SÜRÜ ZEKASI YÖNTEMLERİ İLE APACHE SPARK DESTEKLİ VERİ KÜMELEME

ŞÜHEDA SEMİH AÇMALI

2021
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ

Tez Danışmanı
Dr. Öğr. Üyesi Yasin ORTAKCI

SÜRÜ ZEKASI YÖNTEMLERİ İLE APACHE SPARK DESTEKLİ VERİ KÜMELEME

Şüheda Semih AÇMALI

T.C.
Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalında
Yüksek Lisans Tezi
Olarak Hazırlanmıştır

Tez Danışmanı
Dr. Öğr. Üyesi Yasin ORTAKCI

KARABÜK
Mart 2021

Şüheda Semih AÇMALI tarafından hazırlanan "SÜRÜ ZEKASI YÖNTEMLERİ İLE APACHE SPARK DESTEKLİ VERİ KÜMELEME" başlıklı bu tezin Yüksek Lisans Tezi olarak uygun olduğunu onaylarım.

Dr. Öğr. Üyesi Yasin ORTAKCI .....
Tez Danışmanı, Bilgisayar Mühendisliği Anabilim Dalı

Bu çalışma, jürimiz tarafından Oy Birliği ile Bilgisayar Mühendisliği Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir. 05/03/2021

Ünvanı, Adı SOYADI (Kurumu)	İmzası
Başkan : Dr. Öğr. Üyesi Caner ÖZCAN (KBÜ)	.....
Üye : Dr. Öğr. Üyesi Yasin ORTAKCI (KBÜ)	.....
Üye : Dr. Öğr. Üyesi Abdullah ELEN (BANÜ)	.....

KBÜ Lisansüstü Eğitim Enstitüsü Yönetim Kurulu, bu tez ile, Yüksek Lisans derecesini onamıştır.

Prof. Dr. Hasan SOLMAZ .....
Lisansüstü Eğitim Enstitüsü Müdürü

"Bu tezdeki tüm bilgilerin akademik kurallara ve etik ilkelere uygun olarak elde edildiğini ve sunulduğunu; ayrıca bu kuralların ve ilkelerin gerektirdiği şekilde, bu çalışmadan kaynaklanmayan bütün atıfları yaptığımı beyan ederim."

Şüheda Semih AÇMALI

ÖZET

Yüksek Lisans Tezi

SÜRÜ ZEKASI YÖNTEMLERİ İLE APACHE SPARK DESTEKLİ VERİ KÜMELEME

Şüheda Semih AÇMALI

Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı:
Dr. Öğr. Üyesi Yasin ORTAKCI
Mart 2021, 56 sayfa

Son yıllarda internet kullanımın artması ve her şeyin sanal ortamda saklanmasından dolayı, yüksek hacimli ve farklı türlerde (görüntü, ses, metin, sayısal değerler) veriler üretilmektedir. Bu verilerin büyük çoğunluğu etiketlenmemiş verilerden oluşmaktadır. Veri kümeleme işlemi veri madenciliğinin en önemli problemlerinden biridir. Bu problem, veri setini belirli bir sayıda farklı gruba bölen bir minimizasyon problemi olarak ele alınabilir. Bu tür minimizasyon problemlerinin çözümünde sıklıkla meta-sezgisel algoritmalar kullanılmaktadır. Bu çalışmada veri kümeleme probleminin çözümü için Armoni Arama (HS), Gri Kurt Optimizasyon (GWO) ve Yapay Alg Kolonisi (AAA) algoritmaları kullanılmıştır. Ayrıca büyük hacimli verilerin kümelemesi yapıldığı için Apache Spark teknolojisinin dağıtık hesaplama modeli işlem süresini kısaltmak için kullanılmıştır. Apache Spark mimari olarak sürücü ve işçi düğümlerden oluşur. Sürücü düğüm işlemleri dağıtmak, organize etmek ve toplama

görevlerini üstlenirken, işçi düğümler verilen işlemi yapmak ve sürücü düğüme sonuçları vermekle görevlidirler. Yapılan testler sonucunda artan düğüm sayısının işlem süresini kısalttığı görülmektedir.

Anahtar Sözcükler : Meta-Sezgisel algoritmalar, armoni arama algoritması, gri kurt optimizasyon algoritması, yapay alg algoritması, veri kümeleme, büyük veri, Apache Spark.

Bilim Kodu : 92414

ABSTRACT

M. Sc. Thesis

DATA CLUSTERING WITH SWARM INTELLIGENCE METHODS SUPPORTED APACHE SPARK

Şüheda Semih AÇMALI

Karabük University
Institute of Graduate Programs
Department of Computer Engineering

Thesis Advisor:
Assist. Prof. Dr. Üyesi Yasin ORTAKCI
March 2021, 56 pages

In recent years, high volume and different types of data (image, sound, text, numerical values, etc.) are produced due to increasing internet usage and everything stored digitally. Most of these data consists of unlabeled data. Data labeling (clustering) is one of the most important problems of data mining. This problem can be considered as a minimization problem that divides the data into a certain number of different groups. Meta-heuristic algorithms are often used to solve such minimazition problems. In this thesis, harmony search, gray wolf optimizer, and artificial algae colony algorithms are used to solve this data clustering problem. In addition, the distributed computing model of Apach Spark is used to shorten the running time because large volume data is clustered. Apache Spark architecture consists of driver and worker nodes. The driver node takes over the tasks of distibuting, organizing, and aggregating processes, while the worker nodes are tasked to perform the given process and

delivering results to the driver node. The results of the experimental studies revealed the increasing number of nodes shortens the running time.

Key Word : Meta-heuristic algorithms, harmony search algorithm, grey wolf optimization algorithm, artificial algae algorithm, data clustering, big data, Apache Spark.

Science Code : 92414

TEŞEKKÜR

Bu tez çalışmasının planlanmasında, araştırılmasında, yürütülmesinde ve oluşumunda ilgi ve desteğini esirgemeyen, engin bilgi ve tecrübelerinden yararlandığım, yönlendirme ve bilgilendirmeleriyle çalışmamı bilimsel temeller ışığında şekillendiren sayın hocam Dr. Öğr. Üyesi Yasin ORTAKCI'ya sonsuz teşekkürlerimi sunarım.

Aynı zamanda bu tez çalışmasını "FYL-2020-2225" proje numarası ile desteklemeye layık gören Karabük Üniversitesi Bilimsel Araştırma Projeleri Birimi'ne teşekkürlerimi sunarım.

Sevgili aileme manevi hiçbir yardımı esirgemeden yanımda oldukları ve özellikle kız kardeşim Mercan Şura AÇMALI'ya çizimleri için tüm kalbimle teşekkür ederim

İÇİNDEKİLER

Sayfa

Sayfa

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

KISALTMALAR DİZİNİ

KISALTMALAR

AAA : Artificial Algea Colony Algorithm (Yapay Alg Kolonisi Algoritması)

ABC : Artificial Bee Colony (Yapay Arı Kolonisi)

ACO : Ant Colony Algorithm (Karınca Kolonisi Algoritması)

AHDO : Armoni Hafıza Dikkate Alma Oranı

BG : Bant Genişliği

COA : Cuckoo Optimization Algorithm (Guguk Kuşu Optimizasyon Algoritması)

GA : Genetic Algorithm (Genetik Algoritma)

GCP : Google Cloud Platform

GWO : Grey Wolf Optimizer (Gri Kurt Optimizasyon)

HDFS : Hadoop Distributed File System (Hadoop Dağıtılmış Dosya Sistemi)

HS : Harmony Search (Armoni Arama Algoritması)

IMO : Ions Motion Algorithm (İyon Hareketi Algoritması)

PSO : Particial Swarm Optimization (Parçacık Sürü Optimizasyon)

