MYCIELSKI ÖNGÖRÜ ALGORİTMASI ÜZERİNE SIKIŞTIRMA VE RASSALLIK UYGULAMALARI

Mehmet FİDAN

Yüksek Lisans Tezi

Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Ağustos-2006

JÜRİ VE ENSTİTÜ ONAYI

Mehmet Fidan'ın **"Mycielski Öngörü Algoritması Üzerine Sıkıştırma ve Rassallık Uygulamaları"** başlıklı **Elektrik-Elektronik Mühendisliği** Anabilim Dalındaki, Yüksek Lisans tezi 21/07/2006 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı-Soyadı	**İmza**
Üye (Tez Danışmanı)	**:Doç. Dr. ÖMER NEZİH GEREK**	**…….**
Üye	**:Prof. Dr. ATALAY BARKANA**	**…….**
Üye	**:Doç. Dr. BİLGİNER GÜLMEZOĞLU**	**…….**

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun …… tarih ve …… sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ÖZET

Yüksek Lisans Tezi

MYCIELSKI ÖNGÖRÜ ALGORİTMASI ÜZERİNE SIKIŞTIRMA VE RASSALLIK UYGULAMALARI

Mehmet FİDAN

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Ömer Nezih GEREK

2006, 42 Sayfa

Bu tezde Mycielski öngörücüsünün öngörüye dayalı sıkıştırma amaçlı ve şifreleme uygulamaları için rassal sayı üretiminde kullanılışı incelenmiştir. Mycielski öngörücüsü, sınırsız geçmiş kullanan gerekirci bir öngörücüdür. Bu nedenle harcanan süre uzun boyutlu sinyallerde hem sıkıştırma hem de şifreleme uygulamalarını olanaksız kılmaktadır. Bu çalışmanın ilk aşamasında öngörücü Mycielski algoritmasında zaman iyileştirilmesi yapılmış ve sıkıştırma algoritması olarak kullanılmıştır. Uygulanan zaman iyileştirme yöntemi, literatürde LZ78 olarak bilinen sıkıştırma yöntemindeki metoda benzediğinden, önerilen zaman iyileştirmeli yönteme Mycielski78 adı verilmiştir. Öngörücülerin kullanılabileceği bir diğer alan da şifreleme amaçlı ve başlangıç anahtar dizisine bağımlı rassal sayı üretimidir. Bu çalışmanın ikinci aşamasında Mycielski yöntemini temel alıp öngörüye dik çıktılar üreten ve Antimycielski şeklinde adlandırılan yeni bir sayı üreteci tanıtılmakta ve rassallık kalitesi sınanmaktadır. Algoritmanın yavaşlığı nedeniyle belli uzunluklar sonra öngörünün kullandığı geçmişi kesmek gerekmektedir. Bu çalışmada, anahtar seri uzunluğunun ve tutulan tarih uzunluğunun rastsallık üstündeki etkisi de incelenmektedir.

Anahtar Kelimeler: Öngörücü Sıkıştırma, Rassal Sayı Üreteci, Mycielski Öngörücüsi, Antimycielski, Şifreleme, Blok Kodlayıcı

ABSTRACT

Master of Science Thesis

COMPRESSION AND RANDOMNESS APPLICATIONS ON THE MYCIESKI PREDICTION ALGORITHM

Mehmet FİDAN

Anadolu University
Graduate School of Sciences
Electrical and Electronics Engineering Program

Supervisor: Assoc. Prof. Dr. Ömer Nezih GEREK
2006, 42 Pages

In this thesis, compression and pseudorandom number generation for encryption properties of the Mycielski algorithm are investigated. Mycielski predictor is a deterministic algorithm which uses infinite history. Therefore, the time complexity of the algorithm diverges for long sequences. In the first part of this work, a time reinforcement of Mycielski algorithm is discussed and the modified algorithm is used as compression algorithm. Due to the resemblence of the time improvement to the famous LZ78 compression algorithm, the proposed method is named the Mycielski78 algorithm. In the second part of this work, random number generation properties of the Mycielski predictor is investigated. An algorithm, called the Antimycielski algorithm, which produces sequences that are orthogonal to the Mycielski prediction output starting from an initial key sequence, is developed. The randomness quality and cryptography strength of the Antimycielski sequences are examined. Again, due to the complexity of the algorithm, the history used for making prediction that gets longer than a constant limit must be chopped. Performance effects of key length and history chopping length on randomness are examined.

Keywords: Predictive Coding, Random Number Generator, Mycielski Predictor, Antimycielski, Ciphering, Blok Ciphers.

TEŞEKKÜR

Bu çalışmada yardımlarından dolayı danışman hocam Doç. Dr. Ömer Nezih GEREK'e en içten teşekkürlerimi sunarım. Bu çalışma süresince her zaman yanımda olan ve bana her türlü desteği veren aileme de teşekkür ederim.

Mehmet FİDAN
Ağustos-2006

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

1. GİRİŞ

1.1. Genel Bakış

Öngörü bir sistemle ilgili bilinmeyen bir veri hakkında, bilinen veriler kullanılarak elde edilen kesin olmayan kanıdır. Bu kanıyı elde etmek için kullanılan yöntemlerin genel ismi öngörücü olarak geçmektedir. Öngörücüler hava öngörüyü, güç analizi, biyoinformatikte olası DNA, RNA gibi yapı taşlarının sıralama öngörüleri, herhangi bir sistemin gelecekte göstereceği performans öngörüyü gibi konularda kullanılabileceği gibi öngörüye dayalı veri sıkıştırmada ve ilk başta ilgisiz görülmekle birlikte rassal sayı üretiminde kullanılabilmektedir.

Öngörücüler izledikleri yola göre olasılıksal ve gerekirci olarak iki ana gruba ayrılmaktadırlar. Olasılıksal yaklaşımda verilen girdiler ışığında öngörü edilecek değer için tek bir sonuç değil, olası sonuçlar kümesi çıkartılır. Gerekirci yaklaşımda ise öngörü değeri için çıkartılabilecek tek bir değer vardır.

Mycielski algoritması herhangi ayrık bir sinyalin bilinmeyen örneğinin değerini hesaplamaya çalışan bir öngörücüdür. Bu öngörücü dizinin verilen elemanlarından sonraki elemanını öngörürken tek bir sonuca döndüğü için gerekirci gruba girmektedir. Ayrıca bir sonraki anın öngörüsünde içinde bulunulan an ve daha önceki değerleri kullandığı için nedensel bir fonksiyon olarak da belirtilebilir. İncelerken geçmiş elemanlarının sınırlı bir kısmını değil, tamamını kullanır. Bu öngörücünün çıkış noktası örnekleri birbiriyle ilişkili bir sinyalin, içindeki yapıların gelecekte de tekrarlanacağı varsayımıdır. Doğal olarak tekrarlanan en uzun yapı, en güçlü tahmin bilgisini taşıyan yapıdır. Algoritma yaptığı incelemede bu dizi içindeki tekrar eden alt dizileri kontrol etmektedir. Tekrar eden alt dizilere dayalı öngörülerde bulunmasından dolayı, yapılan öngörüler ancak ve ancak örnekleri birbiriyle ilişkili olan sinyallerde anlam kazanmaktadır. Tamamen rassal örnekler içeren sinyallerde ise tekrar eden alt diziler bulunamamasından veya bulunsa dahi anlamlı bir bilgi vermemesinden dolayı, gerçek değere yakın öngörü yapma olasılığı düşüktür. Öngörü edilen değerden önceki tüm değerleri, bir diğer değişle sınırsız geçmişi kullanması Mycielski algoritmasına yaptığı öngörülerde büyük bir güç vermektedir. Mycielski algoritmasının güçlü olmasının diğer bir nedeni ise öngörüyü en uzun benzer iki alt diziyi, dolayısıyla aynı hareketi devam ettirmekte en olası iki alt diziyi dikkate almasıdır.

1.2. Sıkıştırma Uygulamaları

Öngörücü sıkıştırma eski ve çok kullanılan bir yöntem olup, başarılı sıkıştırma sonuçları vermektedir. Bu tür sıkıştırma algoritmaları başlangıç olarak sinyalin her bir örneğini öngörü eder. Bu öngörülerin teker teker ilgili örnekle olan uzaklığını hesaplar ve bu uzaklıklardan yeni bir sinyal oluşturulur. Eğer sinyal kendi içinde ilişkiliyse, öngörü algoritması da mantıklı öngörülerde bulunabiliyorsa oluşan fark sinyali daha düşük bir entropi ve daha düşük bir enerji taşıyacaktır. Daha düşük entropi ve enerji taşıyan bir sinyal de başlangıç sinyaline göre daha az sayıda bitle ifade edilebilecektir. Mycielski [1] öngörücüsü elde ettiği başarılı öngörü serisi nedeniyle öngörücü bir sıkıştırma algoritmasının öngörü aşamasında kullanılabileceği fikrini vermektedir. Öngörücü her bir örneğin öngörüsünde kullanılıp, diğer fark sinyali oluşturma aşamalarının aynen uygulanabileceği düşünülmüştür.

Mycielski yöntemi sınırsız geçmişi kullanmakta ve geçmiş içinde tekrar eden alt dizileri incelerken ardışık taramalarda bulunmaktadır. Bu yöntemde öngörü için harcanan süre, kullanılan geçmişin boyutundaki artışa bağlı olarak, hem taranan alanın büyümesinden hem de taranacak alt dizi sayısının ve boyutlarının artmasından dolayı polinom olmayan bir artış gösterir**.** LZ77 ve LZ8 algoritmaları Mycielski algoritması ile tamamen farklı amaçlarda kullanılan sıkıştırma algoritmalarıdır. Fakat LZ77 [2]de sıkıştırma yaparken sinyal örneklerini inceler ve bu incelemeyi yaparken Mycielski gibi tüm geçmişi kullanır. Dolayısıyla Mycielski'ye benzer bir zaman sorunu LZ77'de de vardır. Bu zaman sorunu LZ78 algoritmasında sınırsız geçmişi bir sözlük yapısıyla ifade ederek aşılmıştır. LZ77'nin polinom zamana indirilmesini sağlayan LZ78 algoritmasının [3] uyguladığı değişiklik, bize Mycielski üzerinde zaman iyileştirmesinin benzer bir yapı ile sağlanabileceği fikrini vermiştir. Uygulamada, LZ78 algoritmasının sıkıştırma sonuçları, LZ77 algoritmasınınkiler kadar iyi olmamakla birlikte, sözlük tabanlı olmasından dolayı LZ78 daha hızlı çalışmaktadır. Bu sözlük özelliği Mycielski algoritmasında da kullanılarak, algoritma polinom zamana indirgenebilir. Bu çalışmada elde edilen hızlandırılmış Mycielski algoritmasına Mycielski78 adı verilmiştir.

Mycielski algoritması, öngörü için kendi elemanları arasındaki karşılaştırmalarda Hamming uzaklığını kullanır. Herhangi bir örnek eğer 8 bitle ifade ediliyorsa 256, 16 bitle ifade ediliyorsa 65536 farklı değer alabilmektedir. Hamming uzaklığı karşılaştırılan iki örneğin aldıkları değerlerin birbiriyle aynı olup olmadığını anlamakta kullanılır. Buna bağlı olarak, örnekler birbiriyle çok yakın değerler taşısa bile bu değerlerin birbirinden farklı olması, örneklerin farklı olarak değerlendirilmesi için yeterlidir. Bu özellik nedeniyle

birbirine yakın değerler taşıyan örneklerin verebileceği bilgi kullanılamamaktadır. Mycielski78 algoritmasında geçmiş bilgisinin kullanımındaki verimi arttırmak için Hamming uzaklığı ifadesi, belirtilen olumsuz yanı düzeltilecek şekilde tekrar tanımlanmıştır. Bu yeni uzaklık tanımında sadece örneklerin aldığı değerlerin aynılığı değil, alınan değerlerin birbirine yakınlığı da incelenmektedir. Değerlerinin büyüklük farkları belli bir tolerans sınırının içinde kalan örnekler bu yeni uzaklık ifadesinde aynı kabul edilmektedirler. Oluşturulan eski ve yeni Mycielski78 algoritmaları ve başlangıç Mycielski algoritmasıyla ilgili deneyler yapılmış ve elde edilen sonuçlar Bölüm-7'de ayrıntılarıyla gösterilmiştir.

1.3. Şifreleyici ve Rassallık Uygulamaları

Günümüz bilgisayar teknolojisinde kişiye veya kuruma özel verilerin güvenliği büyük bir sorun oluşturmaktadır. Bu verileri sadece istenen kullanıcıların kullanabilmesi, belli şifreleyicilerle bu verilerin şifrelenmesiyle mümkün olmaktadır. Blok şifreleyici yöntemleri de veri saklama için kullanabilecek yöntemlerin en çok kullanılanlarındandır. Bir blok şifreleyici verinin bölünmüş bloklarını birbirinden bağımsız olarak şifreler. Bu tip şifreleyiciler şifreleme ve konulan şifreyi çözmek için bir anahtar kelime girilmesini isterler. Kodlama ve kod çözme aşamalarında aynı anahtar kelimeyi kullandıkları için bu tip kodlayıcılar simetrik kodlayıcılar sınıfına girerler. Simetrik kodlamanın bir yolu başlangıç verisiyle şifreyi dışlamalı-veya işleminden geçirmektir. Bu işlem nedeniyle kodlayıcı, girilen anahtar kelimeye bağlı olarak üretilen rassal bir seriye ihtiyaç duymaktadır.

Bu noktada rassal sayı üretimi ciddi bir sorun haline gelmektedir. Bu çalışmada bu sorun için çözüm olarak Antimycielski sözde rassal sayı üreteci sunulmuştur. Antimycielski algoritması rassal bitleri, anahtar sözcük olarak düşünülebilecek ikili bir diziden üretmektedir. Seri, sonsuz geçmiş içindeki tekrar eden örüntüleri bulmaya dayalı bir öngörü yöntemi olan Mycielski kullanılarak oluşturulmaktadır. Antimycielski dizisi anahtar kelimeden başlanarak, hep bir sonraki gelecek örnek değerleri öngörü edilip, bu öngörü değerlerine dik değerlerin diziye eklenerek dizinin ardışık bir şekilde güncellenmesiyle oluşturulur. Üretilen ikili dizinin belirgin rassallık özellikleri taşıdığı gözlenmiş ve elde edilen sonuçlar bu algoritmanın gerçek bir rassal sayı üreteci sınıfına girebileceğini göstermiştir.

