Journal of the Faculty of Engineering and Architecture of Gazi University 38:2 (2023) 771-780

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University
Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

Determination of canonical Huffman codeword lengths by evolution strategies

Mustafa Oral, M. Mustafa Aşşık*
Department of Computer Engineering, Faculty of Engineering, Cukurova University, 01330, Adana, Türkiye

Highlights:

- Evolution Strategies in Data Compression
- An alternative algorithm for Canonical Huffman Coding
- Optimization of Algebraic Canonical Huffman Coding

Keywords:

- Data Compression
- Canonical Huffman Encoding
- Evolutional Algorithms
- Evolution Strategies

Article Info:

Research Article
Received: 18.02.2021
Accepted: 07.04.2022

DOI:

10.17341/gazimmfd.882745

Correspondence:

Author: M. Mustafa Aşşık
e-mail: massik@cu.edu.tr
phone: +90 506 508 5030

Graphical/Tabular Abstract

In this study, it is shown how optimum canonical Huffman code lengths can be generated using the evolution strategies algorithm. In the algorithm used, the Huffman code lengths close to the optimum obtained by an algebraic method are used as the first ancestor. The optimum code lengths are produced as a result of the evolution of these code lengths by mutation. The steps of the algorithm are shown in Figure A.

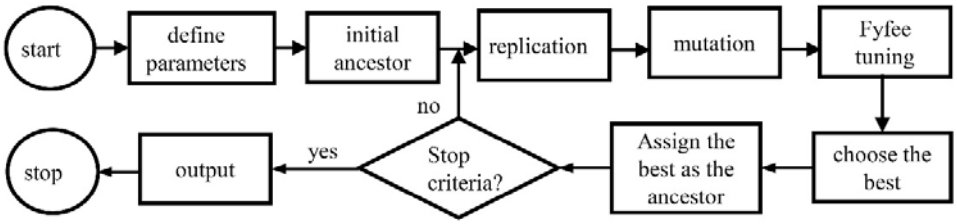


Figure A. The diagram of the proposed algorithm

Purpose:
The purpose of this study is to propose an alternative algorithm that generates traditional canonical Huffman codewords using evolution strategies algorithm and to optimize the Algebraic Canonical Huffman Coding (ACHC) algorithm.

Theory and Methods:
In this study, Evolution Strategies algorithm was used to find the best Huffman code lengths. The ACHC algorithm was used to generate the first ancestor. ACHC is based on "u=s + *l*" formula, where u , considering an imaginary Huffman tree, is the length (or depth) from the root to leaves corresponding to symbols, s is the length from the root to a reference node and l is the length from the reference node to leaves. l is calculated by $l_i = R(\log(e_i / p_i))$ formula, if $p_i < 0.7$. Otherwise $l_i = 1$. R is rounding function and p_i is the probability of i^{th} symbol in the used alphabet, where $1 \leq i \leq n$ and n is the number of the used symbols. The parameter e in the formula is equal to 1 (the sum of the probabilities) in the beginning. However, considering the imaginary Huffman tree, as the reference node descends one level from the root, the parameter e decreases by the sum of the probabilities of the processed symbols. As a result, the code lengths corresponding to the codewords are calculated and canonical Huffman codes are obtained using the calculated lengths. The algorithm that determines the canonical Huffman code lengths by Evolution Strategies begins with one ancestor that is obtained by ACHC. The ancestor is multiplied by replicating itself. The offspring are mutated by the "target directed mutation" method arranged according to the relationship between optimum lengths and near-optimum lengths. The offspring with the best fitness value among the mutated offspring becomes the ancestor of the next generation. The process continues until the stop criterion is met. The fitness function is the average bit length per symbol. Fyffe tuning is used to maintain Kraft-MacMillan inequality.

Results:
The proposed algorithm is tested by using Calgary Corpus. Compared to alphabet size (n), optimum results are obtained with small loops (loop number < n). If the loop number is taken as a constant number (~n), the time complexity is $O(n^2)$ steps and the space complexity is $O(n^2)$ byte.

Conclusion:
In this study, an alternative algorithm that generates optimum canonical Huffman codes using evolution strategies algorithm is proposed. Also, ACHC algorithm is optimized by the proposed algorithm. The optimum results are obtained in a very short loop time. The proposed algorithm can be used for larger alphabets such as syllables, words, etc.

Journal of the Faculty of Engineering and Architecture of Gazi University 38:2 (2023) 771-780

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University

Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

Kanonik Huffman kod sözcükleri uzunluklarının evrim stratejileri algoritması ile belirlenmesi

Mustafa Oral, M. Mustafa Aşşık*
Çukurova Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 01330, Adana, Türkiye

Ö N E Ç I K A N L A R

- Veri Sıkıştırmada Evrim Stratejileri
- Kanonik Huffman Kodlama için alternatif bir algoritma
- Cebirsel Kanonik Huffman Kodlamasının Optimizasyonu

Makale Bilgileri
Araştırma Makalesi
Geliş: 18.02.2021
Kabul: 07.04.2022

DOI:
10.17341/gazimmfd.882745

Anahtar Kelimeler:
Veri sıkıştırma
Kanonik Huffman kodlama
evrimsel algoritmalar
evrim stratejileri

ÖZ

Huffman kodlama, veri sıkıştırma alanında yaygın bir şekilde kullanılmaktadır. Kanonik Huffman kodlama ise, Huffman kodlamanın bir alt kümesidir ve daha kısa başlık ve daha az hafıza yeri kullanılması gibi bazı avantajlara sahiptir. Bu nedenle bu kodlamayla ilgili geliştirme çalışmaları devam etmektedir. Cebirsel Kanonik Huffman Kodlama (CKHK) da bu çalışmalardan birisidir ve bu algoritma ile en iyi değere en yakın Huffman kod uzunlukları cebirsel yoldan elde edilmektedir. Bu çalışmada, kanonik Huffman kodlarının üretimine esas olan kod uzunluklarını Evrimsel Stratejiler (ESs) ile elde eden bir algoritma önerilmekte ve söz konusu algoritma aynı zamanda CKHK algoritmasının ESs yöntemi ile en iyileştirilmesi anlamına gelmektedir. ESs çoğunlukla mutasyonu kullanan bir evrimsel algoritmadır. Tek bir ata çoğalarak kendi kopyalarını oluşturur. Kopyalar mutasyona uğratılarak çocuklar elde edilir. Çocuklar ve atanın arasından en iyi uygunluk değerine sahip birey bir sonraki neslin atası seçilir. Durma şartı sağlanıncaya kadar bu döngü devam eder. Bu çalışmada ilk ata olarak CKHK ile edilen uzunluk dizisi kullanılmıştır. Bu atanın mutasyonla evrimleşmesi sonucunda en iyi değere ulaşılmıştır. Optimum değere ulaşmak için gerekli döngü sayısı testler sonucunda sabit bir sayı olarak belirlenmiş olup, bu durumda zaman karmaşıklığı, n alfabe sayısı olmak üzere $O(n^2)$ olarak tespit edilmiştir. Kullanılan hafıza miktarı ise çoklu bireyler nedeniyle $O(n^2)$ bayttır.

Determination of canonical Huffman codeword lengths by evolution strategies

H I G H L I G H T S

- Evolution Strategies in Data Compression
- An alternative algorithm for Canonical Huffman Coding
- Optimization of Algebraic Canonical Huffman Coding

Article Info
Research Article
Received: 18.02.2021
Accepted: 07.04.2022

DOI:
10.17341/gazimmfd.882745

Keywords:
Data compression
Canonical Huffman encoding
evolutionary algorithms
evolution strategies

ABSTRACT

Huffman coding is widely used in data compression. Canonical Huffman coding is a subset of Huffman coding and has some advantages, such as shorter header and less memory usage. Therefore, studies on this coding have been continuing. The algorithm producing the lengths of Algebraic Canonical Huffman Codes (ACHC) is one of these studies and this algorithm obtains the code lengths nearest to optimum value. In this study, an algorithm obtaining the code lengths that are the basis for the production of canonical Huffman codes using Evolutionary Strategies (ESs) is proposed, and this algorithm means that the ACHC algorithm is optimized by the ESs method. ESs is an evolutionary algorithm that mostly uses the mutation. Replication creates many copies of a single ancestor. The copies are mutated to produce offspring. Among the offspring and the ancestor, the individual with the best fitness value is chosen as the ancestor of the next generation. This cycle continues until the stop criteria is met. In this study, the length array obtained by ACHC was used as the first ancestor. The optimum value was achieved as a result of the evolution of this ancestor by mutation. As a result of the tests, the required number of cycles to reach the optimum value was determined as a fixed number. In this case, the time complexity is $O(n^2)$, where n is the number of symbols used. The amount of memory used is $O(n^2)$ because of the usage of multiple individuals.

