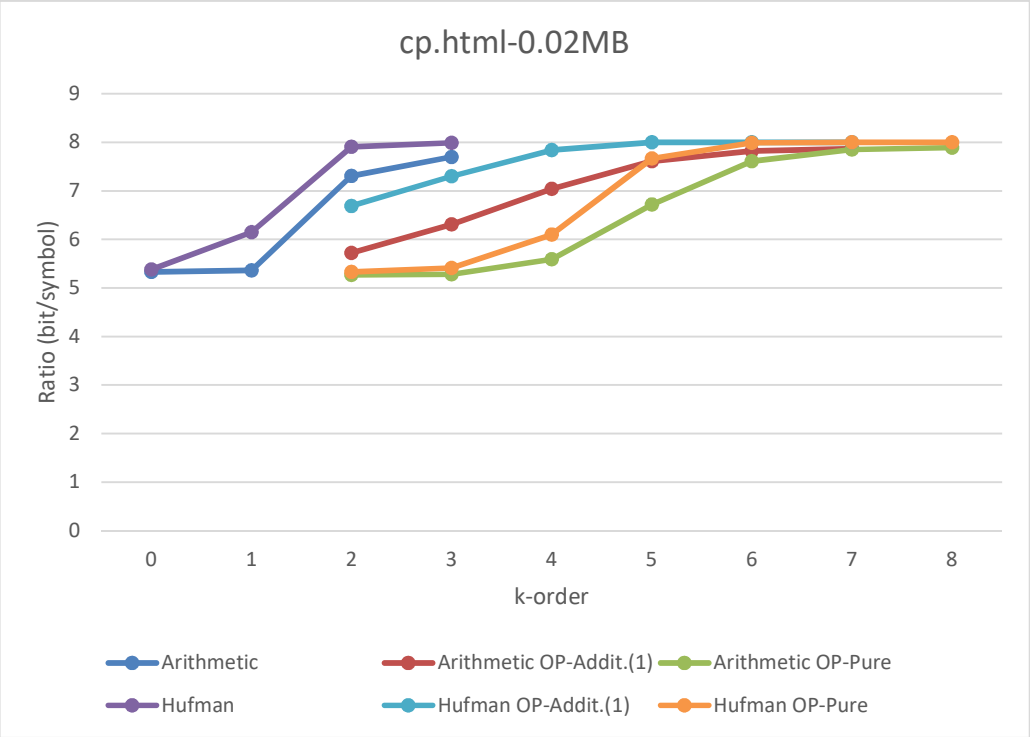
Şekil 6.10'da bir roman dosyası işlenmiş ve sonuçları gösterilmiştir. Bu işlemde OPPM yaklaşımının klasik yaklaşıma (Arithmetic/Hufman) göre benzer sonuç verdiği görülmektedir. 2-order'dan sonra bir iyileşme görülmemiştir. Burda Ek indeks yaklaşımlı OPPM yönteminin (OP-Addit.(1)), standard OPPM (OP-Pure) yaklaşamından bir farkı olmadığı gözlenmiştir. En ideal sonucun Arithmetic OP-Pure 2-order olduğu gözlemlenmektedir. Kullanılan model sayısı 2! = 2'dir. Standard yaklaşımda en iyi sonuç k = 1-order değerinde görülürken, bu değerde kullanılan model sayısının 256^1 = 256 olduğu görülmektedir. Bu değerler ciddi bir kaynak kazanımı sağlandığını göstermektedir. Ek indeks yaklaşımı da standard OPPM

yaklaşımına benzer sonuçlar vermis, ancak budaki model sayısı k!xAlfabe uzunluğu = 2!x256 = 512 dir. OP-Addit.(1) yaklaşımı, OP-Pure yaklaşımından çok az bir miktar daha iyi olmasına rağmen, bellek kullanımları arasında ciddi fark olduğu için ve aradaki fark önemsenmeyecek seviyede olduğu için OP-Pure yöntemi seçilmiştir.