Başlangıç Mycielski algoritması sınırsız geçmiş kullanmasına bağlı olarak, bu algoritmadan türetilen Antimycielski algoritmasında da hesaplama karmaşıklığı ile karşılaşılmaktadır. Sıkıştırma uygulamaları için geliştirilen Mycielski78 öngörü için iyi sonuçlar verse de, ters öngörüleri rassallık özelliği gösteremediğinden Antimycielski'nin

karmaşıklığının çözülmesinde iyi bir çözüm olamamaktadır. Dolayısıyla karmaşıklığın giderilmesinde farklı bir çözüm uygulanmıştır. Önerilen bu çözüm ise geçmişin belli bir sınırla sınırlandırılmasıdır. Bu yöntemde geçmiş belli bir uzunluğa oluştuğunda dizinin sadece anahtar kelime uzunluğundaki son kısmı (kuyruğu) alınmakta ve her yeni zıt öngörüle belirlenen uzunluğa kadar tekrar genişletilmektedir. Bu noktada sorun geçmişe koyulan sınırın üretilen dizideki rassallığı bozmamasıdır. Üretilen rassal ikili seri ile şifrelenecek veri dışlamalı-veya işlemiyle maskeleme işleminden geçirilir. İşin doğası gereği maskeleme işlemi ile şifrelenen veri aynı yöntemle çözülmedikçe kullanıcı için gürültüden başka bir şey ifade etmeyecektir. Ayrıca anahtar kelimeyi bilmeden şifrelenen verinin çözülmesi zor olacağından, bu yöntem güçlü bir şifreleme yöntemi olarak kabul edilebilir.

Şifreleme yönteminin gücü, bu çalışmada rassallık analizi için en çok kullanılan testleri içeren NIST [7] test paketiyle sınanmıştır. Bu paket, sözde rassal sayı üreteçlerinin rassallık analizinde en çok kullanılan 14 testi içermektedir. Rassallık kalitesi geçilen test sayısıyla ölçülmekte, geçilemeyen testler de yöntemin hangi noktalarında açık olduğunu, oluşturulan şifrenin hangi yöntemlerle çözülebileceği hakkında fikir vermektedir. Bunun dışında şifrelenen veri başka rassal dizilerle ve birkaç biti değiştirilmiş anahtar kelimeyle çözülmeye çalışılmıştır. Bu şifrelenen kelimenin deneme sınama yöntemiyle şifrelenen bilginin çözülüp çözülemeyeceğini anlamak için önemli bir testtir. Yapılan deneylerde girilen anahtar kelime başlangıç kelimesine bir bit bile yakın olsa, çözülebilen veri oranının % 0,4'ü geçmediğini göstermektedir. Bu da herhangi bir rassal diziyle çözme denemesinde görülebilecek bir orandır. Deney sonuçları bu alakasız yöntemlerle üretilmiş rassal dizilerle çözme denemelerini de içermektedir. Deney sonuçlarına bakıldığında, Antimycielski algoritmasının şifreleme uygulamalarını kaldırabilecek rassal dizileri üretme başarısını göstermektedir. Rassal sayı üretme yöntemi Bölüm 5'de ayrıntılarıyla verilmiştir. Kullanılan NIST test paketi Bölüm 6'da anlatılmış ve elde edilen performans sonuçları Bölüm 8 ve 9'da sunulmuştur.

2. MYCIELSKI ALGORİTMASI

Mycielski öngörücüsü [1], Bölüm 1'de de belirtildiği gibi ayrık sinyaller de çalışan gerekirci bir fonksiyondur. Herhangi bir andaki sinyal örneğinin değerini, o andan önceki *tüm* geçmiş örnek değerlerini kullanarak öngörüde bulunur. Bu fonksiyon bu özelliğinden dolayı *sınırsız geçmiş* kullanan bir fonksiyon olarak da belirtilebilir. Öngörü, fonksiyon olarak aşağıdaki gibi gösterilebilir:

$$\hat{x}\left[n+1\right] = f_{n+1}\left(x\left[1\right], L\,, x\left[n\right]\right) \tag{2.1}$$

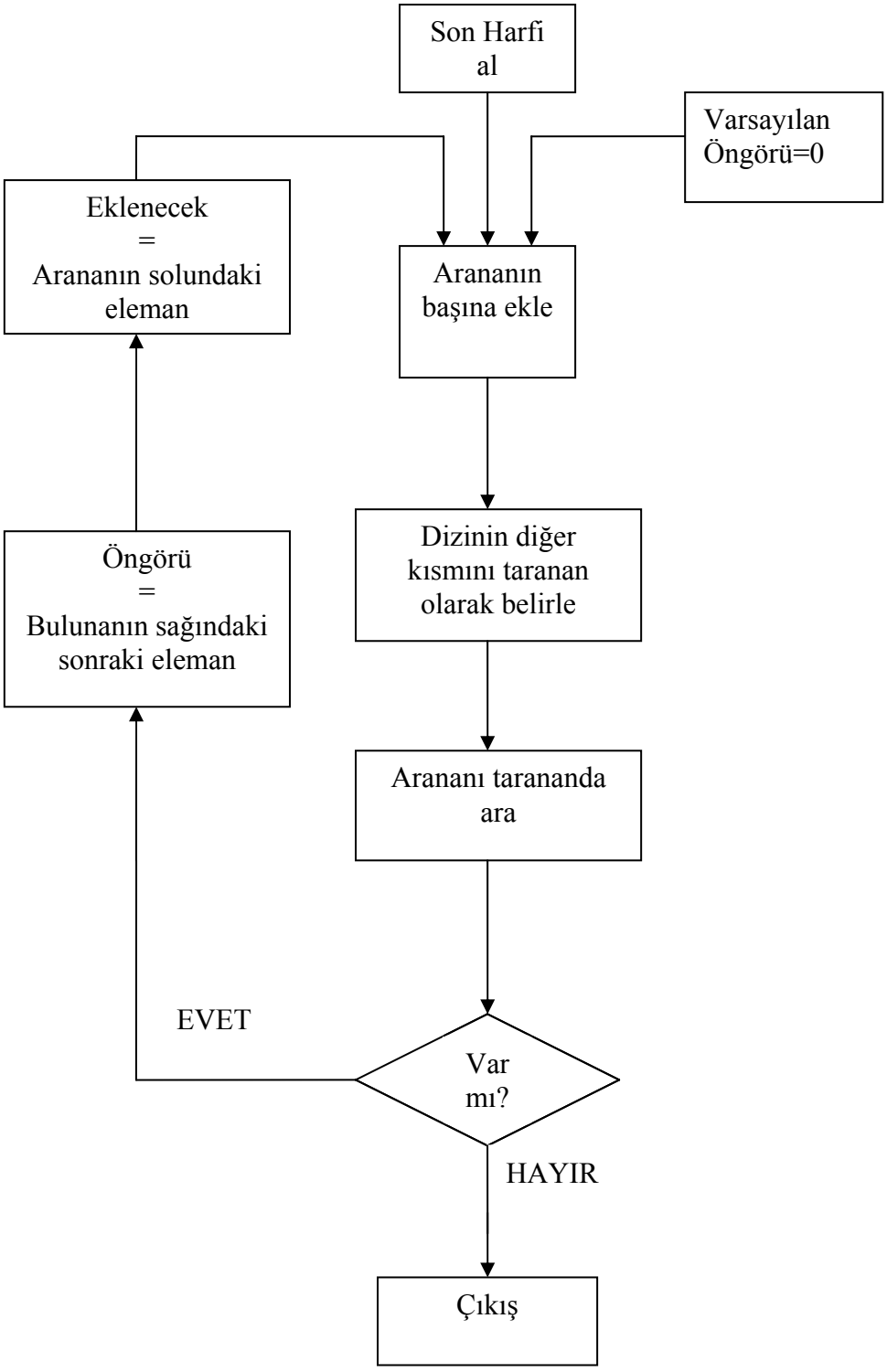
Bir sinyalin örneklerinin aldığı her farklı değer, sinyali tanımlayacak alfabeyi oluşturur. Sinyalin kendisi, bu alfabenin elemanlarının herhangi bir birleşiminden oluşan bir söz dizisi olarak kabul edilebilir. Buradaki f_{n+1} fonksiyonu, bu söz dizisi içindeki alt söz dizilerini birbiriyle karşılaştırarak bir değere döner. Bu fonksiyon "*başlangıç söz dizisinin içinde olup da bu söz dizisinin sonunda tekrarlanan en uzun alt söz dizisi*"ni bulmayı amaçlar.
Bu yöntemi uygulayarak sinyal içindeki aynı alt yapıları bulup, bu yapılardan sonraki davranışın da aynı olacağını varsayar. Bu varsayıma bağlı olarak bu alt söz dizisini kendinden önceki geçmiş içinde bulduğunda, bulduğu yerdeki alt diziden sonra gelen elemanı fonksiyonun görüntü değeri olarak belirler. Fonksiyon analitik olarak aşağıdaki gibi tanımlanabilir:

$$\begin{aligned} m &= \underset{L}{\operatorname{argmax}} \begin{cases} x\left[k\right] = x\left[n\right], x\left[k-1\right] = x\left[n-1\right] \\ ,\ldots, x\left[k-L+1\right] = x[n-L+1] \end{cases}, \\ f_{n+1} &= \hat{x}\left[n+1\right] = x\left[m\right] \end{aligned} \tag{2.2}$$

Tekrarlanan en uzun alt dizinin bulunması için başlangıç noktası sinyalin son elemanıdır. Bu alt dizi, sinyalin kendinden önceki geçmişi üzerinde sondan başa doğru taranır ve bulunduğunda, bulunduğu yerdeki alt diziden sonra gelen eleman $\hat{x}[n+1]$ olarak atanarak öngörü değeri güncellenir. Öngörü değerinin iyileştirilmesi için dizinin sonunda başlangıçta tek eleman olarak aldığımız dizi genişletilir (böylece 2 elemana ulaşır), ve arama tekrarlanır. Bu iterasyon, geçmiş sinyal dizisi içinde aranan dizi hiç bulunamayacak kadar uzun bir dizi elde edilene, ya da aranan dizi geçmiş dizisinden daha uzun duruma gelene kadar tekrarlanır. Bulunan son dizi, kendini son tarafta tekrar eden en uzun dizidir ve bir önceki tekrarın

takibinde bulunan sinyal elemanı, $\hat{x}[n+1]$ için en "olası" öngörüdür. Her dizi uzatması basamağında üzerinde arama yapılan geçmişten de aranana eklenen eleman çıkartılarak geçmiş daraltılır ve tarama tekrar yapılır. Taranan en kısa alt dizi bile geçmişte bulunamadığı durumlarda, öngörü değeri sıfir kabul edilmiştir. Ayrıca dizide hakkında öngörüde bulunabilecek ilk eleman $x[3]$'dür. $x[2]$ ve $x[1]$'den önce, karşılaştırma yapılabilecek bir geçmiş olmadığından dolayı, bu elemanların öngörü değerleri için varsayım olarak 0 atanır. Algoritma Şekil 2.1'deki akış şemasında ayrıntılı olarak anlatılmaktadır.

Bu algoritmanın çalışma süresindeki seri uzunluğuna bağlı artış, uygulamayı güçleştirir. Bu güçleşmenin nedeni hem dizinin kendi uzunluğunun, hem taranan alt dizi uzunluklarının hem de taranacak alt dizi bileşimleri sayısının artmasıdır. Doğal olarak bu artışlar birbiriyle çarpılarak zamanı etkileyeceğinden artış polinom olmayan bir davranış gösterecektir. Benzer bir zaman sorunu olan LZ77 algoritması ve onun polinom zamanda gerçekleşmesini sağlayan bir yaklaşımı olan LZ78 algoritması, Bölüm 3.3'de tanıtılmaktadır. Bu çalışmada yapısal benzerlikleri nedeniyle LZ78 algoritması sayesindeki bir hızlanma, Mycielski algoritması üzerinde de sağlanabileceği fikrini vermiştir.



Şekil 2.1. Mycielski algoritması akış şeması

3. VERİ SIKIŞTIRMA

3.1. Tanım

Gelişen iletişim ve internet teknolojisiyle birlikte insanların kullanabileceği ve birbirine iletebileceği veriler hem çeşitlilik hem de boyut açısından sürekli bir artış göstermektedir. Bunları taşıyacak ve aktaracak donanımlarda da doğal olarak ilerleme görülmekte fakat ilerlemedeki ivme yakalanamamaktadır. Bu noktada taşınan ve aktarılan verilerin yerini en aza indirmeye yarayan yazılımlara ihtiyaç duyulmaktadır. Bu yazılımların kullandığı yöntemler genel olarak veri sıkıştırma[10] olarak adlandırılır. Tek bir cümleyle tanımlanmak istenirse veri sıkıştırma, veriyi en sade şekilde ifade etme sanatıdır. Bu yöntemler genel olarak işlenen veriyi başlangıç haline göre daha az bitle ifade ederler.

3.1.1. Kodlama

Sıkıştırmayı tam olarak tanımlayabilmek için kodlama kavramından da bahsetmek gerekir. *Kodlama* sinyali tanımlayan alfabedeki her bir elemana(sembole) ikili dizilerin atanmasıdır. Bu ikili dizilerin her biri *kod kelimesi* olarak tanımlanır. Bu kod kelimelerinden oluşturulan kümelere *kod* denmektedir. Kod kelimelerinin uzunlukları sabitse, bu kod kelimelerinden oluşturulan kod *sabit uzunluklu kod*dur. Sembol başına düşen bit sayısı *kod oranı* olarak adlandırılır.

Veri sıkıştırma yöntemleri iki ana algoritmadan oluşur. Birincisi kodlayıcı olarak adlandırılır ve genel olarak veriyi daha az bitle ifade etmek için kullanılır. İkincisi ise kod çözücü olarak belirtilir. Kod çözücü daha az bitle ifade edilen yanı sıkıştırılan veriyi kullanılabilmesi için eski haline getirmeye çalışır. Kod çözücünün verdiği sonuca bağlı olarak sıkıştırma algoritmaları iki gruba ayrılabilir. Eğer bu sonuç verinin başlangıç haliyle her zaman birebir aynıysa yöntem kayıpsız sıkıştırma algoritması eğer değilse veri bir miktar değişime uğruyorsa kayıplı sıkıştırma algoritması olarak tanımlanabilir. Kayıpsız sıkıştırma algoritmaları veriyi korumakla birlikte, kayıplı sıkıştırma algoritmalarında veri sıkıştırma daha büyük bir oranla sağlanabilmektedir.

3.2. Performans Ölçümü

Sıkıştırma algoritmalarının performansları incelenirken sinyallerin entropileri ve varyansları büyük önem kazanmaktadır. Çünkü performanslar sinyallerin bu özellikleri üzerinden yaptığı değişikliklerle belirtilmektedir.