*Sorumlu Yazar/Yazarlar / Corresponding Author/Authors : moral@cu.edu.tr, *massik@cu.edu.tr / Tel: +90 506 508 5030

1. Giriş (Introduction)

Hafıza cihazlarının kapasitesinin ve iletim hızlarının artmasına karşın, dosya boyutlarının da artmasından dolayı veri sıkıştırma, iletişim kanalları ve depolama cihazları açısından hala önemli bir alandır. Bu nedenle, araştırmacılar veri sıkıştırma algoritmaları üzerinde çalışmaya devam etmektedirler. Huffman kodlama da bu sıkıştırma algoritmalarından biridir. Araştırmacılar, Huffman kodlama için kullanılan geleneksel yöntem dışında donanım tabanlı tasarım gibi farklı bakış açılarından yola çıkarak yeni algoritmalar ortaya çıkarmaktadırlar [1, 2]. Evrimsel algoritmalar (EA) da metin/resim/video alanlarında veri sıkıştırması için üzerinde çalışılan farklı bakış açılarından biridir. Evrimsel algoritmalar evrimsel ilkelere dayanan algoritmaların bir örneğidir. Evrimsel algoritmalar; evrim stratejileri, evrimsel programlama, genetik algoritma ve genetik programlamalardan oluşur [3]. Evrimsel algoritmalara temel olan ilkeler; üreme, mutasyon, rekabet ve seçilimden meydana gelir [4]. Rechenberg ve Schwefel tarafından geliştirilen Evrim Stratejileri (ESs), çözüm uzayındaki en uygun değere ulaşmak için bireyleri mutasyon yoluyla değişime uğratan strateji parametreleri kullanır. ESs çoğunlukla mutasyona dayanır ve bazı durumlarda çaprazlamayı da kullanabilir. Seçilim operatörü her zaman en iyi birey seçildiği için deterministtir. Ebeveyn ve çocuk sayısı genellikle birbirlerinden farklıdır. L. J. Fogel tarafından geliştirilen Evrimsel Programlama (EP) sadece mutasyona dayanır ve çaprazlama kullanmaz. Seçilim operatörü olasılıksaldır (turnuva seçilimi). Holland tarafından tanımlanan Genetik Algoritmalar (GAs) en önemli arama operatörü olarak çaprazlamayı ve çok küçük bir olasılık değeri ile de mutasyonu kullanır. GAs olasılığa dayalı seçilim kullanır [5, 6]. Bahsedilen algoritmalara ilave olarak, Genetik Programlama (GP) bilgisayar programları geliştirmek amacıyla çaprazlamaya dayanan evrimsel arama ilkesini kullanır [7].

Metin alanında yapılan çalışmalarda EA doğrudan kodlama için değil, kodlamaya esas olacak sözlük seçimi veya sıkıştırılacak dosya için en iyi sıkıştırma algoritmasının seçimi gibi nedenlerle kullanılmıştır. Üstelik bu çalışmalarda çoğunlukla GA ve GP kullanılmıştır. Bu makalede; ikinci bölümde bahsedilen çalışmalardan farklı olarak ve bireylerin permütasyon kodlama ile kodlandığı bir evrimsel algoritma kullanılarak, kanonik Huffman kodları için gerekli olan kod uzunluk değerlerini (bit sayıları) üreten bir algoritma önerilmiş ve analizi yapılmıştır. Bu çalışmada, kod uzunlukları bir kere belirlendikten sonra, n kullanılan alfabedeki sembol sayısı olmak üzere, gerekli kodlar n adımda kolaylıkla üretilir.

Bir Huffman ağacının yapraklarındaki sembollere karşılık gelen kod sözcüklerinin uzunlukları bir uzunluk dizisi (L=$\{l_1, l_2, \dots, l_n\}$, l, n>0 ve $l_{n-1} = l_n$) oluşturur. Bu uzunluk dizisi bireyi meydana getirir. Çocuklar, tek bir bireyin (ata) kendi benzerlerini çoğalttıktan sonra mutasyona uğramaları ile elde edilir. Mevcut atadan ve çocuklar arasından, sembol başına minimum ortalama bit sayısını sağlayan en iyi birey bir sonraki ata olarak seçilir. Yeteri kadar uygulanan döngüden sonra kanonik Huffman kod sözcükleri, seçilen en iyi bireyden (uzunluk dizisi) üretilir. Kullanılan evrimsel algoritma, ESs algoritmasıdır. Ancak; ESs'in bir özelliği olarak evrimleşmesi gereken strateji parametresi, kullanılan modelde evrimleşmemekte ve her kuşakta her birey için rastgele (1 veya -1) seçilmektedir. Bu nedenle kullanılan algoritma "sabit parametreli evrim stratejileri" olarak adlandırılabilir. Aynı zamanda söz konusu algoritma cinsiyetsiz üreme modelinin bir uygulamasıdır.

Bir metin dosyasını sıkıştırmak için genellikle 256 karakterlik alfabe (monogram) kullanılır. 2-gram, 3-gram veya kelime tabanlı sözlükler kullanmak daha iyi sıkıştırma oranı sağlayabilir. Bir metin dosyasında olasılık hesabı için 256 adet sayıcı kullanılır. 2-gram kullanılırsa 256^2 + 256 = 65792 adet sayıcı ve her karakter için iki farklı karşılaştırma gerekir. 3-gramda ise 16 843 008 adet sayıcı ve her karakter için üç farklı karşılaştırma gerekir. Bu oldukça maliyetli bir iştir. Evrimsel algoritmalar bu noktada bir kolaylık sağlar. Ayrıca sıkıştırılacak dosyanın olasılık bilgisinin önceden bilinmesine gerek yoktur [8]. Bu çalışmada ESs kullanılarak monogram alfabe için en iyi kanonik Huffman kod uzunlukların üretilmesi incelenmiş, elde edilen sonuçlar sonraki çalışmalarımızda n-gramlı alfabeler için de bir temel teşkil edecektir.

İlk ata, Oral ve Aşşık tarafından önerilen Cebirsel Kanonik Huffman Kodlama (CKHK) algoritması kullanılarak elde edilmiştir [9]. CKHK, geleneksel Huffman kodlama tarafından üretilen sembol başına ortalama bit sayısına benzerlerine göre en yakın ortalama değeri elde eden uzunluk dizisini üretmektedir. Böylelikle en iyi değere ulaşmak için gerekli evrim süresi kısalmaktadır. Aynı zamanda yapılan çalışma yakın zamanda önerilen ve henüz üzerinde bir çalışma yapılmayan CKHK algoritmasının en iyileştirilmesi anlamına da gelmektedir.

Önerilen algoritma başlangıçta GA kullanılarak tasarlanmıştır. Ancak parametre değerleri belirlenirken en iyi değerlerin çaprazlama oranından bağımsız olarak yüksek mutasyon değerlerinde elde edildiği görülmüştür. Bu da sonuca giden evrimleşme sürecinin çaprazlamanın değil mutasyonun bir fonksiyonu olduğunu göstermektedir. Bu nedenle çaprazlama iptal edilerek evrim operatörü olarak sadece mutasyon kullanılmıştır. Çaprazlamanın olmadığı bir evrimsel algoritmanın ise büyük ölçüde genetik aktarıma dayanan "Genetik Algoritma" olarak tanımlanamayacağı açıktır. Seçilimin de deterministtik olması nedeniyle son aşamada kullanılan EA algoritması ESs'e dönüşmüştür. GA kullanılarak elde edilen döngü sayıları yedinci bölümde verilmiştir.

Bu makale şu şekilde düzenlenmiştir: İkinci bölümde literatür taramasıyla ulaşılan kaynaklarda konuyla ilgili yapılan önceki çalışmalar, üçüncü bölümde Kanonik Huffman kodlama, dördüncü bölümde CKHK algoritması ve beşinci bölümde ESs anlatılmıştır. Önerilen algoritma ES-CKHK altıncı bölümde sunulmuştur. Testler ve sonuçları yedinci bölümde analiz edilerek, sekizinci bölümde de sonuçlar tartışılmıştır.

2. Önceki Çalışmalar (Related Works)

Bu makalenin konusu olan metin sıkıştırması alanında yapılan bazı veri sıkıştırma çalışmaları, evrimsel algoritmaların üyelerinden genetik algoritma (GA) ve genetik programlama (GP) kullanılarak yapılmıştır.

Üçoluk ve Toroslu, veriyi sıkıştırmadan önce harflere ve hecelere dayalı olarak kullanılacak en iyi alfabeyi seçmek için GA kullandılar. Daha sonra standart Huffman algoritması kullanarak Türkçe metinleri sıkıştırmak için seçilen alfabeyi kullandılar. Heceleri elde etmek için, Türkçeye özgü bir heceleme algoritması kullanıldı. Bu çalışmada öncelikle sıkıştırılacak metni oluşturan sembolleri ve metinde bulunan tüm heceleri kapsayan bir ana alfabe oluşturulur. GA ile bu alfabe içinden en iyi sıkıştırma oranını verecek olan bir alt alfabe seçilir. Alt alfabeler şu şekilde oluşturulur: Eğer ana alfabede yer alan bir hece alt alfabede yer alacaksa ikili düzende "1" değerini alır, aksi takdirde "0" değerini alır. "0" değerini alan hece bileşenlerine ayrılır ve her bileşen alt alfabeye bağımsız olarak katılır. Örneğin, "ab" hecesi "0" değerini almış ise "a" ve "b" sembolleri alfabede bulunan "a" ve "b" karakterlerinin frekanslarını arttırırlar. Bu şekilde ikili düzende oluşturulan alfabeler bireyleri oluştururlar. İlk bireyler 100 adettir ve rastgele oluşturulur. Çaprazlama ve mutasyon sonucu elde edilen yeni bireyler uygunluk değeri olarak kullanılan sıkıştırma oranı ile

değerlendirilir. Seçilimde seçkinci (elitist) yöntem uygulanır. Önceden belirlenen bir oran kadar en iyi birey önceki kuşağın en kötüleri ile değiştirilir. Sürecin tamamlanması ile seçilen en iyi alfabe kullanılarak Huffman kodlama ile metin sıkıştırılır. Sonuçta kullanılan alfabede hecelerin de yer almasıyla sadece sembollerin kullanıldığı alfabeye göre daha iyi sıkıştırma oranı elde edilmiştir [10].