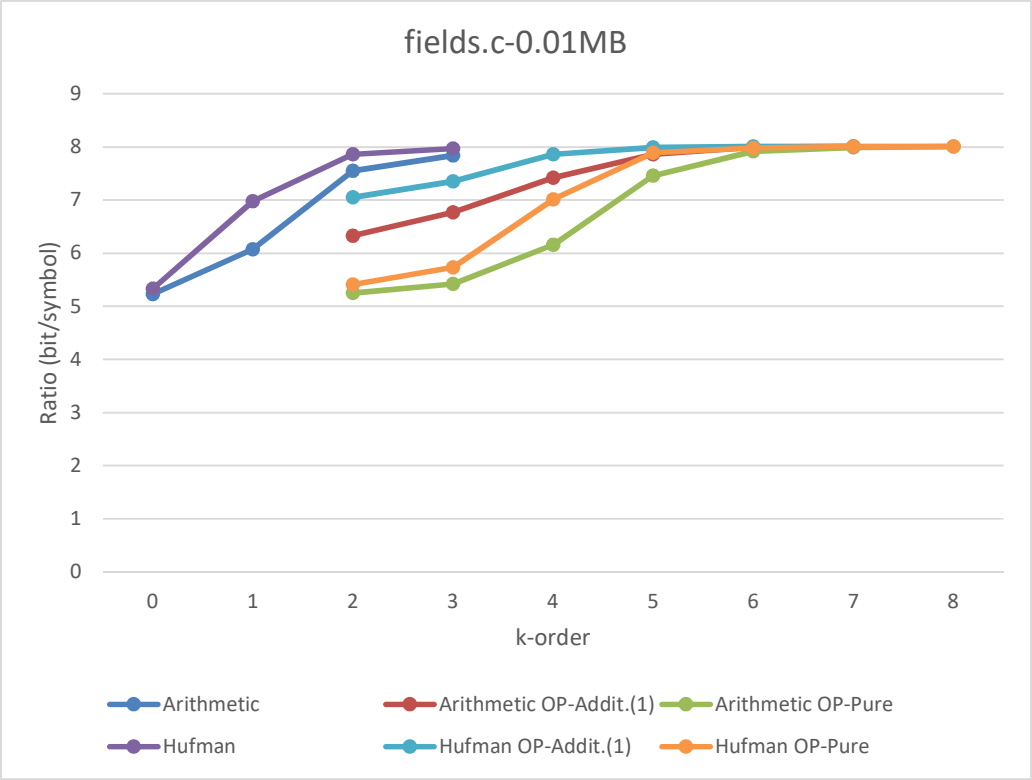
Şekil 6.11 : Dosya : asyoulik – Tüm yöntemler.

Şekil 6.11'de bir metin dosyası işlenmiş ve sonuçları gösterilmiştir. Bu işlemde de aynı şekilde OPPM yaklaşımının klasik yaklaşıma (Arithmetic/Hufman) göre benzer sonuç verdiği görülmektedir. 2-order'dan sonra bir iyileşme görülmemiştir. Burda Ek indeks yaklaşımlı OPPM yönteminin (OP-Addit.(1)), standard OPPM (OP-Pure) yaklaşamından bir farkı olmadığı gözlenmiştir. En ideal sonucun Arithmetic OP-Pure 2-order olduğu gözlemlenmektedir. Kullanılan model sayısı 2! = 2'dir. Standard yaklaşımda en iyi sonuç k = 1-order değerinde görülürken, bu değerde kullanılan model sayısının 256^1 = 256 olduğu görülmektedir. Bu değerler ciddi bir kaynak kazanımı sağlandığını göstermektedir. Ek indeks yaklaşımı da standard OPPM yaklaşımına benzer sonuçlar vermis, ancak budaki model sayısı k!xAlfabe uzunluğu = 2!x256 = 512 dir. OP-Addit.(1) yaklaşımı, OP-Pure yaklaşımından çok az bir miktar daha iyi olmasına rağmen, bellek kullanımları arasında ciddi fark olduğu için ve aradaki fark önemsenmeyecek seviyede olduğu için OP-Pure yöntemi seçilmiştir.