3.2.1. Entropi

Entropi kavramının çıkış noktası termodinamiktir. Termodinamik sistemdeki karmaşanın aynı zamanda kullanılabilir enerjinin ölçüsü olarak kullanılmaya başlanmıştır. Olasılık teorisinde [9] rassal bir deneyin belirsizlik ölçüsü olarak tanımlanmıştır. Bilgi teorisinde ise olasılık teorisine benzer olarak bir sinyalin taşıdığı değişikliklerin bir diğer değişle sinyalin rassallığının bir çeşit ölçüsü olarak kullanılmıştır. Bir sinyal ne kadar çok değişiklik taşıyorsa o kadar farklı bilgi taşıyor demektir. Bu noktadan bakıldığında sinyalin taşıdığı bilginin de bir ölçüsüdür. Bilgi teorisinde kullanılan en genel entropi tanımı Shannon entropisidir. Shannon entropisinde [11] bir olayın taşıdığı bilgi logaritmik bir denklemle ifade edilmiştir. Bu ifadede olayın olma olasılığı ile taşıdığı bilgi arasında ilişki kurulmuştur. Bu olasılık ne kadar düşükse, taşınan bilgi o kadar önemli ve sayısal olarak bakıldığında da o kadar büyük olmalıdır. Bu mantıktan yola çıkılarak, eğer herhangi bir A olayının taşıdığı bilgiyi i(A), olasılığını da P(A) olarak belirlersek aşağıdaki denklem bulunur:

$$i(A) = \log_b\left(\frac{1}{P(A)}\right) = -\log_b(P(A)) \tag{3.1}$$

Birbirinden bağımsız olayları içeren bir örnek uzay ele alındığında, bu uzayın entropisi olayların taşıdıkları entropilerin ortalamasıyla alınır. Bu ortalama alınırken her olayın bilgisi o olayın olasılığı ile ortalamaya doğru orantılı bir katkısı olmalıdır. Bu noktadan yola çıkıldığında entropi aşağıdaki gibi hesaplanır.

$$\left.\begin{array}{l} \bigcup (A_i) = S \\ \sum P(A_i) = 1 \end{array}\right\} \Rightarrow H(S) = -\sum \left[P(A_i) \cdot \log_b \left(P(A_i) \right) \right] \tag{3.2}$$

Entropi hesabında S örnek uzayı sinyal ve dizi, A_i de bu sinyali oluşturan alfabenin bir elemanı olarak kabul edilebilir. $P(A_i)$ ise bu elemanın S sinyali içinde görülme olasılığıdır. Bu

olasılık önceden de verilebileceği gibi bu elemanın sinyal içindeki sayısının sinyalin taşıdığı toplam örnek sayısına bölünerek de bulunabilir. Bu elemanlar bitlerle ifade edildiği ve bu yüzden ikili tabandaki büyüklüğü önemli olduğu için bilgi ve entropi hesabında logaritmik taban (Denklem (3.1) ve (3.2)'deki b'nin değeri) 2 olarak alınır. Buradan da anlaşılacağı üzere hesaplanan entropi ne kadar büyük olursa sinyal için kullanılacak bit sayısı o kadar fazla olacaktır.

Sıkıştırma algoritmaları bu noktada devreye girmektedir. Bu algoritmaların genel amaçlarından biri eleman başına düşen bit sayısını olabildiğince aza indirmek bir diğer değişle entropiyi düşürmektir. Entropi bu açıdan bütün sıkıştırma algoritmaları için önemli olduğu gibi kayıpsız sıkıştırma algoritmaları için daha da önemlidir. Çünkü entropi aynı zamanda sinyalin ne kadar çeşitli elemanlar içerdiğinin de bir göstergesidir. Kayıplı algoritmalarda birbirine benzer elemanlar aynı değerle ifade edilerek çeşitlilik azaltılabilir. Fakat kayıpsız algoritmalarda ise bu çeşitlilik aynen korunmalı ve buna rağmen ortalama bit sayısı (entropi) düşürülmelidir.

3.2.2. Varyans

Varyansın tanımlanabilmesi için öncelikle beklenen değer[13]in tanımlanması gerekir. Bir rassal değişkenin alabileceği değerlerin, bu değerleri alma olasılığı ile ağırlıklandırılmış ortalamasına beklenen değer denir. Eğer bu rassal değişkeni X, alabileceği değerleri x_i ile ifade edersek beklenen değer(E[X])in matematiksel tanımı (3.3) deki gibidir.

$$E[X] = \sum x_i p(x_i) \tag{3.3}$$

Beklenen değer rassal değişkenin alabileceği değer hakkında genel bir bilgi vermekle beraber, bu değerden olabilecek sapmalar hakkında yeterli bir bilgi vermemektedir. Eğer sapmanın yönüyle ilgilenecek olunursa, sapma rassal sayı ile beklenen değer farkının beklenen değerinden (ortalamasından) hesaplanabilir. Fakat bu bize belli bir aralık hakkında fikir vermez. Bu aralığın hesabında kullanılacak sapmayı hesaplamak için alınabilecek değerlerin beklenen değerden uzaklığının mutlak değerine bakmak gerekir. Bu noktada sapmayı hesaplamadan önce beklenen değerden olası uzaklıkların oluşturacağı fonksiyonun genel enerjisinin hesaplanması gerekir. Bu enerji de varyanstır. Varyans hesaplandıktan sonra

varyansın karekökünden sapma da hesaplanabilir. Varyansın hesaplanması (3.4)de açıklanmıştır.

$$VAR[X] = E\left[\left(X - E[X]\right)^2\right] \tag{3.4}$$

Bu denklem denklem (3.5)deki gibi de açılabilir.

$$\begin{aligned} VAR[X] &= E\left[X^2 - 2E[X]X + E[X]^2\right] \\ &= E\left[X^2\right] - 2E\left[X\right]E\left[X\right] + E[X]^2 \\ &= E\left[X^2\right] - E[X]^2 \end{aligned} \tag{3.5}$$

Ayrıca varyansın rassal değişkenin olası değerleri cinsinden hesaplanması gerektiğinde (3.6) kullanılabilir.

$$\begin{aligned} VAR[X] &= E\left[X^2\right] - E\left[X\right]^2 \\ &= \sum\left[x_i^2 p\left(x_i\right)\right] - \left[\sum\left[x_i p\left(x_i\right)\right]\right]^2 \end{aligned} \tag{3.6}$$

Burada rassal değişken X bir dizinin herhangi bir elemanı, x_i'de bu elemanın alabileceği değerleri yani sinyali veya diziyi tanımlayan alfabe elemanları olarak düşünülebilir. Elemanların olasılıkları da elemanın dizi içindeki sayısını dizi içindeki toplam eleman sayısına bölünerek bulunur. Dizinin her bir elemanı $\boldsymbol{a_i}$ olarak adlandırılır ve dizinin N elemanı var kabul edilirse varyans bu değerlerden (3.7) deki gibi hesaplanabilir.

$$VAR[X] \approx \frac{\sum_{i=0}^{N-1} a_i^2}{N} - \left(\frac{\sum_{i=0}^{N-1} a_i}{N}\right)^2 \tag{3.7}$$

Sinyal için hesaplanan varyans değeri aynı entropide olduğu gibi sıkıştırma algoritmaları ile düşürülebilir. Fakat varyans tanımından da anlaşılacağı üzere entropi gibi alfabenin çeşitliliğini değil, değerlerin ortalama değerden ne kadar farklılık gösterdiği bilgisini vermektedir. Buna bağlı olarak varyanstaki değişim kayıpsız sıkıştıma

algoritmalarından çok, kayıplı sıkıştırma algoritmalarını ilgilendirmektedir. Çünkü kayıplı sıkıştırma algoritmaları belli yuvarlama ve nicemleme yöntemleri sinyalin enerjisini düşürebilmekte buna bağlı olarak sapmayı da indirebilmektedir.

Varyans ve entropideki değişimler bu tezde de incelenmiştir. Buna bağlı olarak hem Mycielski algoritmasının hem de Bölüm 4'de anlatılan Mycielski78 algoritmalarının sıkıştırma sonuçları Bölüm 7'de grafik ve çizelgelerle birlikte verilmiştir.

3.3. LZ77 ve LZ78 Algoritmaları

LZ77 ve LZ78, kayıpsız sıkıştırma algoritmalarıdır. Shannon tipi sembol tabanlı entropi sıkıştırma algoritmalarından farklı olarak harfleri ayrı ayrı sıkıştırmazlar. Bunun yerine, harflerin birbirleriyle oluşturduğu birleşimlere ve bu birleşimlerin sinyal içindeki tekrarlarına bakarlar. Bu özelliklerinden dolayı blok entropi kodlayıcı olarak belirtilebilirler.

[2]deki tanıma göre LZ77 algoritması, çıkış noktası açısından Mycielski algoritmasıyla benzerdir. İki algoritmada da dizinin sonunda tekrarlanmış olan en uzun alt diziyi bulmak amaçlanır. Mycielski, Bölüm 2.2' de de belirtildiği gibi bu bilgiyi öngörü için kullanır. LZ77 ise bu bilgi sayesinde tekrar noktalarını sembol atayarak kodlar, böylece gereksiz bilgi atılarak sıkıştırma sağlanır.

LZ77 algoritmasının her yeni aşamasında dizinin o ana kadar kodlanan yerinden bir sonraki noktada başlayıp geçmişte tekrar eden en uzun alt dizi parçası bulunmaktadır. Her kodlama aşamasında kodlama bittikten sonra üç veri yollanmaktadır. Bu verilerin ilki kodlanan veride bulunan konumdan kaç geri gidildiğinde bu alt dizinin yakalandığıdır. İkincisi geri gidildikten sonra varılan noktadan itibaren kaç uzunluklu bir veri parçası alındığı bir diğer değişle tekrar eden dizinin uzunluğudur. Sonuncusu da incelenen andan bir sonraki değerdir. Bu değer tekrar eden diziye eklenecek değerdir Şekil 3.1'de algoritmanın ikili bir dizi üzerindeki uygulaması gösterilmektedir.

Dizi: aaabbabaabbabaaaaababababab

Tekrarlanan	Kod
1. yok	(0,0,a)
2. a	(1,1,a)
3. yok	(0,0,b)
4. b	(1,1,a)
5. ba	(2,2,a)
6. bbabaa	(6,6,a)
7. aab	(9,3,a)
8. baba	(10,4,b)

Kod:0-0-a-1-1-a-0-0-b-1-1-a-2-2-a-6-6-a-9-3-a-10-4-b

Şekil 3.1. LZ77'nin bir dizi üzerindeki işlem adımları

LZ78 algoritmasında dizinin ilk elemanı direk olarak kitaplığa atılır. Daha sonraki eleman eğer ilk elemandan farklıysa doğrudan, değilse ondan sonra gelen eleman eklenerek kitaplığa atılır. Bundan sonraki elemanlar kitaplıktaki bütün kelimelerle karşılaştırılır. Kitaplıkta olmayan bir dizi parçası haline gelene kadar sonraki elemanlar bu karşılaştırılan elemana eklenir. Kitaplıkta olmayan bir dizi parçası elde edildiğinde kitaplığa atılır. Bu işlem dizinin son elemanına varılana kadar devam ettirilir. En sonunda hepsi birbirinden farklı, fakat başlangıç dizisini oluşturmamıza yetecek alt dizilerden oluşan bir kitaplık elde ederiz.

LZ78 algoritması [3]de gösterildiği gibi LZ77'nin zaman iyileştirilmesi yapılmış bir türevidir. Bu algoritmada, geçmişteki tekrar bloklarının tamamını aramak yerine, sinyal içindeki alt dizilerden bir sözlük oluşturulur. Bu alt diziler birbirleriyle sinyal içinde kesişmezler ve sözlükteki sırayla yan yana eklendiklerinde sinyalin kendisini oluştururlar. Ayrıca bu alt dizilerin bir örnekten daha fazla elemana sahip olanları, kendilerinden önce sözlüğe eklenen alt dizilere yeni bir örnek eklenerek türetilirler. Algoritma Şekil 3.2'deki ikili (binary) seri üstünde gösterilmiştir.

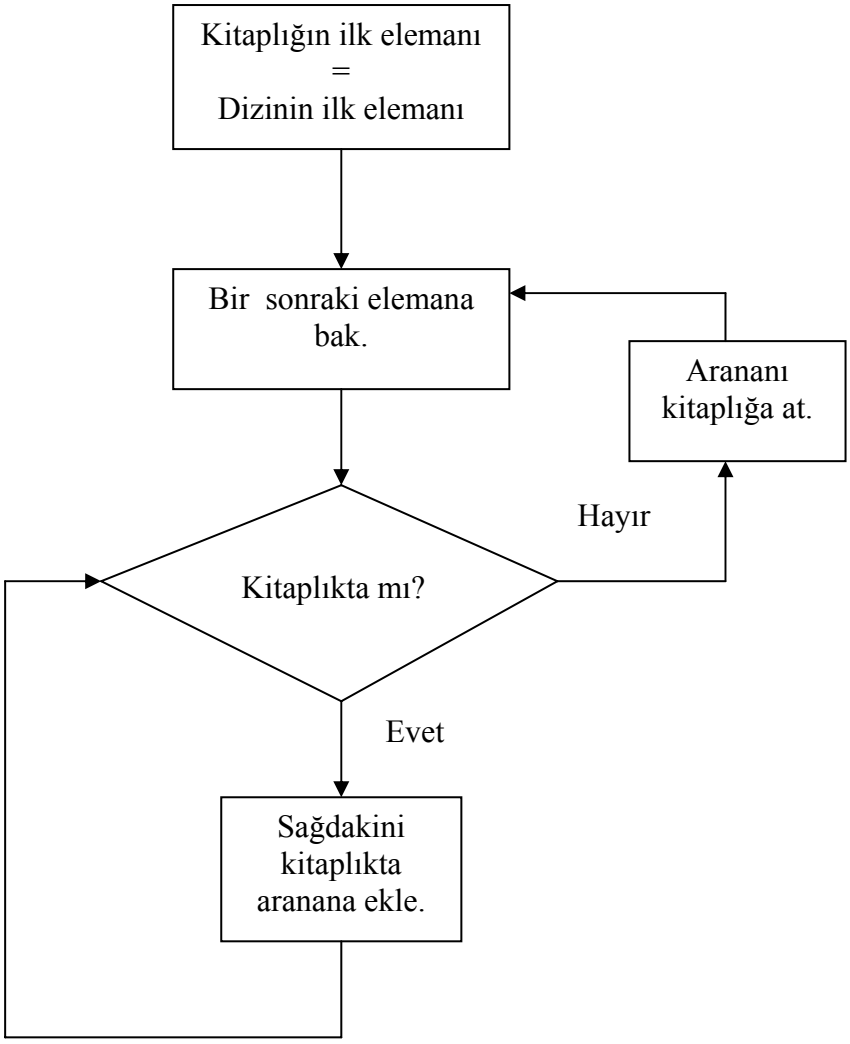
Dizi: aaabbabaabbabaaaaababababab

Sözlük	**Kod**
1. a	(0,a)
2. aa	(1,a)
3. b	(0,b)
4. ba	(3,a)
5. baa	(4,a)
6. bb	(3,b)
7. ab	(1,b)
8. aaa	(2,a)
9. aab	(2,b)
10. aba	(7,a)
11. bab	(4,b)

Kod:0a1a0b3a4a3b1b2a2b7a4b

Şekil 3.2. LZ78'in bir dizi üzerindeki işlem adımları

Şekil 3.3' de algoritmanın kitaplık oluşturma akış şeması ayrıntılı bir şekilde gösterilmektedir. Kod çözme sırasında, oluşturulan kod gelen veriyle eş zamanlı olarak okunup, kod çözücü sözlük, kodlayıcı sözlük ile senkronize bir sırayla oluşturulur. LZ78 algoritması, tüm geçmişi alt-dizileri ile birlikte taramadığı için teknik olarak LZ77'den daha az sıkıştırma elde eder. Ancak, oluşturduğu sözlük içinde tarama yapması sayesinde dizi uzaması ile polinom zamanlı şekilde işlem süresi artar ve LZ77'ye nazaran karmaşıklıkta azalma sağlanmış olur[3].