RDD : Resilient Distributed Datasets (Esnek Dağıtılmış Veri Kümeleri)

TAO : Ton Ayarlama Oranı

TS : Tabu Search (Tabu Arama Algoritması)

WHO : Whale Optimization Algorithm (Balina Optimizasyon Algoritması)

WSA : Weighted Superposition Attraction Algorithm (Ağırlıklı Süper Pozisyon Çekim Algoritması)

WSSSE : Within Set Sum of Squared Errors (Küme İçi Kare Hatalarının Toplamı)

BÖLÜM 1

GİRİŞ

Son 20 yılda, teknolojinin ve internetin hızla gelişmesi sayesinde hem insanlar hem de teknolojik aletler birbirleriyle sürekli iletişim halinde kalmaya başladılar. Bu iletişim esnasında büyük hacimli ve farklı türlerde (ses, görüntü, video vb.) veriler elde edilmeye başlandı. Örneğin Boeing 787 tipi bir uçakta bulunan 140.000 sensörden bir uçuşta yarım terabayt veri üretilmektedir. Bu veride hava durumu, uçuş yüksekliği ve yol mesafesi gibi veriler tutularak ve işlenerek uçağın nasıl daha fazla yakıt tasarrufu yapılabileceği hesaplanmaktadır. Şekil 1.1'de görüldüğü gibi dünya çapında kullanılan sosyal medya uygulamalarında bir dakikada gerçekleşen işlemler gösterilmektedir. Bu tür uygulamaların topladıkları veriler yüksek hacimli ve farklı veri türleri içermektedir. Bu durum da artık büyük veri (big data) kavramını hayatımıza sokmaktadır. Bir verinin büyük veri sayılabilmesi için 5V olarak bilinen özellikleri içermesi gerekmektedir. Bunlar;

1. Hacim (Volume): Veri akışının sürekli veya büyük hacimli olması gerekmektedir. Veri hızı ve hacmi her geçen gün artmaktadır ve bunların işlenmesi ve bilgiye dönüştürülmesi öncelik haline gelmektedir.
2. Hız (Velocity): Veri üretim hızının fazla olması gerekmektedir. Örneğin Twitch'de 1 dakika 1.2 milyon kişi aktif olarak yayın izlemektedir ve Netflix'de 1 dakikada 764.000 saatlik izlenme olmaktadır.
3. Çeşitlilik (Variety): Var olan mevcut veriler, farklı kaynak ve yapıda oldukları için çoğunluğu yapısal değildir. Bu verileri yapısal yapabilmek için farklı veri formatlarına dönüştürülebiliyor olması ve farklı platformlara aktarılabilir olması gerekmektedir.
4. Gerçeklik (Veracity): Verilerin güvenilir ve doğru bilgi içermesi gerekmektedir. Veri içerisinde yanlış ve anlamsız kayıtların olması sağlıklı

5. sonuç alınabilmesini engellemektedir. Veri içerisinden bu tür durumların tespit edilip temizlenmesi gerekmektedir.
6. Değer (Value): Büyük verilerin en önemli bileşenide değer katmasıdır. Burada yukarıda belirtilen verilerin üretilmesi ve hazırlanması işlemlerinin yapılmasındaki tek amaç veri işlendikten sonra artı bir kazanç, değer katması gerekmektedir.

Şekil 1.1. 2020'de bir dakikada internetteki veri akışı.

Büyük verilerin işlenmesi ve analiz edilmesi günümüzün en büyük problemlerden biridir. Özellikle etiketlenmemiş büyük verilerin analiz edilmesi ve etiketlenmesi önem arz etmektedir. Veri etiketlenmesi, veri kümeleme olarak da adlandırılabilir. Veri kümeleme, veri madenciliğinin en temel ve önemli konularından biridir. Çünkü verilerin benzer özellik gösteren veri gruplarına ayrılması incelenmesi için kolaylık

sağlamaktadır. Veri kümeleme probleminin çözümü hiyerarşik veya bölümlemeli kümeleme olarak iki ayrı şekilde ele alınmaktadır. Bölümlemeli kümelemede veri istenen sayıda kümeye ayrılır. Bu ayırma işlemi iteratif olarak belirli bir ölçüt gözetilerek uygun küme merkezi konumlarını bulmayı amaçlar. Bu kümeleme yapısı bir minimizasyon problemi olarak tanımlanır.

Minimizasyon veya maksimizasyon problemlerinin çözümünde sıklıkla meta-sezgisel algoritmalar kullanılmaktadır. Meta-sezgisel algoritmaların en büyük avantajları yapısal olarak yerel minimum noktalardan kaçınma eğilimlerinin olmasıdır. Bu sayede yerel çözümler yerine genel çözüme ulaşma eğilimi göstermektedirler. Büyük veri setlerinin içerisinde çok fazla farklı nitelikleri tanımlayan değerlerin olması ve verinin boyutunun büyük olması bu problemin çözümünü zorlaştırmakta ve birçok yerel minimum noktası oluşturmaktadır. Ayrıca veri boyutunun artması işlem süresinin arttıracağından verilerin işlenmesi için dağıtık hesaplama modeline sahip büyük veri işleme teknolojisi kullanılmalıdır.

Bu tez çalışmasında, meta-sezgisel algoritmalar ile büyük veriler kümelenmiş ve işlem süresini kısaltmak amacıyla Apache Spark teknolojisi kullanılmıştır. Burada yinelemeli olarak yapılan veri setindeki her bir verinin uygun kümeye atanması ve uygunluk fonksiyonu sonucunun hesaplanması durumunun Apache Spark'ın dağıtık hesaplama modeli kullanılarak daha hızlı bir şekilde yapılmasını sağlanmıştır. Bu çalışma için Armoni Arama, Gri Kurt ve Yapay Alg meta-sezgisel algoritmaları tercih edilmiştir.

Meta-sezgisel algoritmaların, kümeleme probleminin çözümünde nasıl kullanıldığı ve Apache Spark teknolojisine nasıl entegre edildiği ileriki bölümlerde açıklanmıştır. Deneysel çalışmalardan elde edilen bulgular ve sonuçlar, son kısımda şekiller ile belirtilmiştir.

BÖLÜM 2

META-SEZGİSEL ALGORİTMALAR

Teknik olarak 1986 yılında ilk defa kullanılan meta sezgisel terimi Yunanca “meta” ve “heuristic” kelimelerinin bir araya gelmesi ile oluşmuştur ve “daha ileri sezgisel” veya “üst seviye sezgisel” şeklinde Türkçeye çevrilebilir. Üst seviye sezgisel algoritmalar, genel olarak bir arama uzayında başlangıçta rastgele ancak ilerlemesini bilinçli bir mantık ile sürdüren yöntemleri kapsamaktadır. Bu yöntemler her adımda oluşan çözüm kümesi içerinden yeni çözüm kümelere oluştururlar. Bu şekilde arama uzayı içerisinde en uygun çözümü bulmaya çalışırken yerel uygun çözümlerden de kaçınmayı sağlamaktadırlar.

Meta-sezgisel algoritmalar genel olarak optimizasyon problemlerinin çözümünde kullanılmaktadır. Bu problemler belirli bir değer aralığında belirtilen uygunluk fonksiyonunun minimizasyon veya maksimizasyon durumuna göre en uygun sonuçları elde etmeyi amaçlar.

Blum ve Roli [1]’nin yaptığı çalışmada, meta-sezgisel yöntemlerinin karakteristik yapıları şu şekilde gösterilmektedir:

1. Arama sürecine kılavuzluk eden stratejilerdir.
2. Arama uzayını etkili bir şekilde kullanarak en uygun çözümü bulmayı amaçlar.
3. Meta-sezgisel algoritmaların arama teknikleri, yerel arama metotlarından karmaşık öğrenme süreçlerine kadar geniş bir alanda çalışmaktadır.
4. Arama uzayının içinde yerel en iyi durumlarda takılıp kalmaması için bunu engelleyecek mekanizmalara sahiptirler.
5. Meta-sezgisel algoritmalar belirli bir probleme özgü değildirler. Matematiksel olarak minimizasyon veya maksimizasyon problemine olarak uygulanabilen tüm problem için kullanılabilirler.