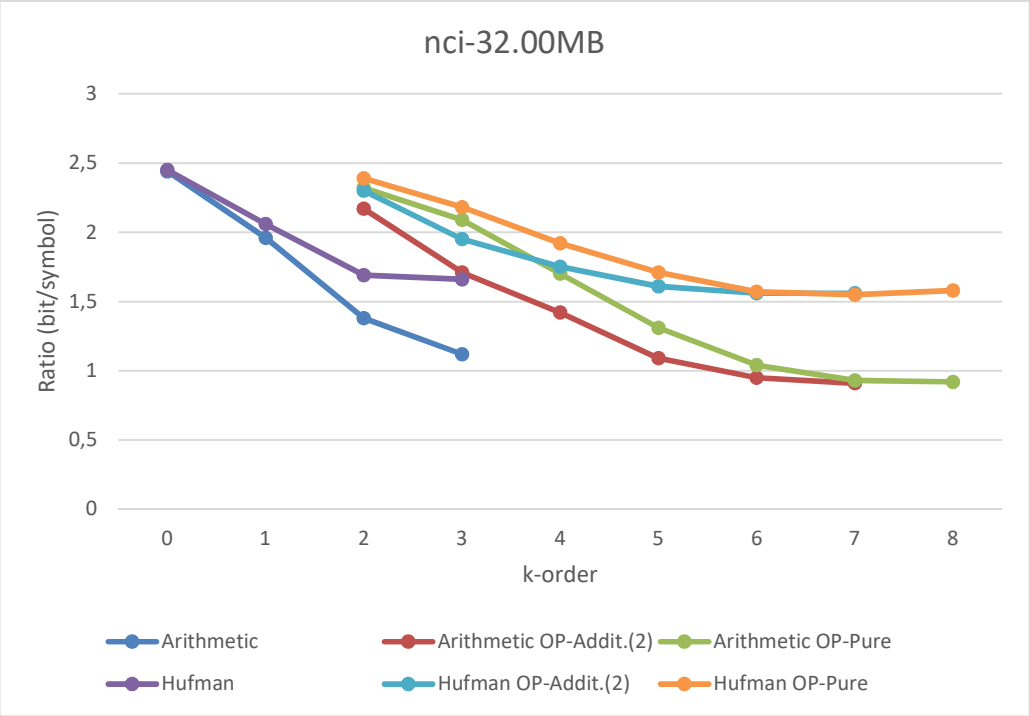
Şekil 6.12 : Dosya : cp.html – Tüm yöntemler.

Şekil 6.12'de bir web sayfası içeriği dosyası işlenmiş ve sonuçları gösterilmiştir. Bu işlemde de aynı şekilde OPPM yaklaşımının klasik yaklaşıma (Arithmetic/Hufman) göre benzer sonuç verdiği görülmektedir. 2-order'dan sonra bir iyileşme görülmemiştir. Burda Ek indeks yaklaşımlı OPPM yönteminin (OP-Addit.(1)), standard OPPM (OP-Pure) yaklaşamından bir farkı olmadığı gözlenmiştir. En ideal sonucun Arithmetic OP-Pure 2-order olduğu gözlemlenmektedir. Kullanılan model sayısı 2! = 2'dir. Standard yaklaşımda en iyi sonuç k = 1-order değerinde görülürken, bu değerde kullanılan model sayısının $256^1 = 256$ olduğu görülmektedir. Bu değerler ciddi bir kaynak kazanımı sağlandığını göstermektedir. Ek indeks yaklaşımı da standard OPPM yaklaşımına benzer sonuçlar vermis, ancak budaki model sayısı k!xAlfabe uzunluğu = 2!x256 = 512 dir. OP-Addit.(1) yaklaşımı, OP-Pure yaklaşımından çok az bir miktar daha iyi olmasına rağmen, bellek kullanımları arasında ciddi fark olduğu için ve aradaki fark önemsenmeyecek seviyede olduğu için OP-Pure yöntemi seçilmiştir.