Şekil 3.3. LZ78 algoritması kitaplık oluşturma akış şeması

3.4. Öngörücü Veri Sıkıştırma

Bir verinin geçmiş örnekleri kullanılarak o veri sıkıştırma konusunda daha verimli sonuçlar elde edilebilecek bir biçime sokulabilir. Bu geçmiş bilgisi kullanılırken gelecek bilgisi hakkında da öngörüde bulunulur ve bu öngörü başlangıç gelecek değeri ile birlikte işlenerek bu yeni biçimdeki veri elde edilir. Bu tür yöntemlerle sıkıştırma yapan algoritmalar genel olarak öngörücü kodlayıcılar[12] olarak adlandırılır.

Elemanlarının olasılıkları birbiriyle simetrik olmayan bir alfabeyle kodlanan veri, simetrik bir alfabeyle kodlanan veriye göre daha verimli sıkıştırılabilir. Bir diğer değişle harflerinden bazılarının olasılığının diğerlerine göre daha büyük olması kodlamayı kolaylaştıran bir özelliktir. Bunun nedeni ise herhangi bir simgenin olasılığının büyük olması, bu simgenin eksi logaritmik fonksiyonun özelliği nedeniyle olasılıktaki büyüklük oranına göre daha büyük bir oranla taşıdığı bilginin küçülmesine neden olmasıdır. Bu da verinin eşit olasılıklı simgeler taşıyan denk bir verinin entropisine göre daha düşük bir entropiye sahip olmasını sağlamaktadır. Simetrik olasılıklı veriler, öngörücü algoritmaların sağladığı dönüşümler sayesinde asimetrik olasılıklı kolay sıkıştırılabilir verilere dönüştürülebilir. İşin doğası gereği, kodun tekrar çözülebilmesi için uygulanan dönüşümün kod çözücü tarafından da bilinmesi gerekir. Eğer yapılan öngörü, hem kodlayıcının hem de kod çözücünün sahip olduğu geçmişe dayalıysa, kod çözcüye ek bir bilgi gönderilmesine gerek yoktur.

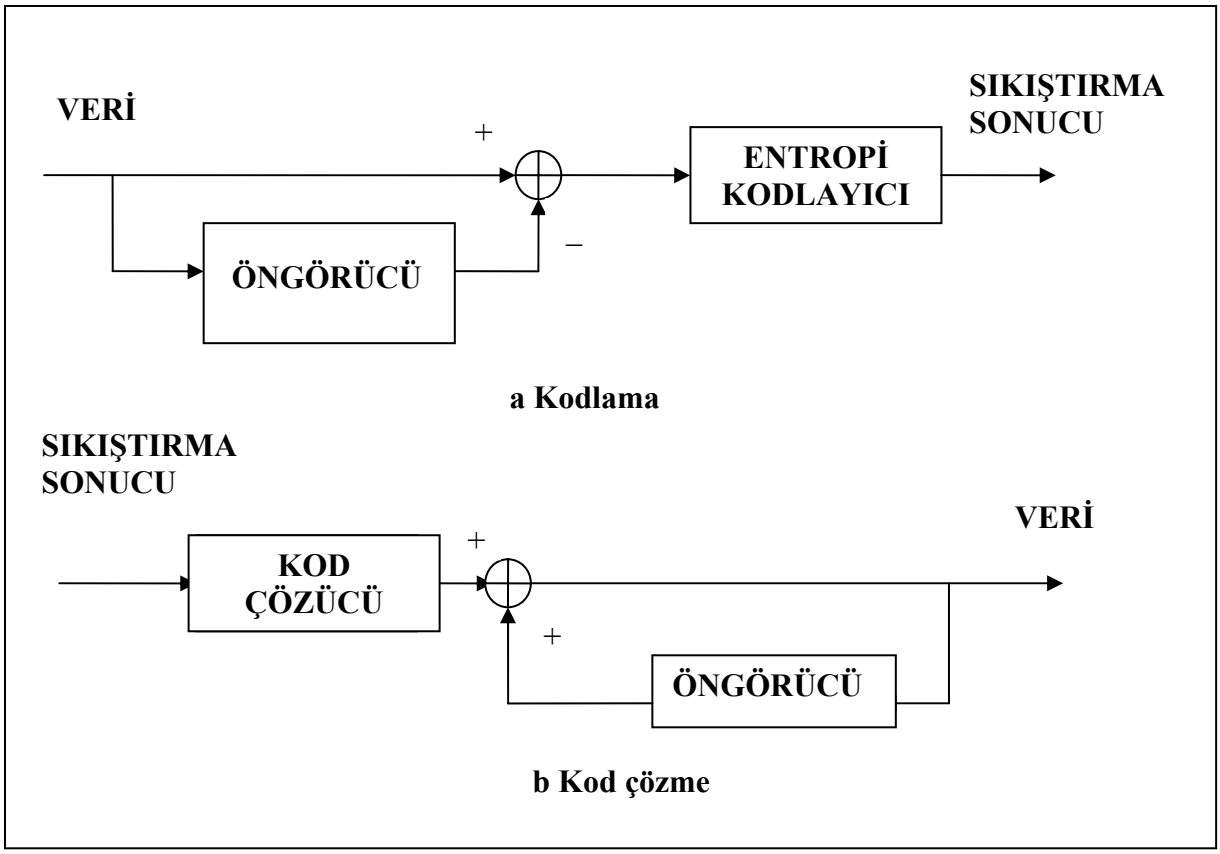
Öngörücü sıkıştırma algoritmaları kayıplı ve kayıpsız algoritmalar olarak iki ayrı türde de tasarlanabilir. Kayıplı sıkıştırma algoritması olarak tasarlananlarda ek olarak kodlama aşamasında nicemleyici kullanılır. Kod çözme aşamasında ise nicemleyiciye ihtiyaç yoktur. Kayıpsız öngörücü sıkıştırma algoritmalarının genel yapısı Şekil 3.4'de, kayıplı öngörücü sıkıştırma algoritmalarının yapısı ise Şekil 3.5'de gösterilmektedir.

Bu tür algoritmalarda kullanılan öngörücülere doğrusal öngörücü iyi bir örnektir. Bir sinyalin N'inci elemanı için doğrusal öngörücü (3.8)deki gibi öngörüde bulunur.

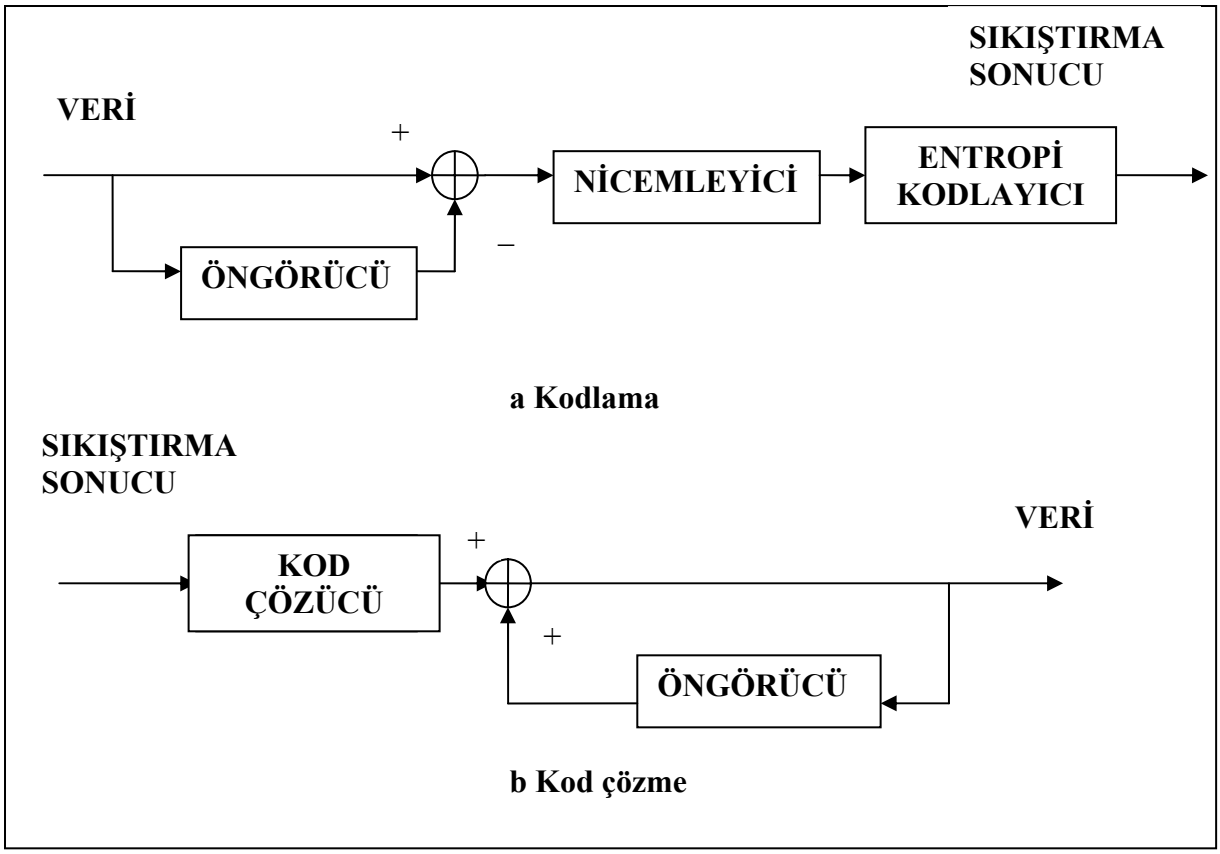
$$\hat{x}(n) = -\sum_{i=1}^{P} a_i x(n-i) \tag{3.8}$$

Bu denklemde "a_i"'ler tasarlanan doğrusal öngörücünün sabit katsayılarıdır. Ayrıca "P" öngörücünün derecesidir. Bunu değiştirmek öngörücüyü tasarlayanın kontrolündedir. Bu öngörü değer hesaplandığında karşı tarafa bu değerle gerçek değer arasındaki fark gönderilir. Bu farkı (3.9) daki gibi belirtebiliriz.

$$e(n) = x(n) - \hat{x}(n) \tag{3.9}$$



Şekil.3.4. Kayıpsız öngörücü sıkıştırma yöntemi



Şekil.3.5. Kayıplı öngörücü sıkıştırma yöntemi

4. MYCIELSKI78 ALGORİTMASI

Mycielski algoritmasında, kullanılan geçmişin boyutu sonsuza yaklaştığında, uygulama süresi olabildiğince uzamakta ve uygulama imkânsızlaşmaktadır. İncelenen dizinin boyutundaki artışa bağlı olarak bu süre polinom harakete göre çok daha ivmeli bir artış göstermektedir. LZ77 sıkıştırma algoritmasında da benzer bir karmaşıklık sorunu vardır ve bu karmaşıklık LZ78 algoritmasında, geçmişi tanımlamak için kullanılan bir sözlük yapısıyla giderilmiştir. LZ78'den ilham alınarak yeni bir öngörü algoritması geliştirilmiş ve geliştirilen algoritma Mycielski78 algoritması olarak adlandırılmıştır. Bu yeni algoritmada, uygulama süresinin polinom zamana indirilmesi sağlanmıştır. Mycielski algoritmasındaki tarama tekniği LZ77 sıkıştırma algoritması ile aynı olduğu gibi, Mycielski78 algoritmasındaki tarama tekniği de LZ78 sıkıştırma algoritması ile aynıdır. Bir diğer değişle Mycielski78'de öngörü için tarama sınırsız geçmiş üzerinden değil, bu geçmiş kullanılarak oluşturulmuş sözlük üzerinden yapılır. Bu sayede aranan alt dizi, sınırsız geçmiş üzerinden tek örnek kayarak görülebilecek kendi uzunluğundaki bütün birleşimler yerine, sadece sözlükte kendisiyle aynı uzunlukta olan alt dizilerle karşılaştırılır. Mycielski ile Mycielski78 arasında LZ77 ile LZ78 arasındakine benzer bir tercih ilişkisi vardır. Mycielski78 daha hızlı çalışmakla beraber Mycielski kadar güçlü öngörüler yapamamaktadır. Fakat uzun seriler için öngörüdeki kayıp zamandaki kazanca göre tercih edilebilir bir seviyede kalmaktadır. Önerilen Mycielski78 algoritmasının örnek bir ikili seri üzerinde çalışması Şekil 4.1'deki uygulamada gösterilmiştir.

Dizi: 1011011010110110

Dizi	Tekrar	Öngörü	Fark	Sözlük Girişi
1. 1	yok	0	1-0=1	-
2. 0	yok	0	0-0=0	-
3. 1	yok	0	1-0=1	-
4. 1	1.(1)	2.[1](0)	1-0=1	(1)1
5. 0	1.(1)	2.[1](0)	0-0=0	-
6. 1	2.(0)	3.[1](1)	1-1=0	(2)0
7. 1	1.(1)	2.[1](0)	1-0=1	-
8. 0	3.(11)	4.[1](0)	0-0=0	(3)11
9. 1	2.(0)	3.[1](1)	1-1=0	-
10. 0	4.(01)	5.[1](1)	0-1=-1	(4)01
11. 1	5.(10)	6.[1](1)	1-1=0	(5)10
12. 1	4.(01)	5.[1](1)	1-1=0	-
13. 0	3.(11)	4.[1](0)	0-0=0	-
14. 1	5.(10)	6.[1](1)	1-1=0	-
15. 1	6.(101)	7.[1](1)	1-1=0	(6)101
16. 0	3.(11)	4.[1](0)	0-0=0	-

Dizi: 101101101 0110110
Öngörü: 000001001 1110110
Fark: 101100100-1000000

Şekil 4.1. Mycielski78 algoritmasının öngörü aşamaları

Bu algoritmada üretilen fark sinyali başlangıç sinyalinin tekrar oluşturulması için gerekli veriye sahiptir. Fark sinyalinin ilk iki değeri başlangıç sinyalinin ilk iki değeriyle aynı büyüklüklere sahiptir. Bir diğer değişle, fark sinyali kullanılarak üçüncü örnek için yapılan öngörü değerine tekrar ulaşılabilmektedir. Elde edilen öngörü değeri ile fark sinyalinin üçüncü örneği toplandığında başlangıç sinyalinin üçüncü değeri elde edilmektedir. Bu yöntem iteratif bir şekilde uygulandığında adım adım başlangıç sinyali oluşturulmakta ve bir sonraki adım için o ana kadar oluşturulmuş başlangıç sinyali kullanılmaktadır. Bu aşamada öngörü için oluşturulan sözlüğün güncellenebilir olması gerekmektedir. Bu algoritmada da sözlük, dizinin örnekleri okundukça güncellenmekte, öngörü de o ana kadar güncellenmiş sözlük

üzerinden yapılmaktadır. Yukarıdaki örnekten de yola çıkacak olursak, sinyalin dokuzuncu örneğine kadar sözlüğün ilk dört elemanı oluşturulmuş ve dokuzuncu örnek değerinin öngörüsü için sözlüğün sadece bu dört elemanı kullanılmıştır. Bu şekilde, LZ77 veya Mycielski gibi doğrudan verinin sonsuz geçmişini her seferinde baştan tarayan bir öngörücü yerine, LZ78 benzeri bir şekilde sözlük üreterek bu sözlük üzerinden arama yapan ve burada adına Mycielski78 dediğimiz algoritma geliştirilmiştir.