Meta-sezgisel algoritmalar, verimli bir şekilde en uygun sonuçlara ulaşmayı amaçlayan tekrarlamalı bir yapıya sahiptirler. Bunun yapılabilmesi için gereken ilk işlem, uygulanacak problemin iyi analiz edilip en iyi şekilde algoritmaya entegre edilmesidir [2]. Meta-sezgisel algoritmalar her problem için uygun değildirler bazı özel problemlerin çözümünde bazı algoritmalar diğerlerinden daha iyi çözümler verebilmektedir [3].

Meta-sezgisel algoritmalar kullanılan yöntemler ve yaklaşımlara göre farklı sınıflara ayrılmaktadır [4]:

1. Doğadan esinlenilerek geliştirilenler – Doğadan esinlenmeden geliştirilenler
2. Popülasyon tabanlı algoritmalar – Tek nokta (yerel arama) algoritmaları
3. Dinamik amaç fonksiyonlu – Statik amaç fonksiyonlu algoritmalar
4. Tek komşu yapılı – Çok komşu yapılı algoritmalar
5. Hafıza kullanan – Hafıza kullanmayan algoritmalar

Meta-sezgisel algoritmalar, sezgisel algoritmaların doğadan esinlenilerek geliştirilmiş hali olarak görülebilir. Bu yöntemler fizik, biyoloji, matematik, zooloji vb. bilimlerden esinlenilerek geliştirilmişlerdir. Bu yöntemlerden bazıları Genetik Algoritma (GA) [5], Parçacık Sürü Optimizasyon (PSO) [6], Guguk Kuşu Optimizasyon (COA) [7] algoritmaları sırasıyla doğada gözlemlenen evrimsel süreçten, kuş sürülerinin besin arayışlarından ve guguk kuşlarının yumurtlama ve göç davranışlarından esinlenilerek geliştirilmiş algoritmalardır. Karaboğa ve Baştürk tarafında geliştirilen Yapay Arı Kolonisi (ABC) [8] algoritması yüksek boyutlu matematiksel problemlerin çözümünde kullanılmak amacıyla arı kolonilerinden esinlenilerek geliştirilmiştir. Dorigo ve arkadaşlarının [9] yaptığı çalışmada Karınca Kolonisi Optimizasyon (ACO)'nun diğer meta-sezgisel algoritmaların yapılmasına nasıl öncülük ettiğini ve meta-sezgisel algoritmalarının birçok farklı mühendislik probleminin çözümünde kullanıldığı göstermişlerdir. Simon tarafından geliştirilmiş olan Biyocoğrafya tabanlı optimizasyon [10] algoritmasında coğrafi etkinin biyolojik yaşam üzerindeki etkisinden ilham alınarak geliştirilen bir meta-sezgisel algoritmadır. Bu algoritmada diğer meta-sezgisel algoritmalar gibi yerel minimumlardan sakınmayı amaçlayarak geliştirilmiştir. Yapılan testlerde kullanılan diğer meta-sezgisel algoritmalardan daha

başarılı sonuçlar vermiştir. Ayrıca gerçek hayat problemlerinden biri olan uçak motoru sağlığı tahmini için sensör seçimi probleminde de uygulanmıştır. Erol ve Eksin [11] tarafında geliştirilen Büyük Patlama – Büyük Çarpışma (BB-BC) algoritması da evren teoremlerinden biri olan büyük patlama ve büyük çarpışma teoremlerinden ilham almıştır. Burada büyük patlamada meta-sezgisel algoritmaların temeli olan rasgelelik konusu işlenmiştir. Büyük çarpışma kısmında ise en uygun sonuçların bulunması yani uygunluk fonksiyonu çıktılarının sıralanması aşaması simüle edilmiştir. Hatamlou [12] tarafından geliştirilen kara delik algoritması ise bir yıldızın ölmesi sonucu oluşan kara delikten esinlenerek geliştirilmiştir. Bu algoritmada her bir birey yıldız olarak adlandırılır ve en uygun sonucu veren yıldız kara delik olarak seçilir ve diğer yıldızları kendine doğru çekerek onları yerel uygun sonuçlardan uzaklaştırarak genel en uygun sonuca yönlendirmeyi amaçlar. Mucherino ve Şeref [13] tarafından geliştirilen maymun arama algoritmasında bir ağaç üzerinde yukarı ve aşağıya gezen bir maymunun besin olan dalları işaretlemesi ve en fazla besin olan dalı seçmesi üzerine tasarlanan bir algoritmadır. Bu algoritmanın küresel optimizasyon problemlerinden Lennard-Jones ve Morse problemlerine uygulandığında diğer algoritmalarla rekabet edebileceği gösterilmiştir. Mirjalili ve Lewis [14] tarafından geliştirilen WOA, balinaların kabarcık çıkartarak balık sürülerini belirli bir alana toplanmasını sağlamak ve onları avlamak için kullandıkları yöntemden esinlenerek geliştirilmiştir.

Yapılan araştırmalar sonucunda biyoloji ve fizik temelli olan birçok olaydan esinlenerek, mühendislik problemlerini çözmek için kullanılan birçok algoritma geliştirilmiştir. Bu tez çalışmasında bir orkestranın en iyi armoniyi bulmasından esinlenerek geliştirilen armoni arama algoritması, kurt sürüsünün avlarını yakalamak için kullandıkları yöntemden esinlenerek geliştirilen gri kurt optimizasyon algoritması ve alg kolonilerinin yaşamlarını sürdürmek için yaptıkları yolculukların matematiksel modellenmesi sonucu elde edilen yapay alg kolonisi algoritması kullanılmıştır.

2.1. ARMONİ ARAMA ALGORİTMASI

Bu algoritma en iyi armoniyi oluşturmak isteyen müzisyenlerden esinlenilerek geliştirilmiştir [15]. Armoni arama ile optimizasyon problemleri arasında benzerlikler vardır. Her müzik enstrümanı bir karar değişkenine karşılık gelmektedir; her bir nota