Oroumchian vd., Farsça WEB sayfaları ve internet tabanlı diğer uygulamalarda kullanılmak üzere en iyi veya en iyiye yakın sıkıştırma oranını sağlamak amacıyla 2-harfli, 3-harfli, 4-harfli ve 5-harfli sözcüklerden oluşan en uygun sözlüğü seçmek için önerdikleri algoritmalarında GA kullandılar. Dilde çok kullanılan n-harfli sözcükler bir eğitim seti kullanılarak tespit edilir. GA ile seçilen n-harfli sözcükler Unicode tablosunun kullanıcı tanımlı bölümündeki tek karakterlerle değiştirildiler. Böylelikle metin dosyasının boyutunda %52 oranına kadar azalma sağladılar. Sıkıştırma işlemi sadece sunucuda gerçekleştirilmekte ve uygun Unicode tablosunun kullanıcı tarafından önceden yüklenmesi şartıyla, kullanıcı tarafında çözme (decode) işlemi yapılmamaktadır. Yöntem her dile uygulanabilir. Ancak yöntemin başarısı n-harfli sözcüklerin tespiti için kullanılan eğitim setine de bağlıdır [11].

Kuthan ve Lánský çalışmalarında Üçoluk ve Toroslu'nun yaklaşımından esinlendiler. Heceleri elde etmek için her dil ile kullanılabilecek dört farklı evrensel heceleme algoritması geliştirdiler. Sıkıştırılacak metnin yazıldığı dilde yazılmış belgelerden oluşan bir eğitim seti kullanarak, heceleme algoritması ile olası tüm heceleri içeren bir ana alfabe oluşturdular. Bu ana alfabeden türetilecek en iyi sıkıştırma oranını sağlayacak alt alfabe GA ile seçildi. Alt alfabeler, ana alfabedeki hecelere "1" veya "0" değeri verilerek oluşturuldu. Sıkıştırma esnasında ilk defa karşılaşılan heceler bileşenlerine ayrılarak kodlandılar ve hece tablosuna eklendiler. Daha sonra bu heceyle karşılaşıldığında diğer heceler gibi işlendiler. Çekçe ve İngilizce metinler, GA ile seçilen en iyi sözlük kullanılarak hece tabanlı LZWL ve HuffSyllable algoritmaları ile sıkıştırıldılar. Sıkıştırılan metinler Gzip 1.3.5, Bzip2 1.0.3 ve Compress 4.2 programları ile karşılaştırıldılar. Sonuçlar değerlendirildiğinde, HuffSyllable + GA ikilisi ile 100 Bayt – 10 KBayt arası Çekçe metinlerde ve 100 Bayt – 1 KBayt arası İngilizce metinlerde daha iyi sıkıştırma oranı elde edildiğini gösterdiler [12].

Tüm veri tiplerini aynı oranda sıkıştırabilecek bir evrensel sıkıştırma algoritması olmadığı düşüncesinden yola çıkan Kattan ve Poli, verilen bir dosyayı küçük parçalara bölerek her bir parçayı en iyi sıkıştıracak algoritmayı tespit etmek için bir GP sistemi önerdiler. Bu sistemde birey (kromozom) sıkıştırılacak dosyanın kendisidir. Parçalar ise genlerdir. Her gene rasgele bir sıkıştırma algoritması ve/veya bir dönüşüm algoritması atanır. Sıkıştırma algoritmaları da önceden belirlenen bir fonksiyon setinden seçilir. Diğer bireyler de aynı genlere farklı sıkıştırma algoritmaları atanması ile elde edilir. Bu şekilde oluşturulan bireylere genetik operasyonlar uygulanarak en iyi sıkıştırma oranını veren birey seçilir. Daha sonra parça uzunluğu arttırılarak aynı işlemler tekrar edilir. Böylece olası tüm parça uzunlukları denenerek en iyi sıkıştırma oranına sahip parça uzunluğu ve algoritma dizisi seçilir. Ancak bu sistemin en büyük dezavantajı zaman maliyetidir (1 Mb için 1 gün). Ayrıca parça uzunluğunun ve hangi parçanın hangi algoritmayla sıkıştırıldığının bilinmesi için bir başlık bilgisi gerektirir. Yapılan testlerde, metin ve exe dosyalarında en iyi sonucun, düzenli veri yapısına sahip olmaları nedeniyle, tek parçada elde edildiği görülmüştür. Buna karşılık farklı veri tiplerinden oluşan arşiv dosyalarında parça sayısı arttıkça daha iyi sıkıştırma oranı elde edilmiştir [13]. Katan ve Poli GP-ZIP adını verdikleri algoritmalarında daha sonra bazı iyileştirmeler yaptılar. GP-ZIP* olarak adlandırılan yeni algoritmalarında dosyaları eş uzunluktaki parçalara bölme yerine algoritmanın farklı uzunlukta parçalar oluşturmasına izin verdiler. Buna bağlı olarak da çaprazlama ve mutasyon operatörlerini yeniden düzenlediler. Yapılan düzenlemenin sonucu olarak daha iyi performans ve daha iyi sıkıştırma zamanı elde ettiler (1 Mb için ortalama 3,73 saat) [14].

Kattan ve Poli yukarıda anlatılan çalışmalarını GP-ZIP2 isimli çalışmaları ile bir adım daha ileriye götürdüler. Bu çalışmalarında "ayırıcı ağaç" ve "nitelik çıkarıcı ağaç" olarak isimlendirilen iki kavram kullandılar. Ayırıcı ağaç; entropi, medyan, standart sapma gibi bazı istatistiksel öğeleri kullanarak bir eğitim setinde yer alan dosyayı uygun boyuttaki bloklara böler. Her blok, fonksiyon setinde yer alan sıkıştırma algoritmaları ile sıkıştırılarak blok için en uygun sıkıştırma algoritması bulunur. 2 nitelik çıkarıcı ağaç kullanılarak, aynı istatistiksel öğeler yardımı ile her blok için bloğu temsil edecek numerik X ve Y değerleri bulunur. Her blok X ve Y değerleri ile Öklid uzayında temsil edilir. K-means algoritması ile bloklar kümelenir. Her küme içindeki blok birbirleri ile benzer özellikleri taşırken, diğer kümedeki bloklarla benzer özellikleri yoktur. Her küme, içindeki bloklarda baskın olan sıkıştırma algoritması ile etiketlenir. Bir dosya sıkıştırılacağı zaman benzer şekilde bloklara ayırılarak kümelenir ve her küme eğitim setinde elde edilen kümelerle karşılaştırılarak uygun sıkıştırma algoritması tespit edilir. GP'nin buradaki görevi, blokları Öklid uzayında gösterecek değerleri üreten en uygun nitelik çıkarıcı ağaçları bulmaktır. GP-ZIP2 özellikle heterojen arşiv dosyalarında diğer programlara göre daha başarılı olurken uygulama zamanı birkaç saat sürecek kadar yavaştır [15]. Bu yavaşlığı aşmak için Katan ve Poli GP-ZIP3'ü geliştirdiler. GP-ZIP3, GP-ZIP2 ile aynı yapıyı paylaşır. Ancak GP-ZIP2 de çok zaman alan her bloğun fonksiyon setinde yer alan sıkıştırma algoritmaları ile tek tek sıkıştırılarak sıkıştırma oranının elde edilmesi yerine, GP-ZIP3'de sıkıştırma oranını tahmin edecek bir "karar ağacı" kullanıldı [16]. Böylece uygulama zamanı 5 kat azaldı. Hız düşmesine rağmen başkaca bir geliştirme yapılmadığından sıkıştırma performansı GP-ZIP2 ile aynıdır [17].

Boisclair ve Wagner, sıkıştırılacak dosya hakkında hiçbir ön bilgiye sahip olmadan ve çok karakterli sembollerin kullanılmasına da imkân tanıyan bir sıkıştırma algoritması önerdiler. Algoritmada kullanılan GA, birey olarak aday Huffman ağaçlarını kullanmaktadır. Başlangıç popülasyonu eş uzunluğa sahip 256 karakterin farklı dizilimlerinden oluşan ağaçlardır. Algoritma, ağaçların doğası gereği bazı dizilerin kaybolması veya bazı dizilerin tekrarlanmasına yol açtığı için çaprazlama kullanmayıp sadece mutasyona dayanmaktadır. Takas (swap) mutasyonu ile yapraklar ve iç düğümler kendi aralarında veya birbirleri ile yer değiştirerek alternatif ağaçlar oluşturulmaktadır. Birleştirme (combine) mutasyonu ile de iki yaprak birleştirilerek çok karakterli sembollerin oluşturulmasına izin verilmektedir. GA sıkıştırılmış dosya boyutunu en az yapacak ağacı seçer. Sonuçta, önerilen algoritma yapılan testlerde insanın genetik şifrelerini içeren dosyalarda verimli olurken, çalıştırılabilir ve metin dosyalarında beklenen verimi gösterememiştir [8].