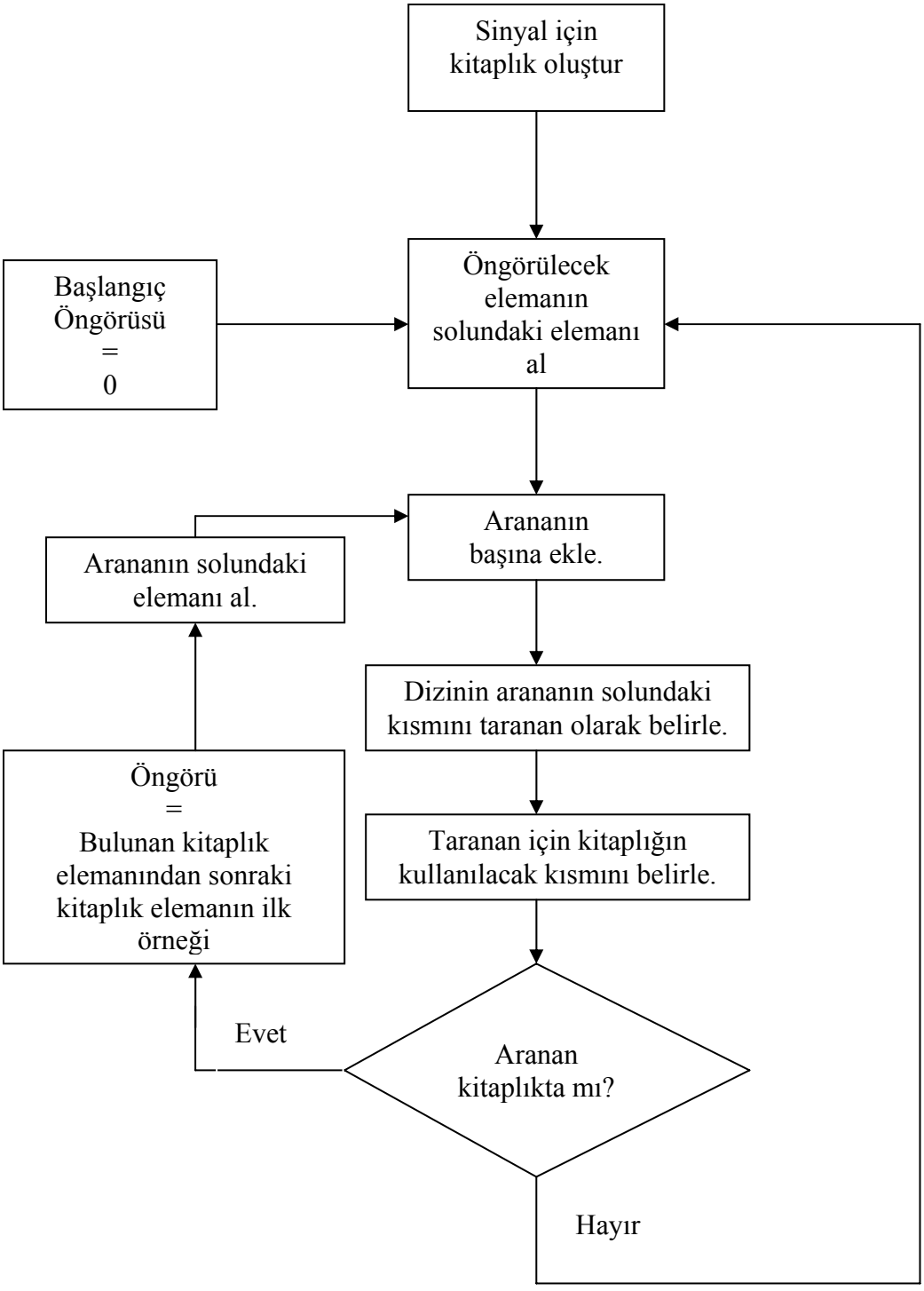
Mycielski algoritmasında karşılaştırmalarda Hamming uzaklığı kullanılmaktadır. Bu uzaklıkta karşılaştırılan örnek değerleri birbirine eşitse aradaki uzaklık 0, değilse 1 alınmaktadır. Dolayısıyla karşılaştırılan örneklerin değerleri birbirine çok yakın olsa dahi aynı olmadıkları sürece, örnekler birbirinden farklı kabul edilmektedir. Hamming uzaklığı (4.1)'de belirtilmektedir.

$$d(x_1, x_2) = \begin{cases} 0, & x_1 = x_2 \\ 1, & x_1 \neq x_2 \end{cases} \tag{4.1}$$

Sürekli zaman sinyallerin örneklenmesi ile elde edilen tip sinyallerde de değerler ikili olmadığı için ve bir büyüklüğün ikinci bir tekrarının görülme olasılığı ikili sinyallere göre daha düşük olmakta, dolayısıyla yapılan öngörü ikili sinyaller için yapılan öngörüye göre daha güçsüz olmaktadır. Bu durumu engellemek için bu çalışmada kullanılan metrik, Hamming uzaklığının toleranslı bir türevi olarak şu şekilde tanımlanmıştır:

$$d(x_1, x_2) = \begin{cases} 0, & |x_1 - x_2| < \dfrac{\max\left(|x_1|, |x_2|\right)}{10} \\ 1, & |x_1 - x_2| \geq \dfrac{\max\left(|x_1|, |x_2|\right)}{10} \end{cases} \tag{4.2}$$

Bu yeni uzaklık hem Mycielski hem de Mycielski78 algoritmasında kullanılmış ve sonuçlar Bölüm 5'te sunulmuştur. Ayrıca Mycielski78 algoritmasının akış şeması Şekil 4.2'de gösterilmektedir.



Şekil 4.2. Mycielski78 algoritması akış şeması

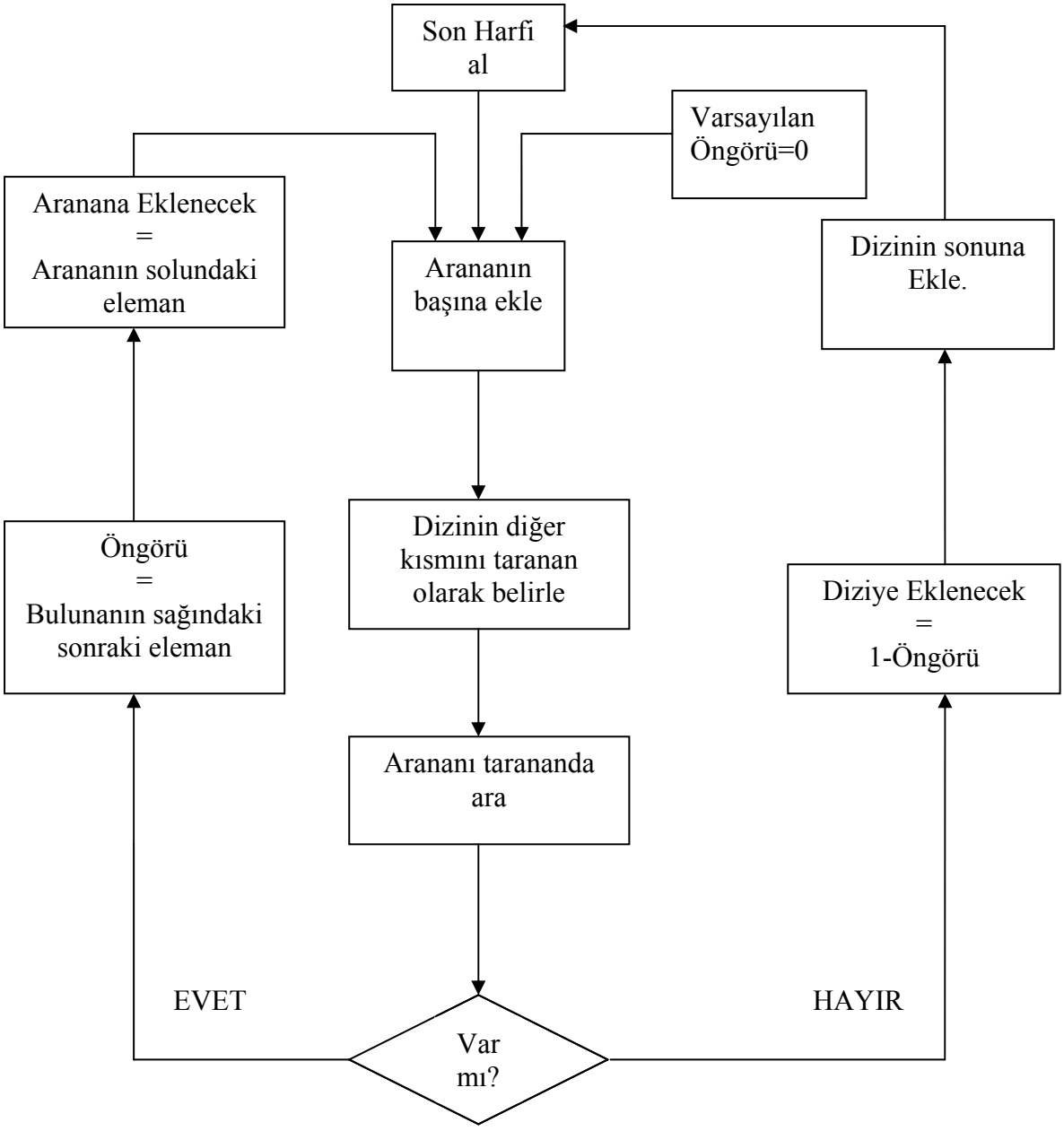
5. ANTİMYCIELSKI SÖZDE RASSAL SAYI ÜRETECİ

Antimycielski [6] sözde rassal sayı üreteci bit üretimi için Mycielski öngörü algoritmasını kullanmaktadır. Sınırsız geçmiş(öngörüden önceki tüm örnekler) kullanılarak elde edilen ikili düzendeki öngörünün tersinden yeni bir bit üretim yöntemi geliştirilmiştir. Esasen elde edilen öngörü zaman serileri analizi için veya sıkıştırma amaçlı kullanılmaktadır. Fakat başlangıç dizisine Mycielski yöntemiyle elde edilen ardışık öngörülerin tersi eklendiğinde, dizi öngörü edilemez bir yapıya dönüşmektedir. Bir diğer değişle, algoritma ters öngörü fikrine dayanmaktadır.

Zıt öngörü üretme sürecinin çekirdek yapısını oluşturan Mycielski algoritması Bölüm 2'de ayrıntılı bir şekilde tanıtılmıştır. Bu sayı üretecinde N+1' inci eleman için yapılan öngörünün bir tamlayanı alınmakta ve bu eleman yerine bu tamlayan değer konulmaktadır. Bu üretecin analitik ifadesi aşağıdaki gibi gösterilebilir:

$$
\begin{aligned}
m &= \arg\max_{L} \begin{cases} x[k] = x[N], x[k-1] = x[N-1] \\ ,\ldots, x[k-L+1] = x[N-L+1] \end{cases}, \\
\hat{x}_{N+1} &= x[k+1], \\
f_{N+1} &= 1 - \hat{x}_{N+1}.
\end{aligned}
\tag{5.1}
$$

Bu algoritmada her sonraki eleman için öngörü yapılıp tamlayanı diziye eklendikten sonra öngörü işlemi bir sonraki değer için tekrar başlatılır. Bu güncelleme ve öngörü adımları şifrelenecek dosya boyutunda rassal dizi elde edilene kadar devam ettirilir. Şekil 5.1'de algoritmanın akış şeması gösterilmiş ve Şekil 5.2'de de örnek üzerinde adım adım uygulaması yapılmıştır. Gösterilen örnekte başlangıç olarak 2 bitlik "11" dizisi alınmıştır.



Şekil 5.1. Antimycielski akış şeması

Başlangıç Dizisi:"11**"**

Üretilecek dizinin uzunluğu: 2 bayt

Dizi	**Tekrar**	**Eklenecek Değer**
1*1*	1	(1)'=0
110	yok	(0)'=1
1**1**0*1*	1	(0)'=1
110*11*	11	(0)'=1
1101*11*	11	(0)'=1
110**11***11*	11	(1)'=0
11011*110*	110	(1)'=0
1101111**0***0*	0	(0)'=1
11**01**1110*01*	01	(1)'=0
110111**10**0*10*	10	(0)'=1
1**101**11100*101*	101	(1)'=0
1**1011**1100*1011*	1011	(1)'=0
11011**110**010*110*	110	(0)'=1
11011110010*1101*	1101	(1)'=0
1101111001011010		

Üretilen Rassal Dizi: 01111001011010

Şekil 5.2. Antimycielski algoritmasının sayı üretme aşamaları

Bu yöntemle rassal dizi üretmenin en olumsuz yanı, dizideki bit sayısının arttığı her aşmadan sonra arama sürecinin daha karmaşıklaşmasıdır. Bu çalışmanın önemli yararlarından biri, bu karmaşıklığın sınırsız geçmiş yerine sınırlandırılmış geçmiş kullanımıyla giderilmesidir. Bu sınırlandırma yapılırken şifreleme için gerekli rassalığın kaybedilmemesi de sağlanmıştır. Geçmişin belirli bir sınırla sınırlandırılmasıyla, arama için harcanan süre makul düzeylere inmiştir.