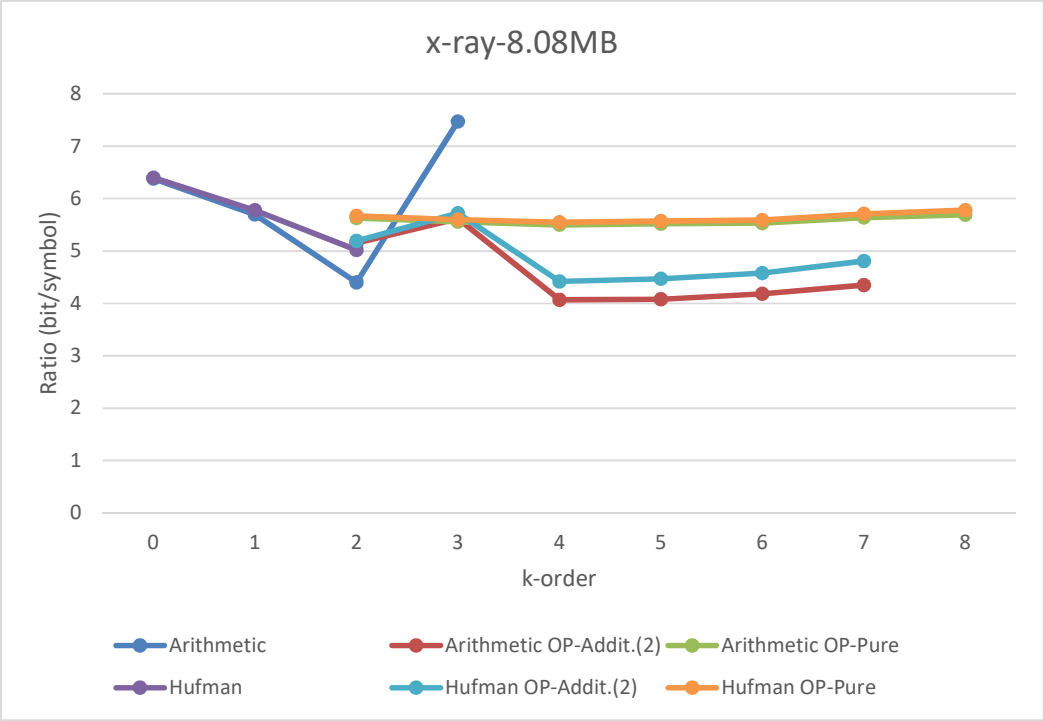
Şekil 6.13 : Dosya : fields.c – Tüm yöntemler.

Şekil 6.13'de bir roman dosyası işlenmiş ve sonuçları gösterilmiştir. Bu işlemde OPPM yaklaşımının klasik yaklaşıma (Arithmetic/Hufman) göre benzer bir sonuç verdiği görülmektedir. Standar yaklaşımda 0-order, OPPM yaklaşımında 2-order'dan sonra bir iyileşme görülmemiştir. Burda Ek indeks yaklaşımlı OPPM yönteminin (OP-Addit.(1)), standard OPPM (OP-Pure) yaklaşamından daha kötü sonuç verdiği gözlenmiştir. En ideal sonucun Standart Arithmetic 0-order olduğu gözlemlenmektedir. Kullanılan model sayısı 0! = 1'dir. OPPM yaklaşımda en iyi sonuç k = 2-order değerinde görülürken, bu değerde kullanılan model sayısının 2! = 2 olduğu görülmektedir. Bu değerler bir kaynak kazanımı sağlanmadığını göstermektedir. Ek indeks yaklaşımı da standard OPPM yaklaşımına göre daha kötü sonuçlar vermis, buradaki model sayısı k!xAlfabe uzunluğu = 2!x256 = 256 dır.



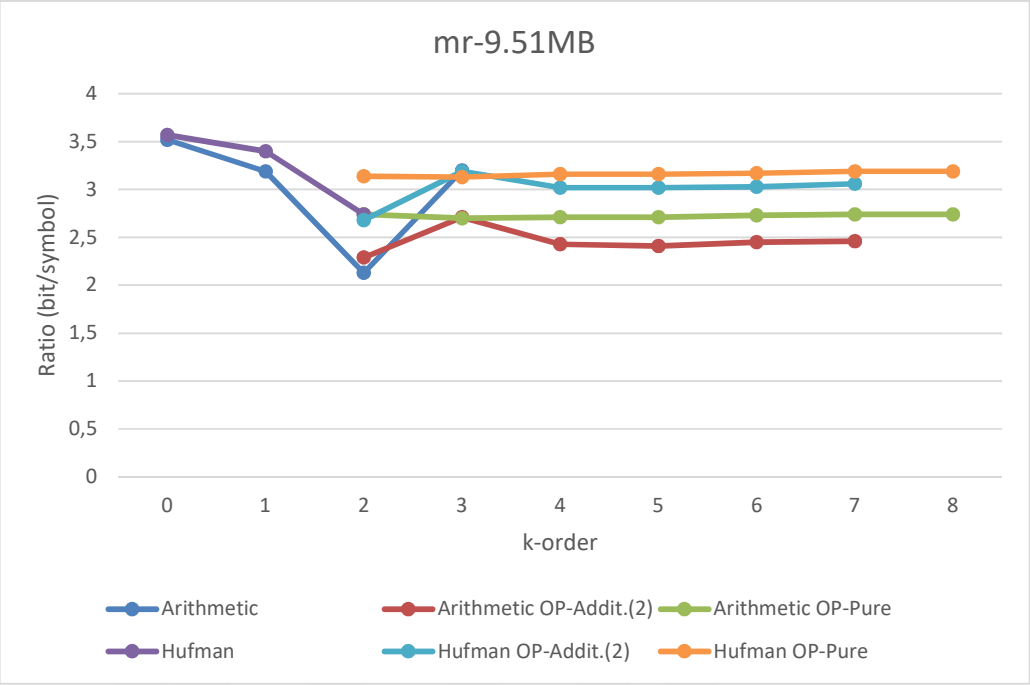
Şekil 6.14 : Dosya : nci – Tüm yöntemler.