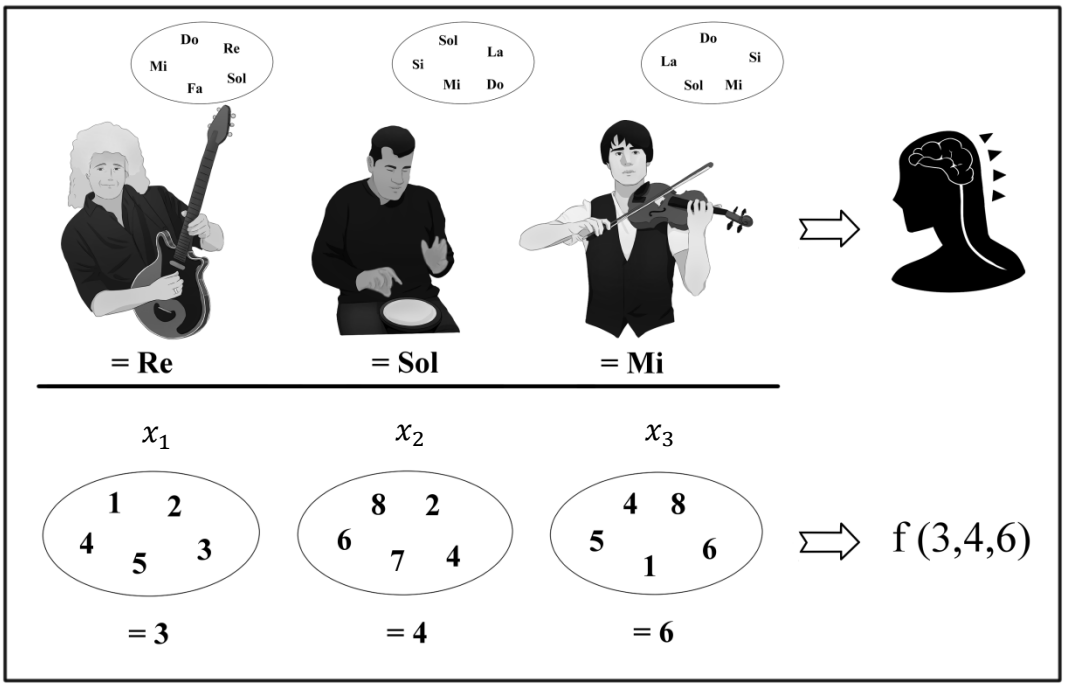
ise karar değişkeninin değerine karşılık gelir; notalar arasındaki armoni ise çözüm vektörüne karşılık gelir. Aynı bir orkestrada müzisyenlerin deneyimlerinden faydalanılarak veya rastgele notalar çalarak en iyi armoniyi oluşturma çabaları HS'ye ilham olmuştur. HS'de rastgele oluşturulan veya önceden bilenen ve hafızadaki çözümler kullanılarak, çözüm uzayındaki en uygun sonucu bulmayı amaçlar. Bir orkestra nasıl kulağa en güzel gelen armoniyi bulmaya çalışırken sürekli denemeler ve çalışmalar yapılıyorsa HS'de bu denemeler tekrar tekrar yapılarak en uygun çözüme ulaşmaya çalışır. Her deneme de müzisyenler parçanın tonunu, ritmini ve hızını değiştirler ve yeniden deneme yaparlar. Orkestradaki her bir sanatçı kendi enstrümanı için en uygun melodiyi aramaktadır. Optimizasyon problemleri için ise bu durum belirli bir çözüm uzayında problemin alabileceği en iyi durum olarak ele alınır. Bu çözüme ulaşabilmek için her bir iterasyonda üretilen sonuçlar arasından en iyisine bakılır. Armoni arama işlemi de optimizasyon problemleri gibi en uygun ve iyi sonuca ulaşmayı amaçlamaktadır. HS yerel en iyi duruma takılmadan genel en iyi duruma hızlı ve etkili bir şekilde ulaşabilen, parametreli bir algoritmadır [16].

Şekil 2.1'de HS'nin temel yapısı gösterilmiştir. Algoritmanın anlatımında üç farklı enstrüman çalan sanatçı göz önüne alınmıştır. Her sanatçının hafızasında belli armoni örnekleri ve bu armonilerin ton ayarları olduğu bilinmektedir. Sanatçılar bunları kullanarak en iyi orkestra performansını vermeyi amaçlarlar. Burada bu en iyiyi bulma süreci bir optimizasyon problemi olarak ele alındığında bu yaratım sürecinin optimizasyon problemlerinin çözümde de kullanılabileceği üzerine çalışılarak geliştirilmiştir.

2.1.1. Standart Armoni Arama

HS armonileri kullanarak işlem yapar. Bu armoniler çözüm uzayı boyutunda vektörlerden oluşmaktadır ve bu vektörler bütününe armoni hafıza (AH) adı verilir. Ayrıca HS parametrelerinde belirlenmesi gerekmektedir. Bunlar armoni hafıza büyüklüğü (N), armoni hafıza dikkate alma oranı ($AHDO$), ton ayarlama oranı (TAO), bant genişliği (BG) ve durdurma kriteri belirlenmelidir. AH tanımlanırken uygunluk fonksiyonun çözüm uzayı genişliği ($k = 1, 2, 3, \dots, D$) ve çözüm uzayının üst ve alt

sınırları belirlenir. N adet armoniden ve D boyutlu çözüm uzayından oluşan AH Denklem 2.1'de verilmiştir.



Şekil 2.1. Armoni ile optimizasyon arasındaki ilişki.

$$AH = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^D \\ x_2^1 & x_2^2 & \cdots & x_2^D \\ \vdots & \vdots & \vdots & \vdots \\ x_N^1 & x_N^2 & \cdots & x_N^D \end{bmatrix} \tag{2.1}$$

Temel olarak algoritmanın ilerleyişi şu şekildedir. Her iterasyonda AH'daki her bir armoninin uygunluk değeri hesaplanır. Buradan en iyi uygunluk değerine ve en kötü uygunluk değerine sahip armoniler belirlenir. Ayrıca her iterasyonda sadece tek bir yeni bir armoni vektörü $x^{yeni} = \left(x_1^{yeni}, x_2^{yeni}, \ldots, x_D^{yeni}\right)$ oluşturulur. Bu vektörün her bir elemanı (x_k) iki farklı şekilde belirlebilir. Bu adımlar x_k'nin AH'daki elemanlardan üretilmesi veya rastgele üretilmesidir. Bu iki adımdan birini seçerken $[0,1]$ arasında rastgele üretilmiş olan $rand$ değeri ile $AHDO$ arasındaki ilişkiye bakılır. $rand, AHDO$ değerinden küçük ise x_k AH içerisinden seçilir. Değilse arama uzayı içerisinden rastgele yeni bir değer üretilir. Eğer x_k, AH'dan üretilmiş ise bu durumda

ton ayarlamasının yapılabilmesi için [0,1] arasında ikinci bir $rand$ üretilir ve TAO arasındaki ilişki ile kontrol edilir. $rand, TAO$'dan küçükse BG değerine göre ton ayarlaması yapılır. BG'nin x_k üzerindeki etkisi Denklem 2.2'de verilmiştir.

$$x_k = x_k \pm (BG * rand) \tag{2.2}$$

Bu aşamalar ile üretilen x_{yeni} armonisin uygunluk değeri hesaplanır ve AH'daki en kötü uygunluk değerini vermiş olan armoni ile karşılaştırılır. Eğer x_{yeni} 'nin uygunluk değeri en kötü sonucundan daha iyiyse en kötü sonucu veren armoninin yerini x_{yeni} armonisi alır. Bu işlemler durdurma kriteri sağlanana kadar devam eder. Bu algoritmada temel olarak AH'daki en kötü sonuçlar elenerek yerel minimumlara takılması durumunun önüne geçilmek istenmiştir. Bu şekilde algoritma yerel iyi durumlara takılmadan genel en iyi durumlara evrilmesi sağlanmıştır. HS 'nin sözde kodu Şekil 2.2'de, akış diyagramı ise Çizelge 2.1'de verilmiştir.

Çizelge 2.1. HS sözde kodu.