Geçmiş uzunluğunda kısıtlama için uygulanabilecek iki yöntem vardır. Birinci yöntemde geçmiş her sınır uzunluğa ulaştığında anahtar olarak kullanılacak son kısmı (kuyruk) dışında kalan kısım tamamen temizlenir. Temizlemeden sonra bir sonraki sınıra ulaşma anına kadar üretilen zıt öngörüler geçmişe eklenir. Bu yöntemde, oluşturulan rassal dizi bir takım daha küçük rassal dizilerden oluşan bir zincir gibi düşünülebilir. Bu zincirin ilk halkası girilen

anahtar kelimeden, diğer halkalar ise bir önceki halkanın kuyruk kısmından üretilir. Bir diğer kısıtlama yöntemi ise geçmiş taramasında kayan pencere yapısının kullanılmasıdır. Bu çalışmada ilk yöntem seçilmiş ve farklı uzunluklardaki anahtar sözcük ve sınırlı geçmişlere yönelik sonuçlar Bölüm 8'de sunulmuştur.

6. RASSALLIK TESTLERİ

Bir dizinin rassallığını ölçmek bir takım genel yöntemler bulunmaktadır. Bu yöntemlerin çoğunda diziye bir grup rassallık testi uygulanmakta ve başarı kıstası olarak da geçtiği test sayısına bakmaktadır. Bir şifre gücünü rassallığı ile doğru orantılı göstermektedir. Bu çalışmada, rassallık analizi için NIST test paketi kullanılmıştır. NIST test paketinin içerdiği testler aşağıdaki gibi sıralanmıştır:

6.1. Yaklaşık entropi testi
6.2. Blok içinde frekans testi
6.3. Birikmiş toplamlar testi
6.4. Kesikli Fourier dönüşümü testi
6.5. Frekans testi
6.6. Blok içinde birlerin en uzun tekrarı testi
6.7. Kesişmeyen şablonların eşlemesi testi
6.8. Kesişen şablonların eşlemesi testi
6.9. Rassal gezinim testi
6.10. Rassal gezinim değişkesi testi
6.11. İkili matris kertesi testi
6.12. Tekrar testi
6.13. Seri test
6.14. Maurer'in "Evrensel İstatistiksel" testi

Bu testlerin amaçları aşağıda belirtilmiştir. Bu açıklamalara göre testler önceden beklenen eşik parametrelerini hesaplamaktadır. Rassal dizinin uzunluğuna göre değişen bu parametreler analiz sonuçlarıyla karşılaştırılmaktadır.

6.1. Yaklaşık Entropi Testi

Bu test dizi içinde olan m ve m+1 uzunluğundaki kesişen yapıların sayılarını hesaplar. Rassallık başarısı için hesaplanan bu iki sayının birbirinden ıraksaması gerekmektedir.

6.2. Blok İçinde Frekans Testi

Bu test dizinin bölünmüş blokları içindeki sıfırların ve birlerin sayılarına bakar. Eğer blokların çoğunluğu yaklaşık eşit sayıda birlere ve sıfırlara sahip ise dizi testten geçer.

6.3. Birikmiş Toplamlar Testi

Birikmiş toplamlar testi, sıfırların büyüklüğünü eksi bir varsayarak dizinin ve eş uzunluklardaki alt dizilerin birikmiş toplamlarını hesaplamaktadır. Rassallığa bağlı olarak, alt dizilerin birikmiş toplamları dizinin birikmiş toplamının etrafında salınım yapması ve diziyle alt dizilerin birikmiş toplamları farklarının ortalama da sıfıra yaklaşması gerekmektedir.

6.4. Kesikli Fourier Dönüşümü Testi

Bu test dizinin hızlı Fourier dönüşümündeki tepeciklerin yüksekliklerini sınamaktadır. Bu testin amacı dizide rassallığı engelleyici herhangi bir baskın harmoninin olup olmadığını sınamaktır.

6.5. Frekans Testi

Bu test tüm dizi içindeki sıfırların ve birlerin sayılarını hesaplamaktadır. Rassallık eş sayıda bir ve sıfır gerektirmektedir.

6.6. Blok İçinde Birlerin En Uzun Tekrarı Testi

Bu testte birlerin en uzun tekrarı incelenmekte ve bu tekrarların beklenen sınırlar içinde olup olmadığı sınanmaktadır.

6.7. Kesişmeyen Şablonların Eşlemesi Testi

Birbirinden farklı olup birbiriyle kesişmeyen m uzunluğundaki alt dizilerin sayısını hesaplamaktadır. Bu sayı rassallık için en büyük olması gerekir.

6.8. Kesişen Şablonların Eşlemesi Testi

Bu test 6.7'ye benzer olarak farklı alt dizileri kontrol etmektedir. 6.7'den farklı olarak burada alt diziler birbiriyle kesişebilmektedir.

6.9. Rassal Gezinim Testi

Bu test rassal gezinimler içindeki durum yürüyüşlerinin sayısının beklenen değeri aşıp aşmadığını kontrol eder. Eğer beklenilen değer aşılıyorsa, dizinin rassal olmadığı kabul edilir.

6.10. Rassal Gezinim Değişkesi Testi

Bu testin amacı farklı durum sayılarının beklenen değerden sapma gösterip göstermediğini anlamaktır.

6.11. İkili Matris Kertesi Testi

Bu test diziden oluşturulan alt matrislerin kertelerini karşılaştırmaktadır. Testin esas amacı eş uzunluklardaki alt diziler arasında doğrusal bağımlılığın olup olmadığını kontrol etmektir.

6.12. Tekrar Testi

Bu test dizi içindeki birlerin ve sıfırların tekrarlarını hesaplamaktadır. Rassallık için birlerin tekrar uzunluğunun sıfırların tekrar uzunluğuna yakın olması ve bu uzunlukların belirli sınırlar içerisinde olması gerekmektedir.

6.13. Seri Test

Bu test m ve 2m boyutlu tekrar etmeyen alt dizilerin sayılarını hesaplamaktadır. Rassallık için m boyutlu dizilerin sayısının, 2m boyutlu dizilerin sayısına eşit olması gerekir.

6.14. Maurer'in "Evrensel İsatatistiksel" Testi

Bu test dizinin kayıpsız sıkıştırılma oranını hesaplamaktadır. Eğer büyük bir sıkıştırma oranı elde edilmişse, dizinin rassal olmadığı kabul edilir.

7. SIKIŞTIRMA SONUÇLARI

Mycielski algoritması ve Mycielski78 algoritmaları Hamming uzaklığı yerine denklem 4.2'deki uzaklık kullanılarak değiştirildi. Elde edilen algoritmalar piksel değerleri 8 bitle tanımlanan 100x100 boyutlarında gri tonlu Lena resminde ve örnek değerleri 16 bitle tanımlanmış 500 örneklik bir wav ses dosyasında denendi. Bu denemeler 1.6 GHz'lik Pentium 4 işlemcili, 256 MB rassal erişilebilir belleği olan bir bilgisayarda, Matlab programının 6.5 sürümü kullanılarak yapıldı. Lena resminde sütunların her biri birbirinden bağımsız diziler olarak düşünüldü ve öngörü ve fark resimleri buna göre elde edildi. Lena resimlerindeki öngörü ve fark sonuçlarını Şekil 7.1'de görebilirsiniz. Resim için hesaplanan değerler aşağıdaki gibidir. Resimlerle de görüleceği üzere üretilen resimlerin ilk iki satırı siyah olarak belirmiştir. Bunun nedeni her sütunun ilk iki değeri için karşılaştırma yapılamadığından öngörü değeri sıfır atanmıştır. Bunun dışında Mycielski yöntemiyle elde edilen öngörü resminin başlangıç resmine Mycielski78'in ürettiğinden daha yakın olduğu görülür ki, bu da beklenen bir sonuçtur. Bu sonuç aşağıda belirtildiği gibi resimler arasındaki fark resimlerinin enerjisinden de anlaşılabilmektedir.

Orijinal Resim (100x100 Siyah-Beyaz Lena):

Enerjisi: 176892773

Mycielski Öngörü Resmi:

Bu resim ile orijinal resmin farkının enerjisi = 21128055
Fark enerjisinin orijinal resim enerjisine oranı = %11.94
İşlem Süresi = 45-50 Saniye

Mycielski78 Öngörü Resmi:

Bu resim ile orijinal resmin farkının enerjisi = 31021163
Fark enerjisinin orijinal resim enerjisine oranı = %17.53
İşlem Süresi = 45-50 Saniye

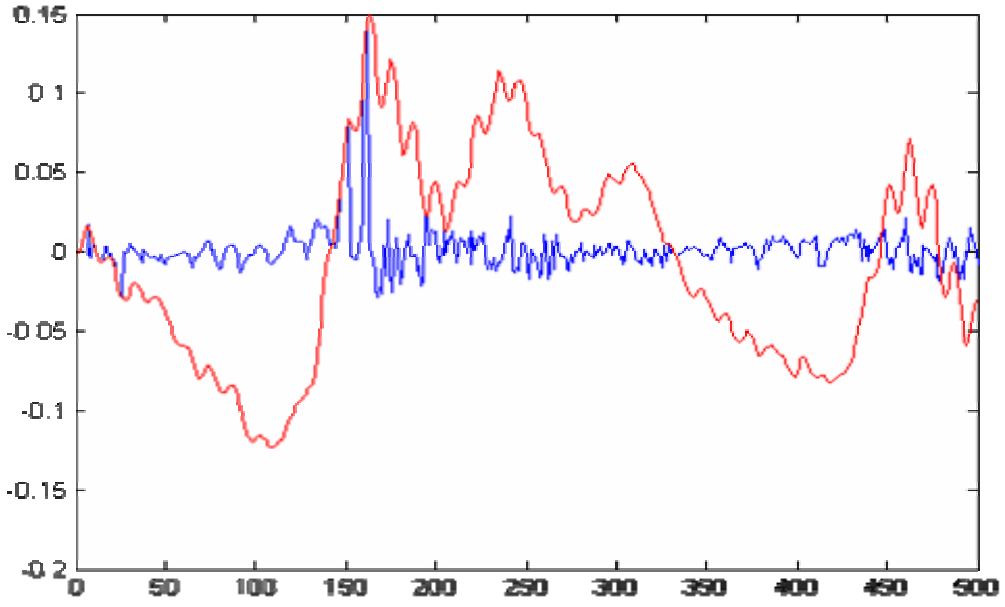
Şekil 7.2 'de Mycielski algoritmasının ses için ürettiği öngörü ve fark sinyalleri gösterilmiştir. Şekil 7.3'de de Mycielski78 algoritmasının öngörü ve fark sinyalleri gösterilmiştir. Bu grafiklerin hepsinde kırmızı olan başlangıç ses sinyali mavi olan da öngörü veya fark sinyalidir.

a Orijinal resim

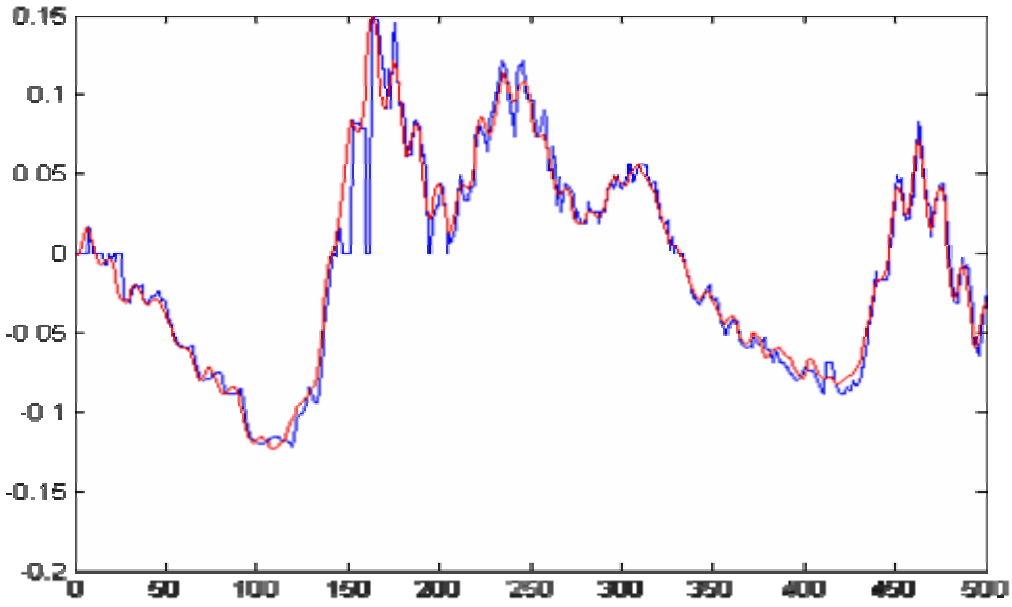
b Mycielski öngörü resmi

c Mycielski78 öngörü resmi

Şekil 7.1. Mycielski ve Mycielski78 algoritmalarının öngörü sonuçları

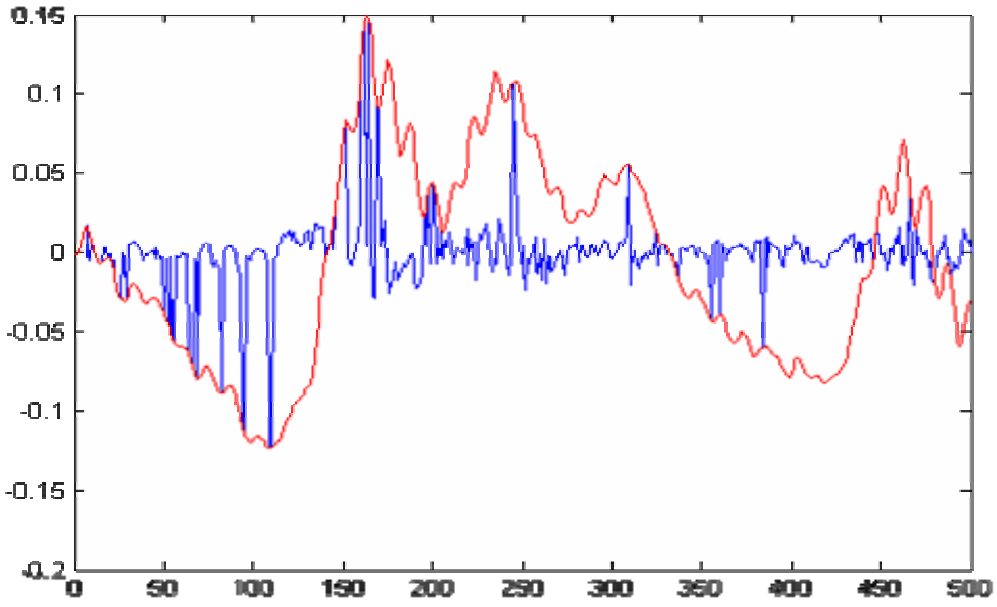


a Orijinal sinyal ve fark sinyali

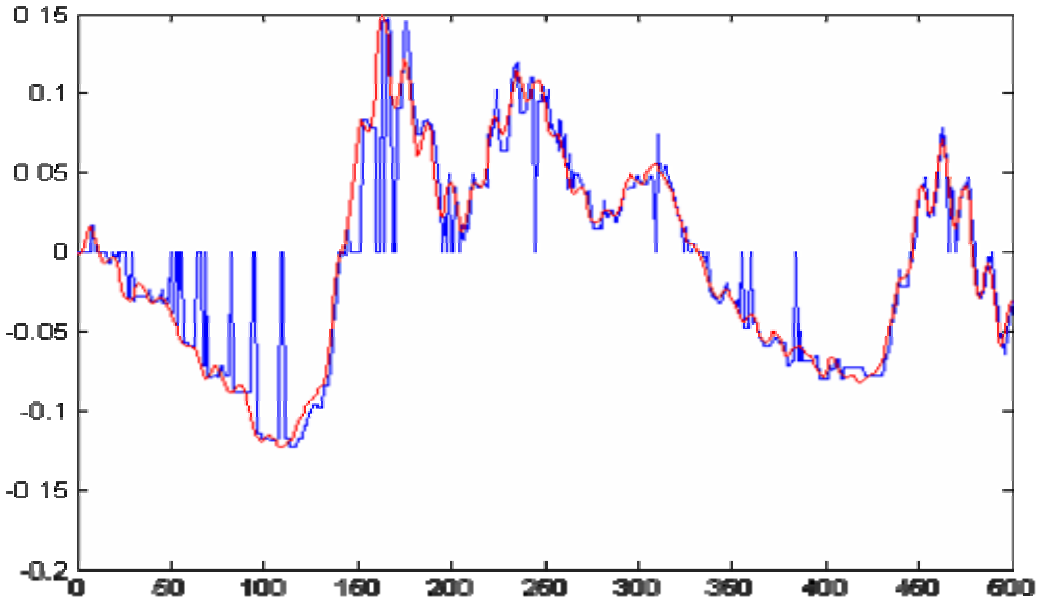


b Orijinal sinyal ve öngörü sinyali

Şekil 7.2. Mycielski algoritmasının bir ses sinyali için öngörü ve fark sinyali sonuçları