Zaki ve Sayed GP kullanarak bireyin bir ağacın düğümlerinden oluştuğu ve sıkıştırma oranını minimize edecek bir Huffman ağacını bulmayı hedefleyen bir algoritma önerdiler. Yazarlar, geleneksel GP operatörlerinin rasgele değişiklikler yapmasından dolayı her sembole atanacak "benzersiz önek" kuralını (prefix-free) garanti edemeyeceği düşüncesi ile bu operatörler yerine farklı operatörler kullandılar. Bunlar: Ekleme (insertion), iki seviyeli mutasyon ve değiştirilmiş çaprazlama operatörleridir. 1- Ekleme operatörü mevcut ağaca yeni alt ağaçlar eklemek için kullanılır. 2- İki seviyeli mutasyon: aralarında ebeveyn/çocuk ilişkisi olmayan rasgele iki düğüm (örneğin a ve b) seçilir. Düğüm b çocukları ile birlikte bir seviye aşağıya indirilirken, b'nin ebeveyni ile a kardeş yapılır. Varsa a'nın önceki kardeşi bir seviye yükseltilir. Böylece yeni bir ağaç elde edilir. 3- Değiştirilmiş çaprazlama: bu yöntemde anne ve baba kullanılmaz. Tek bir bireyin farklı seviyelerinden rasgele seçilen iki noktanın alt ağaçları ile beraber yer değiştirilmesi ile yeni birey elde edilir. Aslında bu işlem

ebeveyn kullanılmadığı, tek bir bireyin iç değişikliği ile yapıldığı için mutasyon işlemidir. Farklı bireylerden gen aktarımı yoktur. Ancak yazarlar bu işlemi değiştirilmiş çaprazlama olarak isimlendirmeyi tercih etmişlerdir. Başlangıç popülasyonu, kullanılan alfabedeki tüm sembolleri içeren ve rasgele oluşturulan tek bir ağaçtan oluşur. Bu bireye genetik operatörlerin uygulanması ile yeni birey elde edilir ve uygunluk fonksiyonu ile değerlendirilir. Uygunluk fonksiyonu ise sembol başına ortalama bit uzunluğudur. Yöntem tek sembollerle beraber çoklu sembollerin de kullanılmasına izin verir [18]. Algoritma en iyi Huffman ağacını hedeflediği için geleneksel Huffman kodlardan daha iyisini elde edemez. Ancak tek sembollü ağaç yerine daha iyi sıkıştırma oranı elde edebilecek çok sembollü ağacı seçebilmektedir. Yapılan testler de bu durumu doğrulamaktadır.

K.I.M. Abuzanouneh, en iyi sıkıştırma oranını elde etmek için karakterlerden, hecelerden ve kelimelerden oluşan çoklu Huffman ağacı yaklaşımını benimseyen bir GA algoritması önerdi. Bu algoritmada 256 karakterden oluşan bir başlangıç Huffman ağacını temsil eden birey, ağacın yaprak ve iç düğümlerini temsil eden 1 ve 0 bit dizisi ile gösterilir. Karakter, hece ve kelimelerden oluşan bireyler ayrı ayrı sıkıştırma oranını minimize edecek uygunluk fonksiyonu ile değerlendirilir. Kodlama için GA tarafından seçilen en iyi üç ağaç arasından en iyi sıkıştırma oranına sahip ağaç bir karar ağacı ile seçilir. Farklı bireyleri elde etmek için de çok noktalı çaprazlama ile takas, ekleme ve birleştirme gibi farklı mutasyon operatörleri kullanılır. Bu mutasyon operatörleri sayesinde hece ve kelimeler elde edilir. Seçilim yöntemi elitizmdir. Test sonuçları incelendiğinde, karakterlerden başka hece ve kelimeleri de değerlendirdiği için karakter tabanlı standart Huffman kodlamaya göre daha iyi sonuçlar elde edildiği görülmüştür. Ayrıca diğer kelime tabanlı Huffman ve LZ77 gibi algoritmalarla karşılaştırdığında da iyi sonuçlar elde edildiği görülmüştür [19].

Her metin dosyası için kullanılacak en uygun alfabenin farklı olabileceği düşüncesinden yola çıkan J. Platos ve P. Kromer, araştırmalarını GA ve Benzetimli Tavlama (BT) algoritmaları ile gerçekleştirmiştir. Bunun için sıkıştırılacak metin dosyası taranarak istenilen uzunluğa kadar (1-gram 2-gram gibi) olası tüm sembolleri kapsayan bir ana alfabe tespit edilir. Bireyler bu ana alfabedeki sembollere karşılık gelen "1" ve "0" değerleri ile oluşturulur. "1", karşılık gelen sembolün bireyde kapsandığını "0" ise kapsanmadığını gösterir. Uygunluk fonksiyonu minimize edilecek sıkıştırma oranı ve kullanılan alfabenin boyutudur. Araştırma, GA ve BT algoritmalarının yukarıdaki yapı kullanılarak farklı sıkıştırma algoritmaları ile uygulanması ile yapılmıştır. Sonuçta 1-gram uzunluğundaki (karakter tabanlı) alfabelerin küçük dosyalar için (200 KB'a kadar), 2-gram, 3-gram gibi hece yapılarının orta boydaki dosyalar için (200 KB – 5 MB), kelime tabanlı alfabelerin ise görece daha büyük dosyalar için uygun olduğu gösterilmiştir [20-22].

Literatürde kanonik Huffman kodları kullanılarak yapılan birçok çalışma vardır. Bu nedenle bu bölümde konu ile ilgili birkaç örnek verilecektir. Yapılan çalışmaların hepsinde kanonik Huffman kodları geleneksel yöntemle elde edilmiştir. Bu yöntem üçüncü bölümde anlatılmıştır.

Yair Wiseman Küresel Konumlama Sistemleri (Global Position Systems, GPS) tarafından kullanılan haritalarda daha iyi sıkıştırma oranı elde etmek bir quantization yöntemi önermiştir. Bu çalışmada haritalar daha küçük bloklara bölünerek, her blok ayrı ayrı JPEG formatında sıkıştırılmıştır. Sıkıştırma işleminde önerilen "quantization" işlemi uygulanmış, işlem sonrası elde edilen sayısal değerler kanonik Huffman algoritması ile sıkıştırılarak sonuçta daha iyi bir sıkıştırma oranı elde edilmiştir [23].

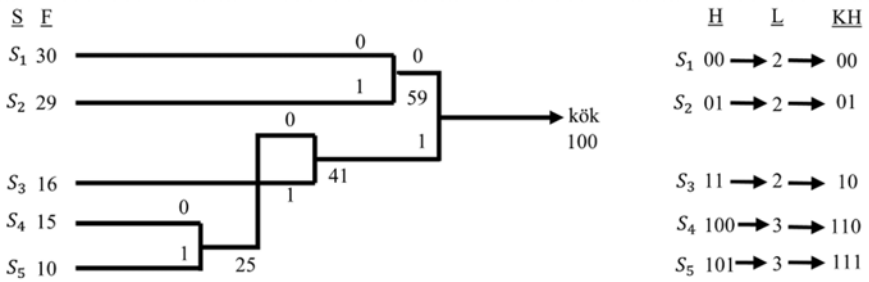
Rajiv Ranjan yaptığı çalışmada sıkıştırılacak görüntüye ayrık kosinüs dönüşümü (DWT) uygulayarak yaklaşık görüntü ve ayrıntılı görüntü olarak isimlendirilen iki alt band elde etmiştir. Alt bandlar [0,1] aralığında normalleştirilerek kanonik Huffman kodlaması ile sıkıştırılmışlardır. Kanonik Huffman kodlama burada daha küçük bir kod tablosu kullanmak ve daha az bir işlem zamanı harcamak için kullanılmıştır [24].

Zhenyu Shao vd. yaptıkları çalışmada kodlama verimini iyileştirmek ve Huffman kod kelimesi tablosunu oluşturma süreci için gereken kodlama süresini azaltmak amacıyla kanonik Huffman yöntemine dayanan yüksek verimli bir VLSI donanım mimarisi önermişlerdir. Önerilen mimari yapıyla 8 bitlik sembollerin sıkıştırılma süresinde standard Huffman algoritmasına göre %26,30 oranında iyileştirme görülmüştür [2].

3. Kanonik Huffman Kodlama (Canonical Huffman Coding)

D. Huffman tarafından tasarlanan Huffman kodlama (HK) değişken uzunluklu bir veri sıkıştırma yöntemidir [25]. Bir mesajdaki en olası sembole en kısa kodu, en az olasılıklı sembole ise en uzun kodu atar. Böylece ortalama sembol başına bit sayısını en küçük değerde tutmayı amaçlar. Ayrıca benzersiz önek özelliğine sahiptir ki, bu bir kod sözcüğünün başka bir kod sözcüğünün ön eki olmadığı anlamına gelmektedir [26]. Bu özellik sağlıklı bir kod çözme işlemi için gereklidir.

Herhangi bir mesaja ait sembollerin ağırlık tablosu verildiğinde, Huffman kodlama en az ağırlığa sahip iki sembolü birleştirerek yeni bir sembol üretir. Bu iki sembolü tablodan çıkararak birleştirilmiş sembolü tabloya ekler ve tabloyu yeniden düzenler. İteratif olarak bu işlemleri tabloda tek sembol kalıncaya kadar tekrarlar. İşlemin sonunda Huffman ağacı denilen bir ikili ağaç elde edilir. Semboller bu ağacın yapraklarında yer alır. Kökten yapraklara doğru gidilerek,



Şekil 1. Huffman ağacında kod sözcükleri ve uzunlukların belirlenmesi S: semboller F: frekanslar H: Huffman kodları L: uzunluklar KH: kanonik Huffman kodları
(determination of codewords and lengths in a Huffman tree S: symbols F: frequencies H: Huffman codes L: lengths KH: canonical Huffman codes)

her düğümden çıkan sol dal "0" ve sağ dal "1" olarak etiketlendirilerek (veya tersi), her sembolün Huffman kodu tespit edilir (Şekil 1).
Kanonik Huffman kodlama (KHK), Huffman kodlamanın bir alt kümesidir. Sembol başına ortalama bit sayısı aynı olmakla birlikte kod sözcükleri farklıdır. KHK, aynı uzunluğa sahip kod sözcüklerini gösteren ikili değerlerin ardışık olduğu anlamına gelen "Sayısal Sıra Özelliği"ne sahiptir. Örneğin; "000","001","010","011" gibi. KHK'nın HK'ya göre bazı avantajları vardır: KHK hızlı bir şekilde kodlanabilir ve kodu çözülebilir. İletim zamanı ve bellek ihtiyacı çok düşüktür. Ağaç yapısının kod çözücüye iletilmesi gerekmez, sadece kod uzunluklarının gönderilmesi yeterlidir [27, 28].