Şekil 6.14'de bir roman dosyası işlenmiş ve sonuçları gösterilmiştir. Bu işlemde OPPM yaklaşımının klasik yaklaşıma (Arithmetic/Hufman) göre benzer, hatta daha iyi sonuç verdiği görülmektedir. 8-order'dan sonra bir iyileşme görülmemiştir. Burda Ek indeks yaklaşımlı OPPM yönteminin (OP-Addit.(2)), standard OPPM (OP-Pure) yaklaşamından bir farkı olmadığı gözlenmiştir. En ideal sonucun Arithmetic OP-Pure 7-order olduğu gözlemlenmektedir. Kullanılan model sayısı 7! = 5040'dir. Arithmetic OP-Pure 8-order yaklaşımı çok az daha iyi olsa da, kaynak kullanımı arasında ciddi fark olduğundan 8-order daki avantaj önemsenmeyecek seviyeye düşüyor. Standard yaklaşımda en iyi sonuç k = 3-order değerinde görülürken, bu değerde kullanılan model sayısının 256^3 = 16.777.216 olduğu görülmektedir. Bu değerler çok ciddi bir kaynak kazanımı sağlandığını göstermektedir. Ek indeks yaklaşımı da standard OPPM yaklaşımına benzer sonuçlar vermis, ancak buradaki model sayısı k!xAlfabe uzunluğu = 7!x256 = 1.290.240 tır. Klasik yaklaşıma göre model sayısı iyi bir seviyede azalmış olsa da, standard OPPM yaklaşımı ciddi bir kaynak katkısı sağlamıştır.



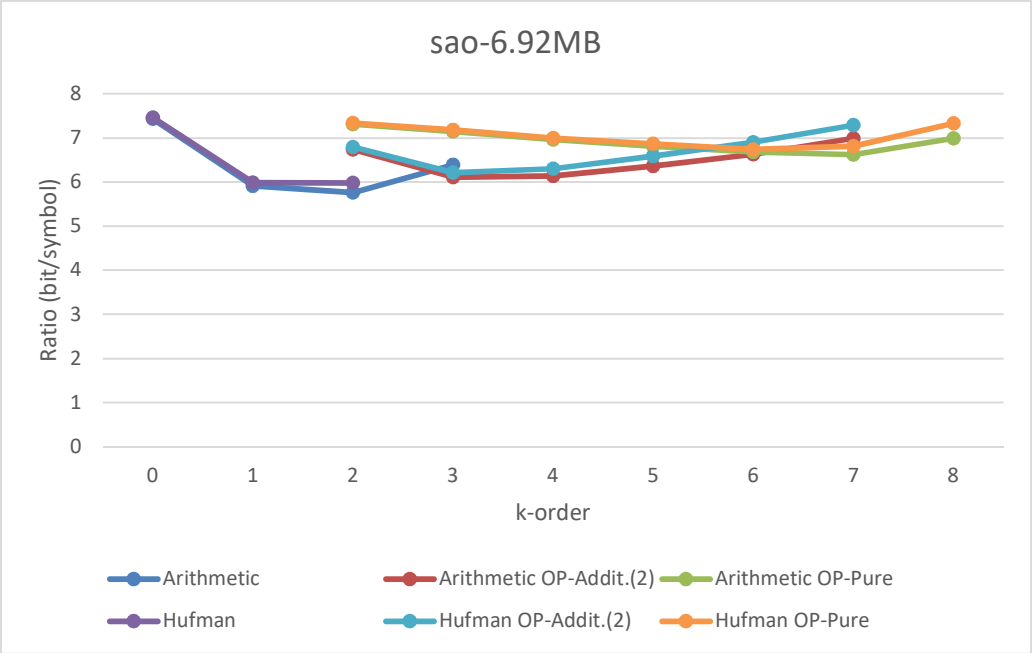
Şekil 6.15 : Dosya : x-ray – Tüm yöntemler.