GİRDİ:
n : Armoni Hafıza boyutu, **D :** Arama Uzayı boyutu, $\boldsymbol{f}_{(x)}$: Uygunluk fonksiyonu, **Iter :** Maksimum İterasyon Sayısı, **AHDO :** Armoni Hafıza Dikkate alma Oranı, **TAO :** Ton Ayarlama Oranı, **BG :** Bant Genişliği
ÇIKTI:
$\boldsymbol{A_{eniyi}}$: En iyi sonucu veren armoni
1: Armoni Hafızanın(AH) oluşturulması
2: $\boldsymbol{f}_{(x)}$ fonksiyonu kullanılarak AH'nın uygunluk değerlerinin hesaplanması
3: En iyi($\boldsymbol{A_{eniyi}}$) ve en kötü($\boldsymbol{A_{enkötü}}$) uygunluk değerlerini veren armonileri belirle
4: while (t < Iter)
5: while (i ≤ D)
6: if (rand < AHDO)
7: $\boldsymbol{x^i_{yeni}}$ için AH'dan armoni seç
8: if (rand < TAO)
9: Denklem 2.2 'e göre ton ayarlaması yap

10: **end if**

11: **else**

12: x_{yeni}^{i}'yi **r**astgele oluştur

13: **end if**

14: **i = i + 1**

15: **end while**

16: x_{yeni}'yi $f_{(x)}$ fonksiyonu ile uygunluk değerini hesapla, A_{yeni}

17: **if** (A_{yeni} < $A_{enkötü}$)

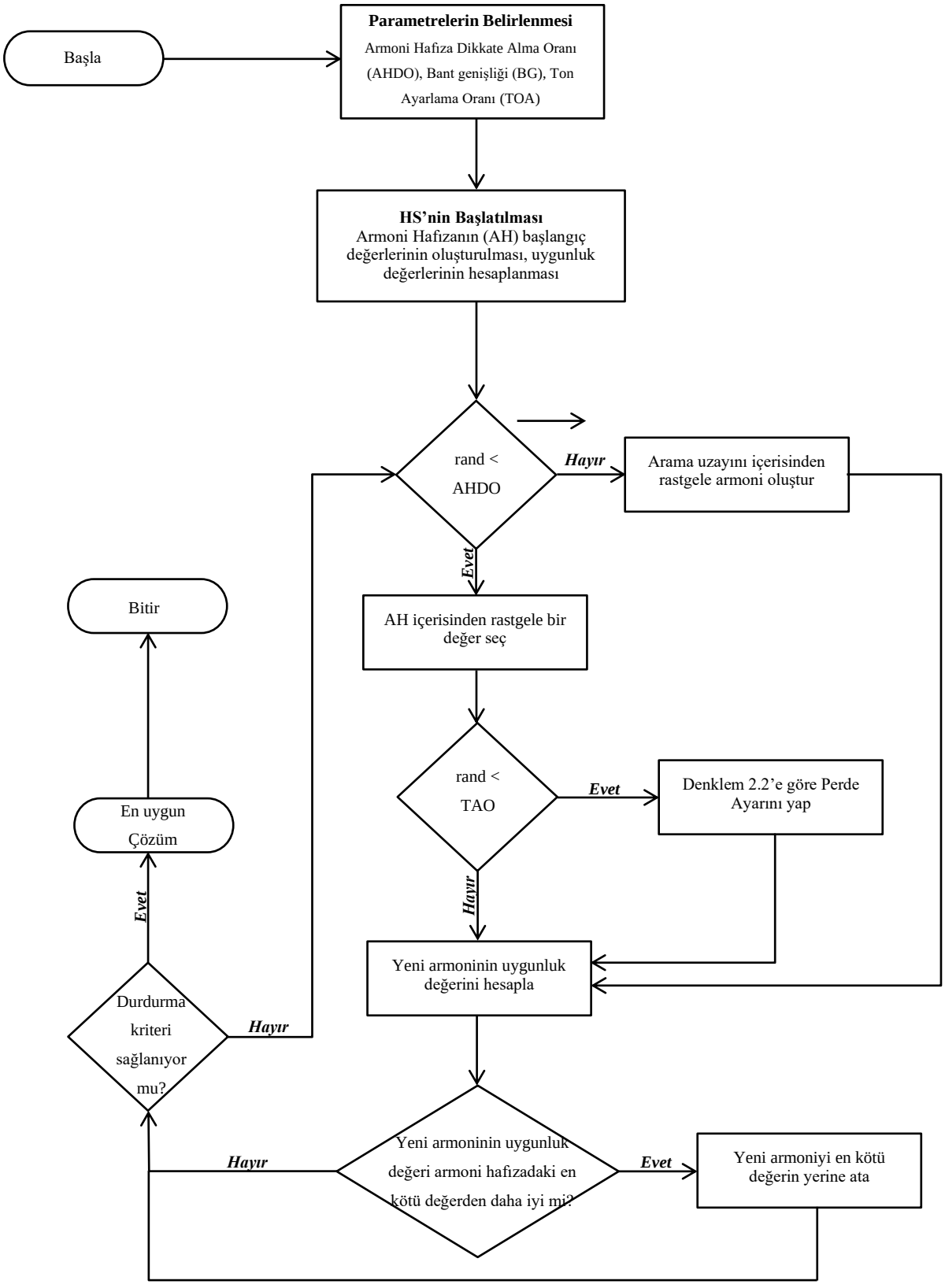
18: A_{yeni}'yi $A_{enkötü}$ yerine atamasını yap

19: AH içerisinde A_{eniyi}'yi bul

20: **t = t + 1**

21: **end while**

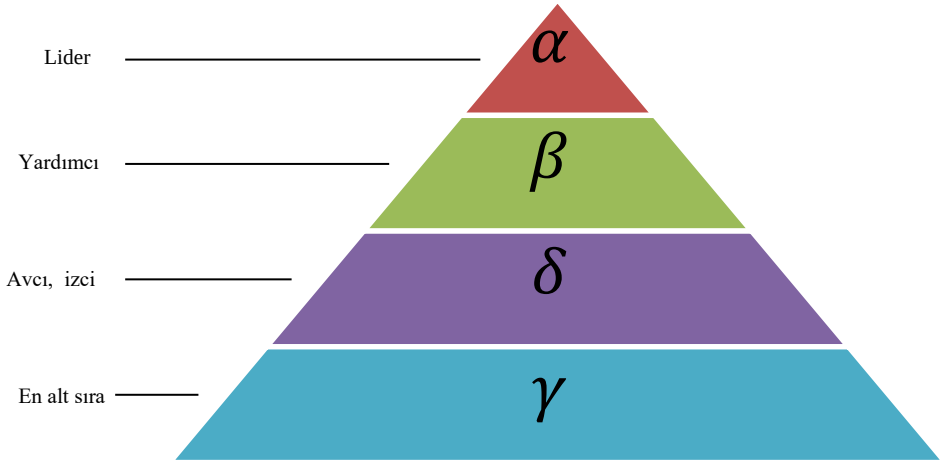
22: A_{eniyi}



Şekil 2.2. HS akış diyagramı.

2.2. GRİ KURT OPTİMİZASYON ALGORİTMASI

Son yıllarda optimizasyon problemlerinin çözümünde meta-sezgisel algoritmaların kullanımı artmıştır. Bu algoritmalar genelde doğadan ilham alınarak geliştirilen ve çözüme ulaşmak amacıyla canlıların yaşamlarını sürdürmek için yapması gereken olaylardan esinlenilmiştir. GWO Mirjalili [17] tarafından 2011 yılında gri kurtların sosyal hiyerarşik yapısından ve avlanma şekillerinden esinlenilerek geliştirilmiş bir meta-sezgisel algoritmadır. Gri kurtların sürü içerisindeki sosyal hiyerarşik yapısı Şekil 2.3'e verilmiştir.



Şekil 2.3. GWO algoritması hiyerarşik yapısı [17].

Gri kurtların hiyerarşik yapısı alfa, beta, delta, omega olarak ayrılmıştır. Alfa kurt sürünün lideridir ve tüm kurtlar üzerinde söz sahibidir. Beta kurt ise, alfaya karar alma konusunda yardımcı olmak ve yönlendirmektir. Delta kurt grubu ise, nöbetçilerden, avcılardan ve bakıcılardan oluşmaktadır ve grubun hayatta kalabilmesi için önemli bir gruptur. En düşük seviye olan gama kurt grubu, diğer gruplardaki kurtlara uymak zorunda olan hiyerarşinin en alt tabakasında bulunan gruptur.

Gri kurtlar, avlanma şekillerinde de bu hiyerarşik yapıdan yararlanmaktadır. Muro ve arkadaşlarının [18] yaptığı çalışmada bu avlanma şeklinin ana aşamaları aşağıda verilmiştir.