Kanonik Huffman kod sözcüklerinin elde edilebilmesi için öncelikle, verilen ağırlık tablosuna göre Huffman ağacı inşa edilir. Bu ağaçta kökten yapraklara gidilerek, her yaprağındaki sembole karşılık gelen kod sözcüklerinin uzunlukları (bit sayıları) tespit edilir. Uzunluklar elde edildikten sonra küçükten büyüğe doğru sıralanır. En küçük uzunluk değerine karşılık gelen kanonik kod sözcüğü "0" 'dır. Eğer uzunluk 1'den büyükse, uzunluk değeri kadar sağ tarafa "0" ilave edilir (sola kaydırma). Örneğin uzunluk değeri 2 ise kod sözcüğü "00" dır. Bir sonraki kod sözcüğü, önceki kod sözcüğüne ikili düzende "1" ilave edilerek bulunur. Eğer belirlenen kod sözcüğünün uzunluğu, karşı gelen uzunluk değerinden daha kısa ise fark değeri kadar kod sözcüğünün sağına "0" ilave edilir (sola kaydırma). Örneğin, uzunluk değeri 3 ise ve önceki kod sözcüğüne "1" ilave edilerek belirlenen kod sözcüğü "11" ise, "0" bit değeri kod sözcüğünün sağına ilave edilerek kod uzunluğu 3 yapılır ve yeni kod sözcüğü "110" olur [27, 28]. Şekil 1'de HK' ya ait kod sözcükleri ile birlikte KHK'ya ait kod sözcükleri de gösterilmiştir.

Huffman kodlarının performansı sembol başına ortalama bit sayısı ile ölçülür. Herhangi bir mesaj için sembol başına ortalama bit uzunluğu Eş. 1 ile tanımlanır [25].

$$A_v = \sum_{i=1}^{n} p_i \, l_i \tag{1}$$

Burada, 1≤ i ≤n ve n kullanılan alfabedeki sembol sayısı olmak üzere; l_i, i. sembolün kod uzunluğu ve p_i, i. sembolün mesaj içerisindeki kullanım sıklığıdır.

4. Cebirsel Kanonik Huffman Kodlama (Algebraic Canonical Huffman Coding)

Cebirsel Kanonik Huffman kodlama (CKHK) algoritması kanonik kod sözcüklerinin hızlı ve basit bir şekilde elde edilmesi için geliştirilmiştir. Zaman karmaşıklığı Θ(n) ve hafızada işgal ettiği yer Θ(n) kelimedir. Elde edilen kodlar genellikle en iyi değer olmayıp en iyi değere benzerlerinden çok daha yakındır.

CKHK algoritması şu şekildedir: Bir p_i ağırlığına sahip s_i sembolüne karşılık gelen kod sözcüğünün uzunluğu (bit sayısı, başka bir deyişle kökten s_i yaprağına olan derinlik) $u_i = l_i + d_j$ eşitliği ile hesap edilir. Bir Huffman ağacı göz önüne alındığında d_j kökten bir referans düğüme olan uzaklık ve l_i referans düğümden s_i'nci sembole karşılık gelen yaprağa olan uzaklıktır. Burada $1 \leq j \leq n-1$, $1 \leq i \leq n$ ve n kullanılan alfabedeki sembol sayısıdır. Başlangıçta kök olan referans düğüm, d_j değişkeninin her artışında d_j seviyesinde bulunan en sağdaki düğüm olur. Y yuvarlama fonksiyonu olmak üzere, l_i Eş. 2 ile hesaplanır:

$$l_i = \begin{cases} 1 & eğer \;\; p_i \geq 0.7 \; ise \\ Y(\log(e_i / p_i)) & aksi \; takdirde \end{cases} \tag{2}$$

Hesaplama önce kökten ve en olası sembolden başlar. Kök, bu durumda üzerinde işlem yapılan yani referans düğümdür. Burada $d_1 = 0$ ve e_1=1'dir. Referans düğümden l_i uzaklığında olması gereken yaprak ve düğüm sayısı $N_j = 2^{l_i}$ eşitliği ile verilir. Yine referans düğümden l_i uzaklığında bulunan s_i sembolünün konumu (bulunduğu seviyede sağdan itibaren kaçıncı yaprak/düğüm olduğu) $0 \leq m \leq N_j - 1$ olmak üzere, $t_m = (2^{u_i - u_{i-1}} * t_{m-1})$-1 eşitliği ile tanımlanır. s_i'ye ait l_i hesaplanırken s_{i-1}'in konumuna (t_{m-1}) bakılır. t_{m-1}'in değerine göre e_i ve d_j parametreleri Eş. 3 ve Eş. 4 ile hesaplanır.

$$e_i = \begin{cases} e_i = 1 - \sum_{x=1}^{i-1} p_x & eğer \; t_{m-1} \leq \frac{N_j}{2} \;\; ise \\ e_i = e_{i-1} & aksi \; takdirde \end{cases} \tag{3}$$

$$d_j = \begin{cases} d_j = d_{j-1} + 1 & eğer \; t_{m-1} \leq \frac{N_j}{2} \;\; ise \\ d_j = d_{j-1} & aksi \; takdirde \end{cases} \tag{4}$$

Hesaplama işlemi her sembol için yapılarak tüm uzunluklar n adımda hesaplanır. Ardından da n adımda her uzunluğa karşılık gelen kanonik kodlar belirlenir. Algoritma ilgili kaynakta detaylı bir şekilde anlatılmıştır [9].

5. Evrim Stratejileri (Evolution Strategies)

Evrim Stratejileri (ESs), verilen bir problem için bir arama uzayındaki en iyi ya da en iyiye yakın çözümleri bulmak için kullanılır. ESs, gerçek dünya problemlerinin çözümü için arama operatörleri olarak mutasyon ve elitist seçilimi kullanır. Kromozomlar, strateji parametreleri ve bireylerden oluşan ikililer ile temsil edilir. X_n, n boyutlu bir bireyi ve *SP* de strateji parametresini göstersin. Bu durumda, bir kromozom $C_n = (X_n, SP_n)$ ile gösterilir. *SP* strateji parametresi ait olduğu bireyi mutasyona uğratan parametredir. *SP* her kuşakta en uygun değere sahip olmak için evrilir. Bu özellik ESs'in en önemli özelliğidir. Araştırmacılar tarafından SP için farklı adaptasyon yöntemleri önerilmiştir. Basit bir yöntem Eş. 5 ile gösterilebilir:

$$SP = \sigma Z \tag{5}$$

σ mutasyon adım boyutudur ve mutasyonun gücünü kontrol eder. Z bir normal dağılmış rasgele vektördür ve $Z \sim N(0, \boldsymbol{I_n})$ ile ifade edilebilir. I_n, n dereceli birim matristir. Z, probleme göre farklı olasılık fonksiyonları kullanılarak tanımlanabilir.

Mutasyon önce SP'ye uygulanır. Daha sonra birey, SP vasıtasıyla mutasyona uğratılır. Mutasyonun aşamaları Eş. 6 ve Eş. 7 ile gösterilmiştir: X(t)'yi t'inci kuşağın bir bireyi olarak kabul edelim. Burada t ≥ 0'dır. $f(X(t))$ ise t. kuşakta X'in uygunluk fonksiyon değeridir.

$$SP(t+1) = \text{mutate}\,(SP(t)) \tag{6}$$

$$X(t+1) = \begin{cases} X(t) + SP(t+1) & f(X(t) + SP(t+1)) \leq f(X(t)) \\ X(t) & aksi \; takdirde \end{cases} \tag{7}$$

ESs yapısını göstermek için sembolik bir gösterim kullanılır. Bu gösterim genellikle $(\mu + \lambda)$ – ES veya (μ, λ) – ES ifadesi ile ifade edilir. μ ifadesi ebeveynlerin sayısını gösterirken λ ifadesi μ adet ebeveynden üretilen çocukların sayısını gösterir. $(\mu + \lambda)$ – ES gösterimi, en iyi μ adet bireyin, μ ebeveyn ve λ çocuk arasından seçildiğini gösterir (elitizm). μ ve λ değer aralığı şu şekildedir: $1 \leq \mu \leq \lambda < \infty$. (μ, λ) – ES gösterimi, en iyi μ adet bireyin sadece λ çocuk arasından seçildiğini gösterir. μ ve λ değer aralığı $1 \leq \mu < \lambda < \infty$'dır [29]. ES'in bazı türleri farklı gösterimlerle gösterilebilirler. Örneğin, $(\mu + \lambda)$ – SA-ES SP'nin kendinden uyarlamalı (self-adaption) olduğunu [30] ve $(\mu + \lambda)$ – CMA-ES SP'nin kovaryans matris ile adaptasyonunu gösterir [31].

6. Önerilen Algoritma (The Proposed Algorithm)

Optimuma yakın Kanonik Huffman kod sözcüklerinin uzunlukları CKHK algoritması ile elde edilir. Elde edilen bu uzunluk dizisi ES algoritmasında atayı oluşturur. Kullanılan ES algoritması (1 + λ) – CSP-ES gösterimi ile tanımlanabilir. Burada 1 sayısı tek bir ebeveyni (μ =1) yani, atayı gösterir. Ata λ sayısı kadar çoğaltılır ve mutasyona uğratılır. Sonuçta λ adet çocuk elde edilir. Mevcut ata ve çocukların içinden en iyi birey seçilir. Seçilen en iyi birey bir sonraki kuşağın atası olur. CSP (Constant Strategy Parameter), strateji parametresinin evrimleşmediğini göstermektedir. SP sabittir, çünkü Eş. 5 ile gösterilen denklemde $\sigma = 1$ ve Z rasgele vektörü elemanları sadece +1 ve -1 değerlerini almaktadır; $Z = \{z_1, z_2, \dots, z_n\}, z \in \{-1,0,1\}$. Kullanılan algoritma cinsiyetsiz çoğalma modeli (mitoz bölünme) olarak düşünülebilir: Tek ata kendini kopyalayarak çoğaltır. Ardından kopyalar mutasyona uğrar. İçlerinden en uygun değere sahip birey hayatta kalır ve sonraki kuşağın atası olur.