a Orijinal sinyal ve fark sinyali



b Orijinal sinyal ve öngörü sinyali

Şekil 7.3. Mycielski78 algoritmasının bir ses sinyali için öngörü ve fark sinyali sonuçları

Ses sinyali ile ilgili enerji sonuçları aşağıda verilmiştir. Bu değerlere göre de Mycielski, Mycielski78'den daha iyi öngörü yapmış fakat Mycielski78 zamanda Mycielski'ye göre çok büyük bir kazanç sağlamıştır. Lena resminde bu kazancın görülememesinin nedeni resmin sütunlarının uzun olmaması ve bu sütunların birbirinden bağımsız değerlendirilmesidir.

Orijinal Sinyal Enerjisi=2.0804

Mycielski Algoritması:

Fark Sinyali Enerjisi=0.1145

Fark Sinyal Enerjisinin Orijinal Sinyale Oranı= %5.50

İşlem Süresi=4 dakika 35 Saniye

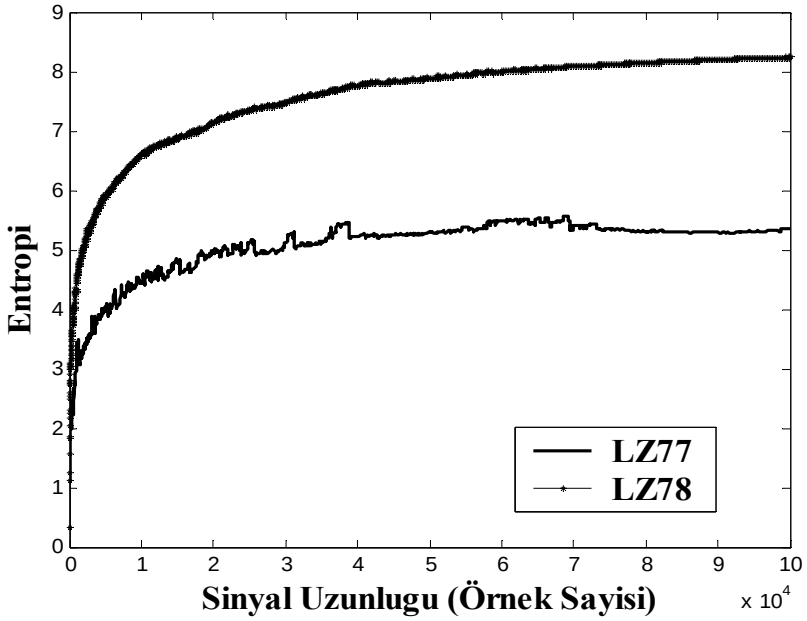
Mycielski78 Algoritması:

Fark Sinyali Enerjisi= 0.2784

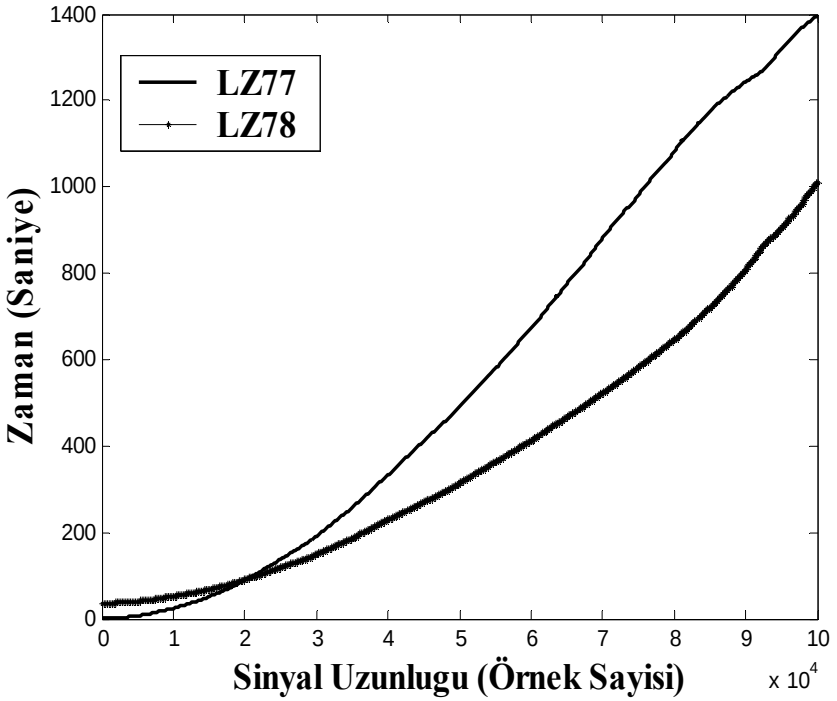
Fark Sinyal Enerjisinin Orijinal Sinyale Oranı= %13.38

İşlem Süresi= 27 Saniye

Literatürdeki LZ77 ile LZ78 arasındaki *"çalışma hızı / sıkıştırma miktarı"* tipi bir ilinti, bu tezde sunulan Mycielski78 ve başlangıç Mycielski öngörü algoritmaları arasında da gözlenmiştir. Bu çalışmada yukarıdaki sonuçlara ek olarak 22000Hz frekansla ve her örnek için 8 bit kullanılarak örneklenmiş wav formatında bir ses dosyasının ilk 100000 örneği kullanılmıştır. Bu sınamalar da bir önceki testte kullanılan bilgisayar ortamı seçilmiştir. Programların sıkıştırma performansları için entropi ve enerji kavramlarının kullanılması düşünülmüş, fakat enerji niceleme gerektiren kayıplı algoritmalar için daha uygun olduğundan ve buna karşılık entropi hem kayıpsız hem kayıplı algoritmaların performansında kullanılabileceği için performans ölçütü olarak "entropi" seçilmiştir. Mycielski78 için entropi elde edilen fark sinyalinin entropisidir. Bu dosyanın LZ77 ve LZ78 kullanarak sıkıştırma performanslarının örnek sayısı arttıkça nasıl değiştiği Şekil 7.4'de gösterilmektedir. Örnek sayısı cinsinden işlem zamanları ise Şekil 7.5'de verilmektedir.



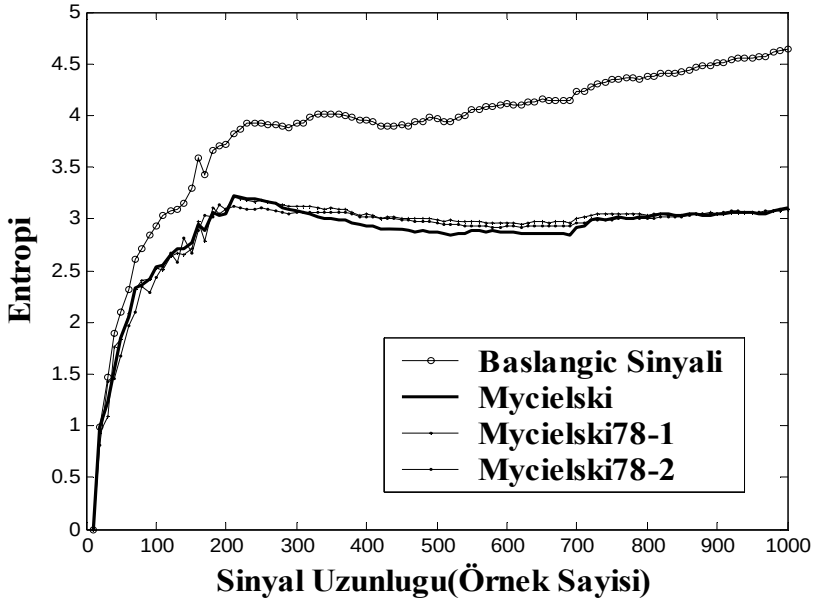
Şekil 7.4. LZ77 ve LZ78'in entropi performansları



Şekil 7.5. LZ77 ve LZ78'in zaman performansları

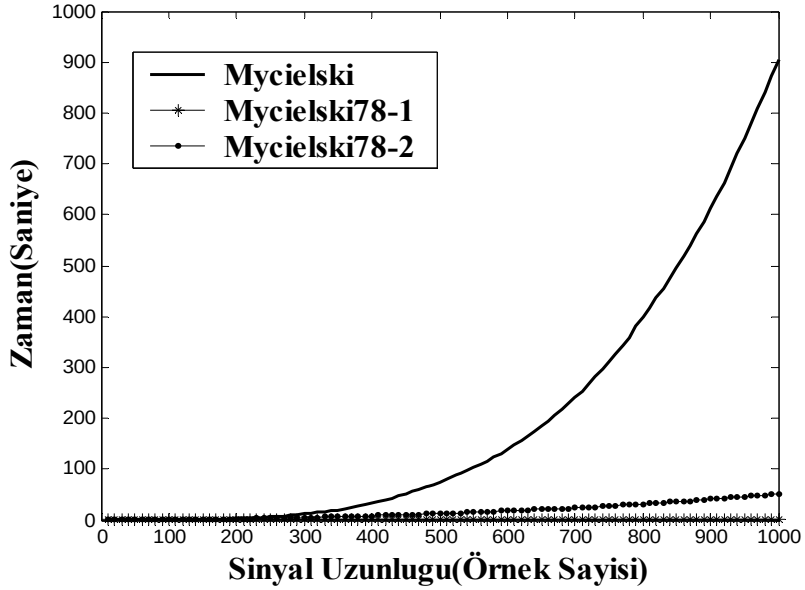
Şekil 7.6 ve 7.7'de ise Mycielski ve Mycielski78 algoritmalarının performansları aynı ses dosyasının ilk 1000 örneği üzerinden karşılaştırılmıştır. Bu üç grafikte Mycielski78–1 Hamming uzaklığı kullanılarak oluşturulmuş Mycielski78 algoritmasını, Mycielski78–2 ise değiştirilmiş Hamming uzaklığı kullanılarak oluşturulmuş Mycielski78 algoritmasını içermektedir.

Şekil 7.6'daki değerlerle entropi performansları incelenecek olursa başlarda Mycieski78–2, ortalarda da Mycielski algoritması daha iyi performans göstermiştir. Fakat sinyal uzunluğu arttıkça her üç fark sinyalinin entropi değerleri aynı değere yakınsamaktadır.



Şekil 7.6. Mycielski ve Mycielski78'in entropi performansları

Şekil 7.7'deki sonuçlara dayanarak bu üç algoritma içinden Mycielski en yavaş çalışan program olarak görülmekte ve harcadığı zaman örnek sayısındaki artışa bağlı olarak hızlı bir şekilde artmaktadır. En hızlı çalışan Mycielski78–1 algoritmasıdır ve harcadığı zaman ile Mycielski algoritmasının harcadığı zaman arasındaki fark örnek sayısı arttıkça ıraksamaktadır.



Şekil 7.7. Mycielski ve Mycielski78'in zaman performansları

Bu sonuçlar Mycielski78 algoritmasının başlangıç sinyalinin entropisini düşürebildiğini göstermektedir. Mycielski algoritmasına nazaran sağlanan zaman kazancı yanında sinyal boyutu arttıkça Mycielski ile birebir öngörülerde bulunması, oluşturulacak yeni sıkıştırma algoritmasının kullanışlılığı açısından Mycielski yerine Mycielski78 algoritmasının kullanılmasının daha mantıklı ve pratik olduğunu göstermektedir.

Çizelge 7.1. Algoritmaların sıkıştırma yüzdeleri
(100x Entropi(Üretilen) / Entropi(Baslangıç))

	.exe	**.wav**	**Resim**	**.txt**	**.xml**	**Genel**
LZ77	72	85	77	103	51	78
LZ78	113	121	103	158	127	124
MYC78	107	69	77	102	42	79

Çizelge 7.1'de Mycielski78 algoritmasının üretilen sinyal ile baslangıç sinyali entropi oranı cinsinden farklı türde dosyalar için sıkıştırma yüzdeleri verilmiştir. Her tür için ortalama 20 KB büyüklüğünde 5 farklı dosya seçilmiş, genel yüzde bu beş dosyadaki performansların ortalamasından elde edilmiştir. Bu dosyalar için *Large Conterbury Corpus'*undan ve *Maptask 2001 Corpus'*undan yararlanılmıştır. Tablodaki verilere göre LZ78 dosyaları entropi olarak sıkıştıramamıştır. Fakat yaptığımız deneylerde örnek sayısını azaltmıştır. LZ77 ise genelde hem örnek sayısı bakımından hem de entropi açısından sıkıştırma yapmıştır. Dolayısıyla LZ77 ve LZ78 sıkıştırma algoritmaları değerlendirilirken sıkıştırılmış verinin uzunluğuna da bakmak gerekir. LZ77 ve LZ78 boyutlar oranı cinsinden sıkıştırma performansları çizelge 7.2'de verilmiştir.