1. Avı izleme, takip etme ve yaklaşma

2. Avı hareket etmeyi bırakana kadar takip etme, kuşatma
3. Ava saldırmak

Bu aşamalar Şekil 2.4'de gösterilmiştir.

Şekil 2.4. Gri kurtların avlanma davranışları A) Avı kovalama, yaklaşma ve avı izlemek B) Takip etme C) Rahatsız etme D) Kuşatma E) Avı hareketsiz bırakma ve saldırma.

Bu avlanma yetenekleri ve katı hiyerarşik yapıları sayesinde uç yırtıcılar olarak hayatlarını sürdürmektedirler. Gri kurtların, bu avlanma ve hiyerarşik yapısının matematiksel modellemesi yapılarak Gri Kurt Optimizasyon (GWO) meta-sezgisel algoritması geliştirilmiştir. Bu yapılar sayesinde bir kurt sürüsünün yaşamlarını sürdürmesi için yaptıkları eylemler bir meta-sezgisel algoritmaya ilham kaynağı olmuştur.

2.2.1. Standart Gri Kurt Optimizasyon Algoritması

Gri kurtların sosyal hiyerarşik yapıları ve avlanma davranışlarının modellenmesi sonucunda sürüdeki her bir kurt (birey) çözüm uzayındaki uygun çözüm kümesini tutmaktadır. Buradaki sosyal hiyerarşik yapı uygunluk fonksiyonu sonuçların atanması

işleminde uygulanmaktadır. En uygun çözüm, alfa olarak tanımlanır. Beta ve delta ise sırasıyla ikinci ve üçüncü en uygun çözümler olarak tanımlanır. Ayrıca avlanma metotlarının modellenmesi için ise bu üç en uygun sonucu veren kurtların pozisyonları temel alınarak sürüdeki bütün kurtların pozisyonlarının güncellenmesi yapılır. Gri kurtların avlanma süreci algoritmanın temel yapılarını oluşturmaktadır. Bunlar; avlanma, avı arama, avı çevreleme ve ava saldırmadır.

2.2.1.1. Avı Çevreleme

Gri kurtların avlanma karakteristikleri temel olarak pozisyonlar güncellenmesinde kullanılmaktadır. Kurtların, avın çevresinde bulunacakları konumlar ve güncellenmeleri Denklem 2.3 ve 2.4'e gösterilmiştir.

$$\vec{D} = \left|\vec{C} * \overrightarrow{X_{p(t)}} - \overrightarrow{X_{(t)}}\right| \tag{2.3}$$

$$\overrightarrow{X_{(t+1)}} = \left|\overrightarrow{X_{p(t)}} - \vec{A} * \vec{D}\right| \tag{2.4}$$

Denklemlerde t mevcut iterasyonu, $\vec{C}$ ve $\vec{A}$ katsayı vektörlerini, $\overrightarrow{X_p}$ avın konumu vektörünü, $\overrightarrow{X_{(t)}}$ ise sürünün içerinde bulunan herhangi bir kurtun(agent) konum vektörünü göstermektedir. $\vec{C}$ ve $\vec{A}$ vektörlerinin değerleri Denklem 2.5 ve 2.6'da gösterilmiştir.

$$\vec{A} = |2\vec{a} * \overrightarrow{r_1} - \vec{a}| \tag{2.5}$$

$$\vec{C} = |2 * \overrightarrow{r_2}| \tag{2.6}$$

$$a = 2 - \left(t * \left({}^{2}/_{Iter}\right)\right) \tag{2.7}$$

Bu denklemlerde a değeri iterasyonlar boyunca 2'den 0'a doğru doğrusal olarak azalan bir bileşendir. $Iter$, maksimum iterasyon sayısı, t, mevcut iterasyonu, r_1 ve r_2 değerleri $[0,1]$ arasında rastgele olarak oluşturulan vektörleri temsil etmektedir.

2.2.1.2. Avlanma

Gri kurtlar avı kovalama ve kuşatma yetilerine sahiptirler. Av genel olarak alfa kurt tarafından yönetilir. Ancak zaman zaman beta ve delta kurtlarda av yönetimde söz sahibi olabilirler. Bu avlanma davranışından faydalanılarak geliştirilen matematiksel model de alfa (en uygun çözüm), beta ve delta sırasıyla en uygun ikinci ve üçüncü çözümleri tutmaktadırlar. Bu sebeple, sürüdeki kurtların pozisyon güncellemesinde bu üç kurdun pozisyonlarına göre güncellenerek arama uzayı içerisinde pozisyonlar değiştirilir. Bu işlemlerin matematiksel tanımlamaları için Denklem 2.8 – 2.14 kullanılır [19].

$$\overrightarrow{D_\alpha} = \left|\left(\overrightarrow{C_1} * \overrightarrow{X_\alpha}\right) - \vec{X}\right| \tag{2.8}$$

$$\overrightarrow{D_\beta} = \left|\left(\overrightarrow{C_2} * \overrightarrow{X_\beta}\right) - \vec{X}\right| \tag{2.9}$$

$$\overrightarrow{D_\delta} = \left|\left(\overrightarrow{C_1} * \overrightarrow{X_\delta}\right) - \vec{X}\right| \tag{2.10}$$

$$\overrightarrow{X_1} = \overrightarrow{X_\alpha} - \left(\vec{A}_1 * \overrightarrow{D_\alpha}\right) \tag{2.11}$$

$$\overrightarrow{X_2} = \overrightarrow{X_\beta} - \left(\vec{A}_2 * \overrightarrow{D_\beta}\right) \tag{2.12}$$

$$\overrightarrow{X_3} = \overrightarrow{X_\delta} - \left(\vec{A}_3 * \overrightarrow{D_\delta}\right) \tag{2.13}$$

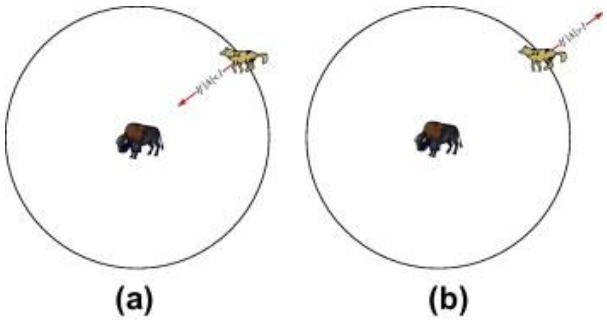
$$\vec{X}_{(t+1)} = \frac{(\vec{X}_1 + \vec{X}_2 + \vec{X}_3)}{3} \tag{2.14}$$

Denklemlerde geçen D_α, D_β ve D_δ değerleri sırasıyla alfa, beta ve delta kurtların diğer kurtlar ile arasındaki mesafeleri, X_α, X_β ve X_δ ise alfa, beta ve delta kurtlarının pozisyonlarını, X, sürüdeki herhangi bir kurdun t. iterasyondaki konumunu, A_1, A_2, A_3, C_1, C_2 ve C_3, alfa, beta ve delta kurtların Denklem 2.6 ve 2.7 ile üretilen katsayılarıdır. X_1, X_2 ve X_3, alfa, beta ve delta kurt ile üretilen güncelleme konum

vektörleridir. $\vec{X}_{(t+1)}$, sürüdeki herhangi bir kurdun (t+1). iterasyondaki konumunu göstermektedir.

2.2.1.3. Ava Saldırma

Gri kurtlar, avları hareket etmeyi kestiğinde saldırıya geçerek avlanma sürecini sonlandırırlar. Bu adımda, a değeri azalır ve avın konumu sabitlenmeye başlar. a değişkenine bağlı olan A vektörü $[-2a, 2a]$ arasında rastgele sayılardan oluşmaktadır. İterasyonlar ilerledikçe a değişkeni 2'den 0'a doğru düşmektedir. A vektörü $[-1, 1]$ arasında değer aldığında, sürüdeki kurtların bir sonraki konumları ava daha yakın bir konum alacaktır. Bu sebepten kurtlar ava saldırmaya zorlanacaktır. Şekil 2.5(a)'da $|A| < 1$ olması durumunda kurdun ava daha da yakınlaştığı gösterilmiştir.