Süreç, sonlandırma ölçütü sağlanıncaya kadar devam eder. Algoritmanın akış şeması Şekil 2'de gösterilmiştir. Algoritmanın çıkışı Kanonik Huffman kod sözcüklerinin uzunluk dizisidir.

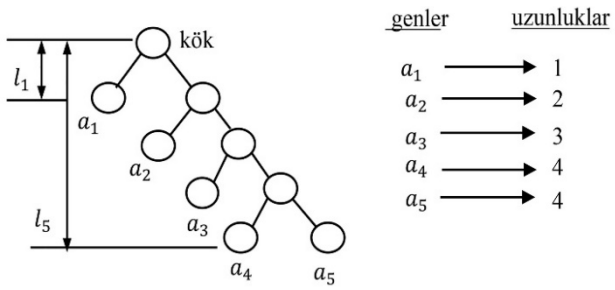
6.1. Birey (Individual)

Bir evrimsel işlemde kromozomlar EA'nın başlangıç adımıdır. ESs algoritmasında kromozomlar, $C(t) = (X(t), SP(t))$ eşitliği ile tanımlanan strateji parametreleri ve bireylerden oluşan ikililer şeklindedir. Bu ifadede $C(t)$ kromozom, $X(t)$ birey ve $SP(t)$ strateji parametreleri olup t kuşak sayısıdır. Bireyler verilen problemlerin olası çözümleridir. Burada bireyler, bir Huffman ağacındaki kod sözcüklerinin uzunluk (veya bit sayıları) dizilerini temsil eder. Bir bireydeki herhangi bir gen (eleman) Huffman ağacındaki bir yaprağa karşılık gelen kod sözcüğünün uzunluğudur.

ES-CKHK algoritması ata olarak isimlendirilen tek bir ebeveyn kullanır. İlk ata olarak da CKHK ile elde edilen uzunluk dizisi kullanılır. Bu dizi en iyi değere en yakın uzunluk dizisidir. Ata λ adet çoğaltıldıktan sonra mutasyona uğratılır. Şekil 3 örnek bir bireyi ve karşılık gelen Huffman ağacını göstermektedir.

6.2. Mutasyon (Mutation)

Mutasyon bir veya daha fazla gende meydana gelen rastlantısal değişikliklerdir. Ata λ adet kopyalanarak çoğaltıldıktan sonra tüm kopyalar mutasyona uğratılır. Böylece λ farklı çocuk elde edilir. Mutasyon, her çocuğa ait *SP* değerine 1 ilave edilerek veya çıkarılarak uygulanır. Burada $SP = \sigma Z$ ve $\sigma = 1$, Z ise rasgele bir vektördür. Z vektörü, $z \in \{-1,0,1\}$ olmak üzere $Z = \{z_1, z_2, \dots, z_n\}$ olarak tanımlanmıştır. Bu durumda mutasyon şu şekildedir: $L(t+1) = L(t) + Z(t) = \{l_1 + z_1, l_2 + z_2, l_3 + z_3, \dots, l_n + z_n\}$, n verilen bir mesajdaki sembollerin sayısıdır. Başlangıçta z sıfıra eşittir, mutasyon noktasında %50 olasılıkla +1 veya -1 değerini alır. Mutasyonun önceden belirlenmiş iki parametresi vardır: mutasyon noktalarının sayısı (MN) ve mutasyon oranı (MO). MN'ler iki şekilde seçilir: 1. Geçiş noktaları 2. Ara noktalar. Geçiş noktaları, uzunluk dizisi içindeki herhangi bir elemanın en az bir komşusunun kendinden farklı bir değere sahip olduğu noktalardır. Ara noktalar ise iki komşusunun da aynı değere sahip olduğu noktalardır (Tablo 1). Mutasyon noktaları, geçiş noktaları ve ara noktalar arasından rasgele seçilir. Bireylerin MO * λ adedi geçiş noktasından, (1-MO) * λ adedi de ara noktalardan mutasyona uğrar. Ortalama sembol başına bit uzunlukları birbirlerine yakın olan uzunluk dizilerindeki farklar çoğunlukla geçiş noktalarındadır. Bundan dolayı MO, 1'e yakın seçilir. Mutasyonun, bu şekilde problemin özelliğine göre tasarlanması "hedefe yönlendirilmiş mutasyon" olarak tanımlanabilir.

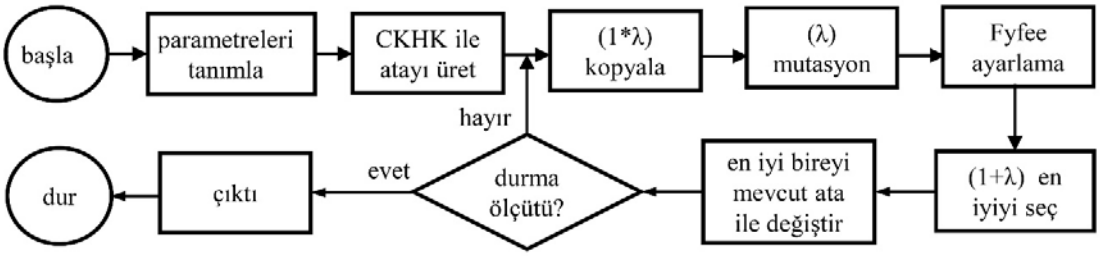


Şekil 3. Beş elemanlı örnek bir birey
(An example individual with five elements)

Tablo 1. Mutasyon Noktaları (mutation points)

P	optimal	optimale yakın	ara noktalar	geçiş noktaları
0,26087	2	2		
0,17391	2	2	X	
0,17391	3	2		X
0,13043	3	4		X
0,08696	3	4		X
0,04348	5	5		X
0,04348	5	5	X	
0,04348	5	5	X	
0,04348	5	5		
Av	2,913	2,956		

Mutasyona uğramış bireyler yapısal olarak bozulmuş olabilirler. Yani, bir Huffman ağacını temsil eden uzunluk dizisi benzersiz önek özelliğini sağlamak için Kraft-MacMillan (KM) eşitsizliğini sağlamalıdır. Bu nedenle, mutasyondan sonra bozulma ihtimaline karşılık tüm bireyler "Fyffe ayarlama" algoritması ile gözden geçirilir ve bozuk olanlar ıslah edilir. Şüphesiz yapısal bozukluğa uğrayan (KM eşitsizliğini sağlamayan) çocuklar elenip sağlam olanlar ile yola devam edilebilirdi. Ancak bu durumda çeşitlilik azalıp, evrim süresi oldukça fazla sürmekte ve en iyi değere ulaşma garanti edilememektedir.



Şekil 2. ES-CKHK algoritmasının akış şeması (Flow chart of ES-CKHK algorithm)

6.3. Fyffe Ayarlama (Fyffe Tuning)

Mutasyon işleminden sonra, uzunluk dizisine karşılık gelen ağaç yapısı bozulabilir. Bu durumda KM eşitsizliği sağlanamaz. KM eşitsizliği $\sum_{i=1}^{n} 2^{-l_i} \leq 1$ formülü ile tanımlanır. Burada n kullanılan alfabedeki sembol sayısı ve l_i i. sembole karşılık gelen kod sözcüğünün uzunluğudur (bit sayısı) [32].

Yapısı bozulan bireye Fyffe ayarlama olarak isimlendirilen düzeltme işlemi uygulanır. Fyffe ayarlama, ismini 1997 yılında Graham Fyffe tarafından geliştirilen Fyffe kodlamadan alır. Fyffe kodlama, en iyi değere yakın benzersiz önekli kodlar elde etmek için kullanılır [9]. Burada Fyffe kodlama, küçük bazı değişikliklerle yapısı bozulan bireyi ıslah etmek için kullanılmıştır. Böylece KM eşitsizliği tekrar sağlanır. Fyffe ayarlama adımları Tablo 2'de gösterilmiştir: l_i bozulan uzunluk dizisi elemanı, eşitlikler *KM*= $\sum_{i=1}^{n} 2^{-L_i}$ ve artık değer *R*=1-*KM* 'dir. Fyffe ayarlamanın amacı R değerini sıfıra eşitlemektir. İlk elemandan başlayarak R değerine bağlı olarak uzunluk değerleri 1 arttırılır veya 1 azaltılır. Eğer R<0 ise uzunluk değeri 1 arttırılır. R>0 ise 1 azaltılır. Eğer R=0 ise işlem tamamlanmış demektir.

R bir kere pozitif değere sahip olunca negatif değer almasına izin verilmez. R değerinin negatif olması Fyffe kodlama ile Fyffe ayarlama arasındaki farktır. Tablo 2'deki örneğin ilk adımında R negatiftir. Bundan dolayı ilk değer 1 arttırılır (adım 2) ve R pozitif olur. Arkasından ikinci değer (adım 3) 1 azaltılır, R negatif olur. Bu durumda işlem geri alınır ve arkasından üçüncü eleman 1 azaltılır (adım 4), R tekrar negatif olacağından üçüncü eleman da değişmez. Beşinci adımda dördüncü eleman 1 azaltılır ve R=0 olur. İşlem tamamlanır.