Şekil 6.15'da bir tıbbi röntgen dosyası işlenmiş ve sonuçları gösterilmiştir. Bu işlemde OPPM yaklaşımının klasik yaklaşıma (Arithmetic/Hufman) göre daha iyi sonuç verdiği görülmektedir. 4-order'dan sonra bir iyileşme görülmemiştir. Burda Ek indeks yaklaşımlı OPPM yönteminin (OP-Addit.(2)), standard OPPM (OP-Pure) yaklaşamından daha iyi bir sonuç verdiği gözlenmiştir. En ideal sonucun Arithmetic OP-Addit.(2) 4-order olduğu gözlemlenmektedir. Kullanılan model sayısı 4!x256 = 6.144'dür. Standard yaklaşımda en iyi sonuç k = 2-order değerinde görülürken, bu değerde kullanılan model sayısının 2562 = 65.536 olduğu görülmektedir. Bu değerler ciddi bir kaynak kazanımı sağlandığını göstermektedir. Standart OPPM yaklaşımı kaydadeğer bir iyileşme sağlayamamıştır. Hufman OP-Addit.(2) yaklaşımı da aritmetikteki yaklaşıma yakın bir sonuç vermiştir. Ancak aritmetik yaklaşım çok az bir miktar daha iyi sonuç vermiştir.



Şekil 6.16 : Dosya : mr – Tüm yöntemler.

Şekil 6.16'da bir tıbbi mr dosyası işlenmiş ve sonuçları gösterilmiştir. Bu işlemde OPPM yaklaşımının klasik yaklaşıma (Arithmetic/Hufman) benzer bir sonuç verdiği görülmektedir. 2-order'da iyi bir sonuç yakalnmış olup, 4-order'dan sonra bir iyileşme görülmemiştir. Burda Ek indeks yaklaşımlı OPPM yönteminin (OP-Addit.(2)), standard OPPM (OP-Pure) yaklaşamından ddaha iyi bir sonuç verdiği gözlenmiştir. En ideal sonucun Arithmetic OP-Addit.(2) 2-order olduğu gözlemlenmektedir. Kullanılan model sayısı 2!x256 = 512'dir. Standard yaklaşımda en iyi sonuç k = 2-order değerinde görülürken, bu değerde kullanılan model sayısının 256^2 = 65.536 olduğu görülmektedir. Bu değerler ciddi bir kaynak kazanımı sağlandığını göstermektedir. 2-order Standart Arithmetic yaklaşım, 2-order Arithmetic OP-Addit.(2) yaklaşımından çok az daha iyi sonuç vermiştir. Ancak kaynak kullanımındaki kazanç göz önünde bulundurulduğunda bu fark ihmal edilebilir. Ek indeks yaklaşımı da standard OPPM yaklaşımından azımsanmayacak kadar daha iyi sonuçlar vermiştir.



Şekil 6.17 : Dosya : sao – Tüm yöntemler.

Şekil 6.17'de bir roman dosyası işlenmiş ve sonuçları gösterilmiştir. Bu işlemde OPPM yaklaşımının klasik yaklaşıma (Arithmetic/Hufman) benzer bir sonuç verdiği görülmektedir. OPPM yönteminde 3-order'da iyi bir sonuç yakalnmış olup, 4-order'dan sonra bir iyileşme görülmemiştir. Burda Ek indeks yaklaşımlı OPPM yönteminin (OP-Addit.(2)), standard OPPM (OP-Pure) yaklaşamından daha iyi bir sonuç verdiği gözlenmiştir. En ideal sonucun Arithmetic OP-Addit.(2) 3-order olduğu gözlemlenmektedir. Kullanılan model sayısı 3!x256 = 1536'dır. Standard yaklaşımda en iyi sonuç k = 2-order değerinde görülürken, bu değerde kullanılan model sayısının 256^2 = 65.536 olduğu görülmektedir. Bu değerler ciddi bir kaynak kazanımı sağlandığını göstermektedir. 2-order Standart Arithmetic yaklaşım, 3-order Arithmetic OP-Addit.(2) yaklaşımından çok az daha iyi sonuç vermiştir. Ancak kaynak kullanımındaki kazanç göz önünde bulundurulduğunda bu fark ihmal edile de bilir, veya Standart yaklaşım tercih edilebilir.