Şekil 2.5. a) Ava saldırma b) Av arama

2.2.1.4. Avı Arama

Gri kurtlar genelde alfa, beta ve delta kurtların konumlarına göre arama yaparlar. Ancak avı ararken arama uzayında birbirlerinden uzaklaşırlar ama avı bulduklarında ve saldırıya geçecekleri zaman hemen bir araya gelme şartıyla bu durum gerçekleşir. Matematiksel modelde, bu durum $|A| < 1$ veya $|A| > 1$ olması durumuna göre değişmektedir. $|A| > 1$ olduğu durumlarda kurtlar avı aramak için birlerinden ayrılır. Bu durum Şekil 2.5(b)'de gösterilmektedir.

GWO algoritmasının sözde kodu Çizelge 2.2'de ve akış diyagramı Şekil 2.6'da gösterilmektedir.

Çizelge 2.2. GWO sözde kodu.

GİRDİ:

n : Sürü boyutu, **D :** Arama Uzayı boyutu, $\boldsymbol{f_{(x)}}$: Uygunluk fonksiyonu, **Iter :** Maksimum İterasyon Sayısı

ÇIKTI:

$\boldsymbol{X_\alpha}$: En uygun sonucu veren kurt

1: Gri kurt sürüsünün oluşturulması ($\boldsymbol{X_i}$ ($\boldsymbol{i = 1, 2, \dots, n}$)

2: $\boldsymbol{A, C\ ve\ a}$ değerlerinin tanımlanması

3: $\boldsymbol{f_{(x)}}$ fonksiyonu kullanılarak uygunluk değerlerinin hesaplanması

4: $\boldsymbol{X_\alpha}$, $\boldsymbol{X_\beta\ ve\ X_\delta}$'nin tanımlanması

5: **while (t < Iter)**

6: **for each** kurt

7: Denklem 2.15'i kullanarak pozisyon güncellemesi yap

8: **end for**

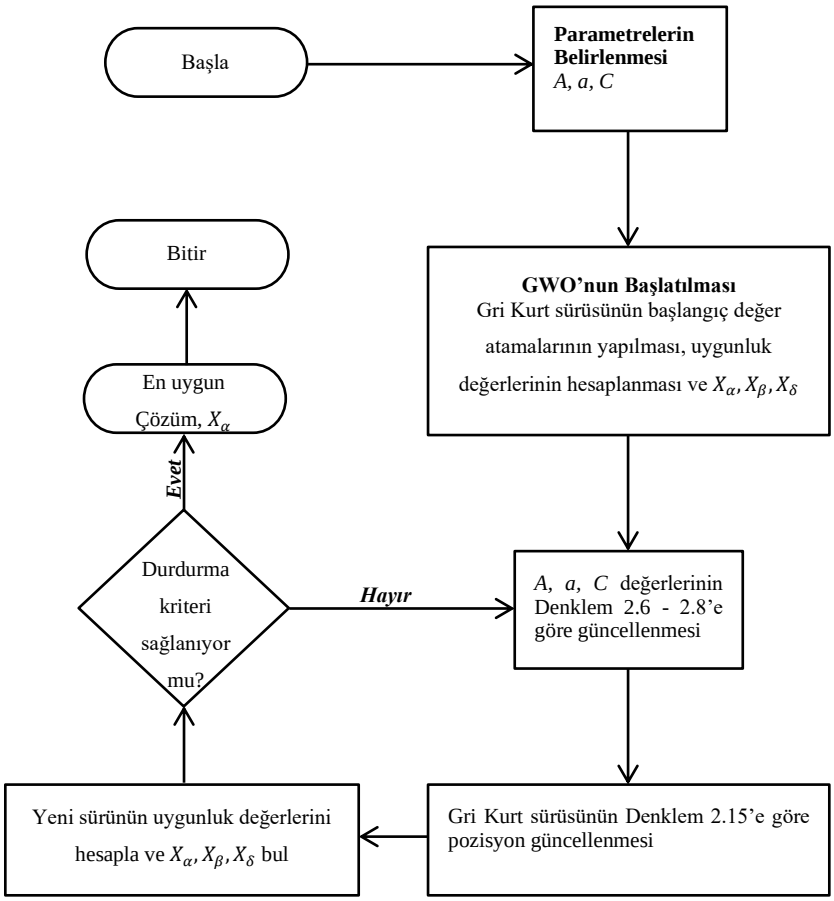
9: $\boldsymbol{A, C\ ve\ a}$ değerlerini güncelle

10: $\boldsymbol{f_{(x)}}$ fonksiyonu kullanılarak uygunluk değerlerinin güncelle

11: $\boldsymbol{X_\alpha}$, $\boldsymbol{X_\beta\ ve\ X_\delta}$'nin güncellenmesi

12: **end while**

13: $\boldsymbol{X_\alpha}$



Şekil 2.6. GWO akış diyagramı.

2.3. YAPAY ALG KOLONİSİ ALGORİTMASI

Yapay alg kolonisi (AAA), alglerin fotosentez yapabilmek için ışık kaynağına doğru hareket etmeleri ve yeterli ışık kaynağı olduğunda çoğalmalarından ilhan alınarak Uymaz ve arkadaşları [20] tarafından 2015 yılında geliştirilen bir meta-sezgisel algoritmadır.

Algler, çeşitli türleri (tek hücreli, çok hücreli vb.) olmasına rağmen çoğunluğu ototrof olan ve fotosentez yapan canlılardır. Birçok farklı ortama (deniz, tatlı su, karasal ekosistem vb.) adapte olabilen bir türdür [21]. Alglerin hayatlarını sürdürmesi ve çoğalmaları için gerekli olan materyaller ışık kaynağı, C (karbon), H (Hidrojen), O (Oksijen), P (Fosfor), N (Azot) ve eser element (metal) içermelidir [22]. Ortam

koşulları ne kadar değişkenlik gösterse bile bahsedilen bu materyaller olduğu sürece algler yaşamlarını sürdürebilir ve çoğalabilirler.

2.3.1. Standart Yapay Alg Algoritması

Algler, yaşamlarını sürdürmek için ototrof canlılar olduklarından en temelde bir ışık kaynağına ihtiyaçları vardır. Yeterli miktar ışık kaynağı olduğunda ise büyümeleri için besin kaynağına ve ortama adapte olmaları gerekir. Bu yaklaşımlardan esinlenilerek geliştirilen AAA'da gerçek hayatta olduğu gibi alg kolonilerinin ışık kaynağına doğru hareket etmesi, ortam değişikliğine adapte olması, ortamdaki dominant türü evrilmeleri gerekmektedir. Bu yaşam döngüsü prensiplerinin matematiksel modellenmesi sonucunda algoritma "Adaptasyon", "Evrimsel Süreç" ve "Helisel Hareket" olarak üç temel bölüme ayrılmıştır.