6.4. Seçilim ve Uygunluk Değeri (Selection and Fitness Value)

Uygunluk fonksiyonu için sembol başına ortalama bit sayısı (Eş. 1) kullanılır. Burada amaç en küçük A_v değerini bulmaktır. Seçilim aşamasında en iyi bireyi bulmak için deterministik seçilim kullanılır. Bunun için önce her bireyin uygunluk değeri hesaplanır. Ardından ata ve çocuklar arasından en küçük uygunluk değerine sahip birey seçilir. Seçilen birey bir sonraki kuşağın atasıdır. Süreç önceden belirlenmiş durma ölçütüne ulaşıncaya kadar devam eder. Yeteri kadar döngüden sonra, en iyi kod sözcüklerine karşılık gelen kod uzunlukları elde edilmiş olacaktır. Daha sonra söz konusu uzunluklardan Kanonik kod sözcükleri elde edilir ve Huffman kodlama bu kod sözcükleri kullanılarak tamamlanır.

7. Test Sonuçları ve Analiz (Test Results and Analysis)

GA ve ES-CKHK algoritmasının testi için farklı tipte dosyalar içerdiği ve herkes tarafından kolayca erişebilinmesi nedeniyle Calgary Corpus seçilmiştir (http://corpus.canterbury.ac.nz/resources/calgary.tar.gz, Son erişim: 17.03.2020). GA ile edilen sonuçlardan sadece aynı külliyatta yer alan "Bib" dosyası örnek olarak kullanılmıştır. Kullanılan GA'da bireyler rastgele oluşturulmuştur. Seçilim için rulet seçilimi kullanılmış, kullanılan alfabe boyutuna bağlı olarak çok noktalı çaprazlama kullanılarak çaprazlama noktaları rastgele seçilmiştir. Mutasyon noktaları da çoklu olup, uzunluk değerlerinin değişim gösterdiği yerlerden rasgele seçilmiştir. Her çaprazlama oranına karşılık 11 adet MO değeri için program 10 kez çalıştırılmış ve elde edilen değerlerin ortalaması Tablo 3'de italik yazı tipi ile gösterilmiştir. Tablo 3'de görülen değerler, algoritmanın başarısının çaprazlama oranından daha çok MO'ya bağlı olduğunu göstermektedir. Bu da Huffman kod uzunlukları gibi belirli kurallara bağlı ve permütasyon kodlamalı dizi uygulamalarında mutasyonun daha önemli olduğunu göstermektedir.

ESs test sonuçları Tablo 4'de gösterilmektedir. CKHK sütunu, CKHK algoritması ile elde edilen en iyi değere en yakın uzunluk dizisi (ilk ata) ile elde edilen ortalama uzunluk değerini göstermektedir. Bu ilk atanın evrimleşmesi ile en iyi ortalama uzunluk değerini (Huff_Av) üreten uzunluk dizisi elde edilmektedir. Algoritmada kullanılan parametrelerin değerleri yapılan birçok denemeden sonra en iyi sonucu verecek şekilde belirlenmiştir. Bu parametreler şunlardır: mutasyon oranı (MO), mutasyon noktalarının sayısı (MN) ve çocukların ve kopyaların sayısı (λ). Bir atadan çoğalan λ adet kopya mutasyona uğrayarak λ adet çocuk oluşturmaktadır Kullanılan alfabedeki sembol sayısı n olmak üzere, parametrelerin değerleri şu şekilde tespit edilmiştir: MO=0,95, MN=yuvarla(n/2), λ=5*n. MO=0,95; mutasyonun %95 oranında geçiş noktalarında, %5

Tablo 2. Fyffe ayarlama (Fyffe tuning)

l_i	2^{-l_i}	l_i	2^{-l_i}	l_i	2^{-l_i}	l_i	2^{-l_i}	l_i	2^{-l_i}
1	0,5	2	0,25	2	0,25	2	0,25	2	0,25
1	0,5	1	0,5	1	0,5	1	0,5	1	0,5
4	0,0625	4	0,063	3	0,125	3	0,125	3	0,125
5	0,0313	5	0,031	5	0,031	4	0,0625	3	0,125
KM=1,0938		KM=0,844		KM=0,906		KM=0,9375		KM=1	
R=-0,094		R=0,156		R=0,094		R=0,0625		R=0	
Adım 1		Adım 2		Adım 3		Adım 4		Adım 5	

Tablo 3. Bib dosyası için GA ile elde edilen ortalama döngü sayıları (Average number of loops obtained with GA for Bib file)

		Çaprazlama Oranı										
		0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Mutasyon Oranı	0	>10.000	>10.000	>10.000	>10.000	>10.000	>10.000	>10.000	>10.000	>10.000	>10.000	>10.000
	0,1	3125,6	3457,8	3721,8	3670,1	2619,7	2793	4185,8	4433,6	4853,9	4096,2	4893,3
	0,2	1313,5	2138,4	1827,6	2219,7	2023	2355,6	1628,6	2128	2837,8	2383,5	1516,4
	0,3	2010,3	1228,9	611,2	1029,4	1755,4	1617,3	1433,6	1679,4	1355,4	1026,7	1402,7
	0,4	818,6	1213,5	926,9	1568	694,9	1050,2	723,8	1190,2	1182,4	1507,75	1050,2
	0,5	578,7	719	668,5	546,2	218,8	966,3	474	754,2	1139,4	783,8	463,5
	0,6	595	689,4	1008,7	391,5	535,2	1146,1	779,4	487	655,9	692,6	694,9
	0,7	500,6	513,9	211,5	284,4	359,9	571,1	480,7	432,7	450,1	687,9	388,5
	0,8	296	416,2	261,3	254	*177,4*	307,9	600	198,9	518,1	*249,2*	553,7
	0,9	335,5	*347,6*	*161,9*	*243,1*	347,7	292,4	*139,8*	*90*	476,1	556,2	*244,7*
	1	*101,3*	326	282	449,7	232,4	*275,1*	204,9	229,6	*181,9*	268,8	391,4

Tablo 4. ESs test sonuçları (ESs test results)

Dosya	n	Boyut (Bayt)	Huff_Av	CKHK (Ata)	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Ortalama Döngü
bib	82	111.261	5,2318	5,2320	1	1	1	13	1	1	1	1	1	1	2,2
book1	83	768.771	4,5618	4,5626	30	34	5	44	38	9	48	42	47	63	36
news	99	377.109	5,2270	5,2297	3	3	2	2	5	1	2	3	12	5	3,8
paper1	96	53.161	5,0169	5,0168	2	2	1	1	2	1	1	1	2	1	1,4
paper2	92	82.199	4,6342	4,6372	9	14	9	8	7	11	11	6	7	6	8,8
progc	93	39.611	5,2339	5,2341	10	3	21	23	13	5	7	20	25	6	13,3
progp	90	49.379	4,8952	4,8966	59	5	1	34	32	1	64	69	12	1	27,8
trans	100	93.695	5,5686	5,5692	5	4	12	7	7	8	4	3	3	3	5,6
geo	257	102.400	5,6687	5,6695	10	5	4	6	5	5	5	4	5	5	5,4
obj1	257	21.504	5,9718	5,9721	5	5	7	6	6	6	6	7	6	5	5,9
obj2	257	246.814	6,2913	6.2931	9	9	8	6	9	8	7	9	8	11	8,4
pic	160	513.216	1,6609	1,6615	22	14	25	18	18	15	23	20	13	20	18,8

oranında da ara noktalarda meydana geldiğini göstermektedir. Program, her bir dosya için 10 kez çalıştırılmış ve her çalıştırmada en iyi değere ulaştığı döngü sayısı not edilmiştir.

En son olarak ortalama döngü sayısı son sütunda yazılmıştır. En iyi değere ulaşılması için gerekli olan döngü sayısının; alfabe sayısı, dosya boyutu gibi parametrelerle bir ilişkisi olmadığı görülmektedir.

Ancak, en iyi değere ulaşmak için kullanılacak döngü sayısının, alfabe sayısı olarak (n) almanın veya 100 gibi sabit bir sayı almanın yeterli olacağı görülmektedir.

Algoritmanın zaman karmaşıklığı ise şu şekildedir: Döngü içinde yer alan kopyalama fonksiyonu, n uzunluğunda 5xn adet kopya oluşturduğundan yani 5*n döngüde n kere çalıştığından $O(n^2)$ zaman karmaşıklığına sahiptir. Aynı şekilde Fyffe ayarlama ve uygunluk fonksiyonları da $O(n^2)$ zaman karmaşıklığına sahiptir. Algoritmanın genel olarak zaman karmaşıklığı ise, eğer döngü sayısı n değişkenine göre alınırsa $O(n^3)$, sabit bir sayı alınırsa $O(n^2)$ olur. Yavru sayısı, n uzunluğunda (5*n) adet olduğundan algoritmanın yer karmaşıklığı da $O(n^2)$ bayttır.