KAYNAKLAR

Amir Said, 2003: "Arithmetic Coding," in Lossless Compression Handbook, (K. Sayood, Ed.), Academic Press, San Diego, CA.

A. Amir, Y. Aumann, P. Indik, (2009). Efficient computations of l_1 and $l_{INFINITY}$ rearrangement distances Theor. Comput. Sci., 410 (43), pp. 4382-4390

A. Amir, M. Farach, S. Muthukrishnan, 1994: Alphabet dependence in parameterized matching Inf. Process. Lett., 49 (3), pp. 111-115.

A.V. Aho., 2014: Algorithms for finding patterns in strings, Handbook of Theoretical Computer Science, vol. A: Algorithms and Complexity, pp. 255-300.

Boyer, R.S., Moore, J.S., 1977. A fast string searching algorithm. Communications of the ACM 20(10), 762–772

Crochemore, M., Iliopoulos, C., Kociumaka, T., Kubica, M., Langiu, A., Pissis, S., Radoszewski, J., Rytter, W., Walen´, T., 2013. Order-preserving incomplete suffix trees and order-preserving indexes. In: Kurland, O., Lewenstein, M., Porat, E. (eds.) String Processing and Information Retrieval, Lecture Notes in Computer Science, vol. 8214, pp. 84–95..

E. Cambouropoulos, M. Crochemore, C.S. Iliopoulos, L. Mouchard, Y.J. Pinzon, 2002: Algorithms for computing approximate repetitions in musical sequences Int. J. Comput. Math., 79 (11), pp. 1135-1148.

Fano, R. M., 1944: The Transmission of Information, Technical Report 65, Research Laboratory of Electronics, MIT, Cambridge, MA.

Golomb, S. W., 1966: Run length encodings. IEEE Transactions on Information Theory, Vol. 12, No. 3.

Jinil K., Peter E., Rudolf F., Seaok-Hee H., Costas S. I., Takeshi T., 2014: Theoretical Computer Science, Volume 525, pp. 68-79.

Kim, J., Eades, P., Fleischer, R., Hong, S.H., Iliopoulos, C.S., Park, K., Puglisi, S.J., Tokuyama, T., 2013. Order preserving matching. CoRR abs/1302.4064.

Knuth, D.E., Morris, J.M., Pratt, V.R., Fast pattern matching in strings. SIAM Journal on Computing 6(2), 323-350

Larmore, L. L., and D. S. Hirschberg, 1990: A fast algorithm for optimal length-limited Huffman codes. Journal of the ACM, Vol. 37, pp. 464-473.

Pigeon, S., and Y. Bengio, 1997. A Memory-Efficient Huffman Adaptive Coding Algorithm for Very Large Sets of Symbols, Technical Report 1081, Drpartement d'Informatique et Recherche Oprrationnelle, Universit6 de Montrral, Montrral, Qu6bec, Canada.

Pasco, R., 1976. Source Coding Algorithms for Fast Data Compression, Ph.D. thesis. Department of Electrical Engineering, Stanford University, Stanford, CA.

Pawel G., Przemysl U., 2013. Order-preserving pattern matching with k mismatches, *PhD Thesis,* Inria Bordeaux Sud-Ouest, France.

Siemifiski, A., 1998: Fast decoding of the Huffman codes. Information Processing Letters, Vol. 26, pp. 237-241.

Tamanna C., Jorma T., 2014. Order-Preserving Matching with Filtration, Supported by the Academy of Finland, Aalto University, Aalto, Finland

Ziv J., A. Lempel, 1978. Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory, Vol. 24, No. 5, pp. 530-536.

ÖZGEÇMİŞ

Ad Soyad **:** Muhammed Veysel KINGIR

Doğum Yeri ve Tarihi: Siirt – 30.11.1983

E-Posta **:** veyselkingir@hotmail.com

EĞİTİM:

Lisans: 2006, Kocaeli Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği

MESLEKİ DENEYİM VE ÖDÜLLER:

- *Primefor Bilişim Hizmetleri:* 2018 Haziran - Kıdemli Yazılım Uzmanı
- *Cardtek Bilişim:* Eylül 2012 – Haziran 2018, Teknoloji Grup Müdürü
- *IBTECH:* 2011 Temmuz – 2012 Eylül, Kıdemli Yazılım Uzmanı
- *Banksoft Bilişim:* 2005 Mart – 2010 Aralık, Yazılım Geliştirme Uzmanı