Çizelge 7.2. LZ77 ve LZ78'in sıkıştırma yüzdeleri
(100x Boyut(Üretilen) / Boyut(Baslangıç))

	.exe	**.wav**	**Resim**	**.txt**	**.xml**	**Genel**
LZ77	83	79	52	52	24	58
LZ78	71	64	52	50	44	56

Çizelge 7.1'e tekrar geri dönüldüğünde resim, ses ve xml dosyalarında mycielski78 en az LZ77 kadar genelde daha iyi sonuç verdiği görülmektedir. Uygulama ve metin dosyalarında ise bu başarıyı söz konusu değildir. Bunun nedeni, resim, ses ve xml dosyaları örneklerinin birbiriyle olan ilşkisi yakalanabilirken, uygulama ve metin dosyalarındaki örnekler arasındaki ilişkinin yakalanamamasıdır. Özellikle metin dosyalarında karakterler arasındaki fark bilgisi bu karaktarlerin ASCII kodlarının farkı alınarak bulunduğu için harflerin birbirleri yerine hangi sıklıkla kullanıldığı bilgisinden çok, alfabetik sıra bilgisi kullanılmıştır. Bir diğer deyişle metin dosyalarındaki karakterler için kullanış yerine ve sıklığına göre bir kodlama kullanılarak Mycielski78 programıyla bu tür dosyalarda da sıkıştırmada verim alınabilir.

8. RASSALLIK SONUÇLARI

Ardışık anahtarları bir önceki alt dizinin kuyruk kısmından üreten sınırlı geçmiş yöntemi yapılan deneylerde çeşitli değişiklikler yapılarak gözlemlenmiştir. Sınırlı geçmiş yönteminde birbirine zincir şeklinde bağlanmış her bir anahtar kelime bloğu kendisinden önce gelen bloğun kuyruk kısmından üretilmektedir. Bu yöntemin sonuçları yapılan deneylerle gözlenmiştir. Kuyruk kısmının ve geçmiş sınırlamasının bir diğer değişle anahtar kelimeler bloğunun uzunlukları oluşturulan dizinin rassallığını önemli bir şekilde etkilemektedir. Beklenildiği gibi, geçmişin ve kuyruğun daha uzun alınması üretilen dizinin rassallık kalitesini arttırmaktadır. Çizelge 8.1'de testler Bölüm 6'daki sıraya göre 1 ile 14 arasındaki sayılarla numaralandırılmıştır. Tabloda satırlar farklı kuyruk uzunluklarını, sütunlar farklı geçmiş uzunluklarını belirtmektedir. Kuyruk uzunlukları bit geçmiş uzunlukları Kbit cinsindendir. Farklı geçmiş uzunluğunun ve farklı kuyruk uzunluğunun kesiştiği konumdaki kutucuk, bu iki şartın aynı anda sağlandığı durumda hangi testlerden geçildiğini numaralarıyla göstermektedir. Buna ek olarak kutucuk içinde ne kadar çok testten başarıyla geçildiyse kutucuk o kadar koyu bir renkle doldurulmuştur.

Bu analizde her başlangıç koşulu için farklı 260 Kbitlik test verileri elde edilmiştir. Genel olarak bakıldığında hem geçmiş hem de kuyruk uzunluğunun artması rassallık kalitesini arttırmıştır. Fakat 64 bit uzunluğundaki kuyruk ve 100 Kbit uzunluğundaki geçmiş gerçek rassallık için yeterli olmuş, bir diğer değişle bundan daha uzun geçmiş ve kuyruk için rassallıkta tam başarı sağlanmıştır.

Sonuçlar ayrıntılı bir şekilde incelendiğinde Maurer'in testinin daha çok geçmiş uzunluğuna bağlı olduğu görülür. Bilindiği üzere bir çok sözde rassal sayı üreteci kayıpsız sıkıştırma testini(Maurer'in testi) geçmekte başarı göstermektedir. Dolayısıyla Antimycielski'nin bu testte başarı göstermesi beklenmedik bir sonuç değildir. Bir diğer ilginç sonuç ise yaklaşık entropi testinin geçmiş boyutuna değil, kuyruk boyutuna bağlı olmasıdır. Bir diğer değişle anahtar sözcük ne kadar uzun tutulursa üretilen dizinin entropisi o kadar yüksek olmaktadır. Bu bağıntı sınırsız geçmişin uygulandığı durumlarda görülmemektedir. Herhangi bir alt dizinin son bitleri(kuyruğu) bir sonraki alt dizinin üretiminde anahtar kelime görevi yapmaktadır. Bu anahtar kelimenin boyutu ne kadar uzun alınırsa bir sonraki alt diziye o kadar çok rassal başlangıç noktaları taşınacak bu da doğal olarak entropiyi arttıracaktır. Anahtar kelimenin kısa tutulduğu durumlarda bir önceki ile bir sonraki alt dizilerin kuyrukları aynı olma olasılığı buna bağlı olarak alt dizilerin kendilerinin de aynı olma olasılığı artacaktır. Buna bağlı olarak entropi düşecek ve rassallık özelliği zarar görecektir. Bu bağımlılık kayan

pencere yöntemiyle ortadan kaldırılabilir. Anahtar sözcük(kuyruk) 64 bit veya üzeri tutulduğunda rasal gezinim değişkesi testi, ikili matris kertesi testi ve tekrar testi başarıyla geçilmektedir. Seri test incelendiğinde bu testin geçmiş boyutuna bağlı olduğu görülür. En kolay geçilen test blok içinde frekans testidir. Bunu kesişmeyen şablonlar testi izlemektedir. Kesişmeyen şablonlar testinin başarıyla geçilmesi belli bir uzunlukta maksimum sayıda alt dizi birleşimlerine rahatlıkla ulaşılabildiğini göstermektedir. Üçüncü başarılı test ise kümülatif toplamlar testidir. Alt dizilerin kümülatif toplamlarının dizinin kümülatif toplamı etrafında salınım yaptığı kolaylıkla görülmektedir.

Çizelge 8.1. Farklı geçmiş ve anahtar kelime uzunluklarına göre Antimycielski rassallık performansları

	Geçmişin Maksimum Uzunluğu (Kbit)								
Anahtar Kelime Uzunluğu(bit)		**1**	**2**	**5**	**10**	**20**	**50**	**100**	**250**
	2	**(2)(3)(5)(7)**	**(2)(3)(5)(7)(11)**	**(2)**	**(2) ((7)/2)(11)**	**(2)(7)(11)**	**(2)(3)(5)(7)(8)(11)**	**(2)(3)(5)(7)(8)(11)(12)**	**(1)(2)(3)(5)(6)(7)(8)(11)(12)**
	10	**(2)(7)(11)(12)**	**(2)(3)(5)(7)(12)**	**(2)(3)(7)(11)**	**(1)(2)(3)(5)(7)(11)(12)**	**(1)(2)(3)(5)(7)(9)(10)(11)(12)**	**(1)(2)(3)(4)(5)(7)(8)(9)(10)(11)(12)(13)**	**(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)**	**(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)**
	64	**(1)(2)(4)(5)(7)(9)(10)(11)(12)**	**(1)(2)(4)(5)(7)(9)(10)(11)(12)**	**(1)(2)(3)(4)(5)(7)(9)(10)(11)(12)((13)/2)**	**(1)(2)(3)(4)(5)(7)(9)(10)(11)(12)((13)/2)**	**(1)(2)(3)(4)(5)(7)(9)(10)(11)(12)((13)/2)**	**(1)(2)(3)(4)(5)(7)(8)(9)(10)(11)(12)(13)**	**(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)**	**(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)**
	128	**(1)(2)(4)(5)(7)(9)(10)(11)(12)**	**(1)(2)(4)(5)(7)(11)(12)**	**(1)(2)(3)(4)(5)(7)(9)(10)(11)(12)((13)/2)**	**(1)(2)(3)(4)(5)(7)(9)(10)(11)(12)(13)**	**(1)(2)(3)(4)(5)(7)(9)(10)(11)(12)((13)/2)**	**(1)(2)(3)(4)(5)(7)(8)(9)(10)(11)(12)(13)**	**(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(14)**	**(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)(12)(13)(14)**

9. ŞİFRELEME SONUÇLARI

Üretilen dizide rassallık aşaması başarıyla geçildikten sonra yapılması gereken adım gerçek hayatta herhangi bir veriyi şifreleme performansının ölçülmesidir. Klasik yaklaşım başlangıç anahtar kelimemizden tamamen farklı rassal bir anahtar kelimeyle başlayıp her aşamada bu kelimenin bitlerini değiştirerek şifrelenen veriyi daha iyi bir şekilde çözmeye çalışmaktır. Bu yöntemde başlangıç anahtar kelimesine daha yakın kelimelerin daha iyi sonuç vermesi beklenmektedir. Fakat yapılan deneylerde anahtar kelimeye ne kadar yaklaşılırsa yaklaşılsın aynı anahtar kelime girilmediği sürece üretilen rassal dizi tamamen farklı bir dizi olmakta ve şifrelenen veri korunabilmektedir. Yapılan deneyler için çözülebilen veri miktarları çizelge 9.1'de 1) 150000 karakterlik metin dosyası, 2) 22 KBaytlık DNA dizisi, 3) 32 KBaytlık dizi halindeki gri tonlanmış resim ve 4) 44 KBaytlık dizi halindeki farklı bir gri tonlanmış resim için yaklaşık anahtare kelimeler ve tamamen rassal diziler uygulanarak verilmiştir. Girilen anahtar kelimeler başlangıç anahtar kelimesine çok yakın olmasına rağmen, çözülebilme oranları tamamen rassal dizilerle çözülebilme oranları kadar küçük kalmıştır. Ardışık olarak birden fazla baytın çözülebildiği ise nerdeyse hiç görülmemiştir. Bu veriler Antimycielski ile şifrelenen verinin orijinal anahtar olmadan ancak kaba kuvvet yöntemiyle çözülebileceğini göstermektedir. Bu yöntemi günümüz bilgisayar teknolojisiyle gerçekleştirmek imkansızdır.

Çizelge 9.1. Farklı denemeler için çözülen bayt ve 2bayt sayıları

	Metin (148484 Bayt)		**DNA Dizisi (22532 Bayt)**		**Resim1 (33168 Bayt)**		**Resim2 (45562 Bayt)**	
	Çözülen Bayt	**Çözülen 2 Bayt**	**Çözülen Bayt**	**Çözülen 2 Bayt**	**Çözülen Bayt**	**Çözülen 2 Bayt**	**Çözülen Bayt**	**Çözülen 2 Bayt**
1 bit yaklaşık	592	1	105	0	148	0	193	0
2 bit yaklaşık	589	2	93	0	131	0	188	1
3 bit yaklaşık	557	4	94	0	127	0	175	2
4 bit yaklaşık	622	4	98	0	139	0	200	0
Rassal1	586	5	80	0	120	0	173	0
Rassal2	596	2	97	0	124	0	173	0

10. SONUÇ VE DEĞERLENDİRME

Bu çalışmada sınırsız geçmişte kendini tekrar eden serileri arayarak buna göre gelecek sembolü öngörü eden Mycielski algoritmasının üzerine öngörüye dayalı sıkıştırma ve öngörünüye dik örnekler üreterek de rassal seri üretimi ve şifreleme uygulamaları geliştirilmiştir. İlk aşamada sıkıştırma amacı ile sözlük üretimi tabanlı bir zaman iyileştirmesi önerilmiş, ve elde edilen yönteme Mycielski78 denmiştir. Mycielski öngörücüsü gücünü önerilen Mycielski78 sıkıştırıcısında göstermektedir. Bu sıkıştırıcı özellikle yüksek koralasyonlu resim ve ses dosyalarında önemli bir entropi düşürme oranı göstermiştir. Ayrıca öngörücü eklenen sözlük yapısıyla gücünü kaybetmeden büyük bir hız kazanmıştır. Elde edilen sonuçlar tamamen farklı yöntem uygulamasına rağmen LZ türü sıkıştırma algoritmalarına yakın performans göstermiştir. Mycielski öngörücüsüne dayanan Antimycielski rassal maske dizisi üretim süreci başlangıç dizisini zıt öngörüler elde edip bu öngörülere kuyruğa ekleyerek genişletmektedir. Bu sayede diziye öngörü edilemez örnekler eklenmekte ve dizi rassal bir özellik kazanmaktadır. Önerilen bu rassal sayı üretme ve şifreleme sistemi, yapılan deneylerde hem rassallık açısından hem de şifreleme açısından başarılı sonuçlar vermiştir. Yapılan gerçeklemeler bu yöntemle şifrelenen verinin, yalnız ve yalnız başlangıç anahtar kelimesinin bu üretece sokulup elde edilen rassal diziyle maskelenmesiyle çözülebileceğini göstermiştir.

KAYNAKLAR

[1] Jacquet, P., Szpankowski, W. ve Apostol, I., "A Universal Predictor Based on Pattern Matching" , *IEEE Trans. Inform. Theory,* **48(6)**, 1462-1472, Haziran 2002.

[2] Salomon, D., "LZ77 (Sliding Window)", "Data Compression-The Complete Reference" , Springer-Verlag Newyork, 104-106, 1998.

[3] Salomon, D., "LZ78", "Data Compression-The Complete Reference" , Springer-Verlag Newyork, 112-116, 1998.

[4] Menezes, A., van Oorschot, P. ve Vanstone,S., "Block Ciphers", "Handbook of Applied Cryptography", CRC Press, 223-282, 1996.

[5] Mirza, F., "Block Ciphers and Cryptanalysis" , 1998, http://members.aol.com/jpeschel3/report.pdf .

[6] Ehrenfeucht, A., Mycielski, J, "A Pseudorandom Sequence–How Random is It", American Mathematical Monthly , **99**, 373–375, 1992.

[7] A. Rukhin ve ark., "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications" , 2001, http://csrc.nist.gov/rng/SP800-22b.pdf .

[8] Leon-Garcia, A., "The Expected Value of Random Variables","Probability and Random Processes for Electrical Engineering", Addison-Wesley Publishing, 126-130, 1994.

[9] Leon-Garcia, A., "Entropy", "Probability and Random Processes for Electrical Engineering", Addison-Wesley Publishing, 162-170, 1994.

[10] Sayood, K., "Compression Techniques", "Introduction to Data Compression", Morgan Kaufmann Publishers, San Francisco, 3-5, 2000.

[11] Sayood, K., "A Brief Introduction to Information Theory", "Introduction to Data Compression", Morgan Kaufmann Publishers, San Francisco, 13-22, 2000.

[12] Sayood, K., "Predictive Coding", "Introduction to Data Compression", Morgan Kaufmann Publishers, San Francisco, 139-176, 2000.

[13] Ross, S., "Expected Value", "A First Course in Probability", Prentice-Hall, Inc., New Jersey, 136-139, 1998.

[14] Ross, S., "Variance", "A First Course in Probability", Prentice-Hall, Inc., New Jersey, 142-144, 1998.