8. Simgeler (Symbols)

D = Sıkıştırılacak dosya boyutu
F = Sembol frekansları
L = Kod sözcüklerinin uzunluklarını kapsayan uzunluk dizisi
KM = Kraft-MacMillan eşitsizliği
MN = Mutasyon noktalarının sayısı
MO = Mutasyon oranı
n = Sıkıştırılacak metinde yer alan birbirinden farklı sembollerin sayısı
S = Semboller
R = Artık değer, 1-*KM*
Z = Normal dağıtılmış rasgele vektör
A_v = Sembol başına ortalama bit sayısı
C_m = Bir birey ve SP'den oluşan m boyutlu kromozom
d_j = Bir Huffman ağacında kökten referans düğüme olan uzaklık
e_i = CKHK algoritmasına göre üzerinde işlem yapılan düğümdeki olasılık değeri
$f(X(t))$ = t. kuşakta X bireyine ait uygunluk değeri
l_i = Bir Huffman ağacında referans düğümden i. yaprağa olan uzaklık
N_j = Bir Huffman ağacında herhangi bir seviyedeki düğüm sayısı
p_i = i. sembolun olasılığı
s_i = i. sembol
SP = Strateji parametresi
u_i = Bir Huffman ağacında kökten i. yaprağa olan toplam uzaklık ($l_i + d_j$)
t_m = CKHK algoritmasına göre işlem yapılan yaprağın bulunduğu seviyedeki konumu
X_m = Evrim Stratejilerinde m boyutlu birey
θ = Zaman Karmaşıklığı, ortalama süre
σ = Mutasyon adım boyutu
μ = Evrim Stratejilerinde ebeveyn sayısı
λ = Evrim Stratejilerinde çocuk sayısı

9. Sonuçlar (Conclusions)

Literatürde optimal Kanonik Huffman kodlarını geleneksel yöntemden farklı üreten bir algoritma tespit edilememiştir. Ancak optimale yakın kodları üreten algoritmalar mevcuttur. Bu makalede Kanonik Huffman kod uzunluklarını geleneksel yöntemden farklı olarak evrimsel algoritmalar yoluyla elde eden bir algoritma önerilmiştir. Veri sıkıştırma alanında yapılan önceki çalışmalardan farklı olarak kullanılan evrimsel algoritma, sabit parametreli evrim stratejileri algoritması ve mutasyon yöntemi hedefe yönlendirilmiş mutasyondur. Bu yöntemle Kanonik Huffman kodlarının üretilmesine esas teşkil edecek kod uzunlukları (bit sayıları) çok düşük bir döngü zamanında elde edilmiştir. Kanonik kodlar bu uzunluklardan elde edilir. Kanonik kodlar elde edildikten sonrada sıkıştırılacak metin bu kodlar kullanılarak sıkıştırılır. Ata olarak en iyi değere en yakın uzunluk dizisi üreten CKHK algoritması kullanılması nedeniyle, önerilen ve incelemesi yapılan bu yöntem aynı zamanda CKHK algoritmasının en iyileştirilmesi anlamına gelmektedir. Sezgisel yöntemlerin bir özelliği olarak arama uzayında çoklu birey kullanıldığından, geleneksel yöntemlere göre sonuca ulaşma hızı yavaş ($O(n^2)$) ve kullanılan hafıza miktarı ($O(n^2)$) yüksektir. Ancak sıkıştırılacak dosya boyutu yanında (D dosya boyutu olmak üzere, n<<D olması nedeniyle toplam zaman karmaşıklığı O(D) dir) ve günümüz bilgisayar teknolojisi göz önüne alındığında algoritmanın hızının bir problem teşkil etmeyeceği düşünülmektedir.

Bu çalışmada, ayrıca permütasyon kodlamalı uygulamalar için sabit parametreli evrim stratejilerinin, genetik algoritmanın yanında araştırmacılar için daha uygun bir seçenek olabileceği gösterilmiştir. İkinci bölümde bahsedilen Zaki ve Sayed'in çalışmasında çaprazlama ve mutasyon sonucu ağaç yapılarında benzersiz önek özelliği bozulmaktadır [18]. Bozulan bu bireyleri onarmak için bu makalede "Fyfee ayarlama" yöntemi önerilmiştir. Ayrıca, bu çalışmada ESs algoritmasının bilinen modellerinden farklı olarak strateji parametresi zamanla değişmeyen bir ESs modeli kullanılmıştır. Kullanılan bu yeni model "Sabit Strateji Parametreli ESs, (1 + λ) – CSP-ES" olarak isimlendirilmiştir.

Karakter tabanlı (monogram) olarak düşünülen ES-CKHK algoritması, hece tabanlı (n-gram) ve kelime tabanlı alfabeler için ilk adımdır ve kolaylıkla hece tabanlı alfabelere uyarlanabilir. Bu nedenle bir sonraki çalışma hece, kelime veya bunların karışımı olan melez alfabelerin ve kodlarının ES-CKHK ile elde edilmesi üzerine olacaktır.

Kaynaklar (References)

1. Chen Y.G. Wan, Z. Xia ve M.S. Tong, A hardware Design Method for Canonical Huffman Code, 2017 Progress in Electromagnetics Research Symposium - Fall (PIERS - FALL), Singapore, 2017.
2. Shao Z.e.A., A High Throughput VLSI Architecture Design of Canonical Huffman Encoder, IEEE Transactions on Circuits and Systems II : Express Briefs, 6 (1), 209-213, January 2022.
3. Back T., Fogel D.B., Glossary, Evolutionary Computation 1: Basic Algorithms and Operations, Bristol and Philadelphia, Institute of Physics Publishing, XXV, 2000.
4. Fogel D.B., 4: Principles of Evolutionary Process, Evolutionary Computation 1: Basic Algorithms and Operations, Bristol and Philadelphia, Institute of Physics Publishing, 23-26, 2000.
5. Demir A.S., Mert M.B.G. A new selection strategy for multi objective genetic algorithm: MultiMoora Rank, Journal of the Faculty of Engineering and Architecture of Gazi University,37 (4), 2119-2131, 2022.
6. Saraç T., Tutumlu B., A mix integer programming model and solution approach to determine the optimum, Journal of the Faculty of Engineering and Architecture of Gazi University, 37 (1), 329-345, 2022.
7. Back T., 7: Introduction to Evolutionary Algorithms, Evolutionary Computation 1: Basic Algorithms and Operations, Bristol and Philadelphia, Institute of Physics Publishing, 59-62, 2000.
8. C. Boisclair ve M. Wagner, Better Huffman Coding via Genetic Algorithm, The Proceedings of the 2008 International Conference on Genetic and Evolutionary Methods, GEM 2008, Las Vegas, 2008.
9. Oral M., Aşşık M.M., An Algorithm that Calculates the Lengths of Codewords Algebraically for Canonical Huffman-like Encoding, Cukurova University Journal of The Faculty of Engineering and Architecture, 34 (4), 9-20, 2019.
10. G. Üçoluk ve H. Toroslu, Genetic algorithm approach for verification of the syllable based text compression technique, Computer Journal of Information Science, 23 (5), 365-372, 1997.
11. F. Oroumchian, E. Darrudi, F. Taghiyareh ve N. Angoshtari, Experiments with persian text compression for web, Proceeding of the 13th international World Wide Web conference on alternate track papers & posters, WWW Alt. '04, 2004, New York, 2004.
12. J. Lánský ve T. Kuthan, Genetic algorithms in syllable based text compression, içinde Proceedings of the Dateso 2007 Annual International Workshop on DAtabases, TExts, Specifications and Objects., Desna – Cerna ˇ Rıcka, 2007.
13. A. Kattan ve R. Poli, Evolutionary lossless compression with GP-ZIP, 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computatioanal Intelligence, Hong Kong, 2008.
14. A. Kattan ve R. poli, Evolutionary Lossless Compression with GP-ZIP*, In GECCO'08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, 2008, Atlanta, 2008.
15. A. Kattan ve R. Poli, Evolutionary synthesis of lossless compression algorithms with GP-zip2, Genetic Programming and Evolvable Machines, 12 (4), 335–364, 2011.
16. A. Kattan ve R. Poli, Genetic-Programming Based Prediction of Data Compression Saving, 9th International Conference on Artificial Evolution (Evolution Artificielle), Strasbourg, France, 2009.
17. A. Kattan ve R. Poli, Evolutionary Synthesis of Losless Compression Algorithms with GP-ZIP3, IEEE Congress on Evolutionary Computation, Barcelona, 2010.
18. M. Zaki ve M. Sayed, The use of genetic programming for adaptive text compression, Int. J. Information and Coding Theory, 1 (1), 88–108, 2009.
19. K. I. M. Abuzanouneh, Develop and Design Hybrid Genetic Algorithms with Multiple Objectives in Data Compression, IJCSNS International Journal of Computer Science and Network Security, 17 (10), Seoul, 2017.
20. J. Platos ve P. Krömer, Evolving alphabet using genetic algorithms, Proceedings of the 2011 3rd World Congress on Nature and Biologically Inspired Computing, Salamanca, 2011.
21. J. Platos ve P. Krömer, Optimizing alphabet using genetic algorithms, International Conference on Intelligent Systems Design and Applications, ISDA, Cordoba, 2011.
22. J. Platos ve P. Krömer, Searching for Optimal Alphabet for Data, IEEE International Conference on Systems, Man, and Cybernetics, Seoul, Korea, 2012.
23. Y. Wiseman, JPEG Quantization Tables for GPS Maps, Automatic Control and Computer Sciences, 55 (6), 568-576, 2021.
24. R. Ranjan, Canonical Huffman Coding Based Image Compression using Wavelet, Wireless Personel Communications,117 (3), 2193-2206, 2021.
25. D. Huffman, A Method for the Construction of Minimum Redundancy Codes, Proceedings of the IRE, 40 (9), 1098-1101, 1952.
26. A. Moffat ve A. Turpin, On the implementation of minimum redundancy prefix codes, IEEE Transactions on Communications, 45 (10), 1200-1207, 1997.
27. A. Moffat, I. H. Witten ve T. C. Bell, Managing Gigabytes:Compressing and Indexing Documents and Images, Second Edition dü., E. Fox, Dü., San Francisco: Morgan Kauffmann Publishers, 1999.
28. J. B. Connell, A Huffman-Shonnon-Fano Code, Proceedings of the IEEE, 61 (7), 1046-1047, July 1973.
29. R. Günter, 9: Evolution Strategies, Evolutionary Computation 1: Basic Algorithms and Operations, Bristol, Institute of Physics Publishing, 2000, 81-88.
30. A. Auger, Convergence results for the (1,λ)-SA-ES using the theory of φ-irreducible Markov chains, Theoretical Computer Science, 334, 35-69, 2005.
31. A. Auger ve A. N. Hansen, A Restart CMA Evolution Strategy with Increasing Population Size, IEEE Congress on Evolutionary Computation, Edinburgh, 2005.
32. D. Solomon, Data Compression:The Complete Reference (4th ed.), Springer Publishing, 71, 2007.