Algoritmada, alg temel türdür ve bütün popülasyon alg kolonilerinden oluşmaktadır (Denklem 2.15). Her alg kolonisi birlikte yaşayan alg hücrelerinden meydana gelirler (Denklem 2.16). Birlikte yaşayan alg kolonileri tek bir hücre gibi davranırlar. Birlikte yaşar, hareket eder ve uygun olmayan yaşam koşullarında birlikte ölürler.

$$Alg\ Kolonisi\ Popülasyonu = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^D \\ x_2^1 & x_2^2 & \cdots & x_2^D \\ \vdots & \vdots & \vdots & \vdots \\ x_N^1 & x_N^2 & \cdots & x_N^D \end{bmatrix} \tag{2.15}$$

$$x_i = [x_i^1,\ x_i^2, \dots, x_i^D], i = 1, 2, \dots, N \tag{2.16}$$

burada x_i^j, i. alg kolonisinin j. boyuttaki alg hücresini, D, çözüm uzayı boyutunu, N, popülasyon boyutunu göstermektedir. Her x_i, bir çözümü ifade etmektedir. Alg kolonilerinin, tüm hücreleri ile birlikte hareket ettiği düşünülür ve en uygun çözüme ulaştığında optimum koloni ismini alır.

2.3.1.1. Adaptasyon

Bulunduğu ortamda yeterli besin ve ışık kaynağına sahip olamayan alg kolonileri hayatta kalmak için ortama adapte olmaya çalışır ve bu süreç doğrultusunda baskın türe evrilirler. Adaptasyon sürecinde bulunduğu ortamda hayatta kalan ama yeterli besine ve ışık kaynağına ulaşamayan alg kolonilerini kendilerini en büyük alg kolonisine benzetme eğiliminde olurlar ki yeterli besin ve ışık kaynağına ulaşabilsinler. Bu durum alg kolonisinin açlık durumu ile kontrol edilir. Koloniler, helisel hareket sonucu daha iyi veya daha kötü konumlara hareket edebilirler. Daha iyi bir konuma giden kolonilerde açlık seviyesi artmaz ama daha kötü konuma giden kolonilerde açlık seviyesi artar. Tüm kolonilerin helisel hareketlerini sonuçlandırdıktan sonra kolonilerin açlık seviyeleri kontrol edilir. En yüksek açlık seviyesine sahip alg kolonisi adaptasyona uğratılır. Ancak adaptasyonun gerçekleşip gerçekleşmemesi başlangıçta belirlenen adaptasyon parametresi (A_p) ile belirlenir. $A_p, [0,1]$ arasında sabit bir sayıdır ve $[0,1]$ arasında rastgele üretilen sayı A_p değerinden küçük ise adaptasyon işlemi gerçekleşir(Denklem 2.17, 2.18).

$$starving^t = \max(A_j^t) \quad , \ (j = 1, 2, \dots, N) \tag{2.17}$$

$$starving^t = starving^t + (biggest^t - starving^t) * rand \tag{2.18}$$

burada A_j^t, t. iterasyonda j. koloninin açlık değerini, $biggest$, t. iterasyondaki en büyük alg kolonisini, $rand, [0,1]$ arasında gerçel sayıyı göstermektedir.

2.3.1.2. Evrimsel Süreç

Yapay alg hücresi, yeterli miktarda ışık aldığı zaman gelişir ve mitoz bölünme gibi kendisine benzeyen iki yeni alg hücresi oluşturur. Yapay alg kolonisinin büyüme kinetiği Monod modeli temel alınarak hesaplanmıştır (Denklem 2.19) [23]. Monod modelinde μ_i^t, t. iterasyonda i. koloninin spesifik büyüme hızı, μ_{max}, maksimum spesifik büyüme hızı, μ_{max} 1 olarak kabul edilir. $f_{(x_i)}^t$, t. iterasyonda x_i. koloninin uygunluk fonksiyonu değeri sonucunu, G_i^t, t. iterasyon i. koloninin büyüklüğünü

gösterir. Monod modeli ile $t+1$ anındaki koloninin büyüklüğü Denklem 2.20 ile hesaplanır.

$$\mu_i^t = \left(\frac{\mu_{max} * f_{(x_i)}^t}{(\frac{G_i^t}{2}) + f_{(x_i)}^t} \right) \tag{2.19}$$

$$G_i^{t+1} = G_i^t + \mu_i^t * G_i^t \quad , i = 1, 2, \dots, N \tag{2.20}$$

Başlangıçta her bir alg kolonisinin büyüklüğü 1 olarak tanımlanır. Helisel hareket sonucunda daha uygun konumlara giden kolonilerin besin kaynağı artacağından daha fazla büyürler. Tüm kolonilerin çevrimleri bittikten sonra en küçük koloninin ölen bir hücresi yerine en büyük koloninin bir hücresi kopyalanır. Bu süreç Denklem 2.21 – 2.23'de gösterilmiştir.

$$biggest^t = \max\left(size(x_i^t)\right) \quad , i = 1, 2, \dots, N \tag{2.21}$$

$$smallest^t = \min\left(size(x_i^t)\right) \quad , i = 1, 2, \dots, N \tag{2.22}$$

$$smallest_m^{t+1} = biggest_m^t \quad , m = 1, 2, \dots, D \tag{2.23}$$

burada D, çözüm uzayı boyutunu, $biggest$, en büyük alg kolonisini ve $smallest$, en küçük alg kolonisini göstermektedir.

2.3.1.3. Helisel Hareket

Alg kolonileri genellikle suda yaşarlar ve su yüzeyine yakın olmaya çalışırlar. Çünkü hayatlarını devam ettirebilmeleri için fotosentez yapmaları gerekmektedir. Bundan dolayı ışık kaynağına ihtiyaçları vardır. Su içerisinde hareket edebilmelerini sağlamak için kamçıları vardır. Kamçıları sayesinde su içerisinde 3 boyutlu olarak hareket ederler. AAA'da da bu 3 boyutlu hareket matematiksel olarak modellenerek yapay alg kolonilerinin en uygun konuma gitmeleri için gereken konum değişikliklerinin hesaplanmasında kullanılmıştır. Tek boyutlu problemlerde sadece Denklem 2.24

kullanılabilir. İki boyutlu problemlerde Denklem 2.24 ve 2.25 kullanılır. Üç veya daha yüksek boyutlu problemlerde ise Denklem 2.24 – 2.26 kullanılır.

$$x_{jm}^{t+1} = x_{jm}^{t} + \left(x_{im}^{t} - x_{jm}^{t}\right) * \left(\Delta - \tau\left(x_j\right)\right) * p \tag{2.24}$$

$$x_{jk}^{t+1} = x_{jk}^{t} + \left(x_{ik}^{t} - x_{jk}^{t}\right) * \left(\Delta - \tau\left(x_j\right)\right) * \cos a \tag{2.25}$$

$$x_{jm}^{t+1} = x_{jl}^{t} + \left(x_{il}^{t} - x_{jl}^{t}\right) * \left(\Delta - \tau\left(x_j\right)\right) * \sin b \tag{2.26}$$

$$\tau(x) = 2 * \pi * r^2 \tag{2.27}$$

$$\tau\left(x_j\right) = 2 * \pi * \left(\sqrt[3]{\frac{3G_i}{4\pi}}\right)^2 \tag{2.28}$$

burada m, k, l, D boyutlu çözüm uzayında rastgele seçilmiş üç tam sayı, x_{jm}^{t}, x_{jk}^{t}, x_{jl}^{t}, t. iterasyonda j. kolonin rastgele seçilmiş üç hücresi, x_i^t, t. iterasyonda turnuva metodu ile seçilmiş komşu alg kolonisi, Δ, başlangıç belirlenmiş olan kesme kuvvetini, $\tau\left(x_j\right)$, j. koloninin sürtünme kuvvetini, $p, [0, 1]$ arasında rastgele seçilmiş gerçel sayıyı, a, b, $[0, 2\pi]$ arasından seçilmiş gerçel derecelerdir.

Alg kolonilerinin belirli enerjileri vardır. Bu enerjiler helisel hareketi kaç kez yapacağını belirler. Her iterasyonda alg kolonilerinin büyüklükleri ile orantılı olarak (büyüklükler 0 (sıfır) ile 1 (bir) arasında normalize edilerek) alg koloni enerjileri hesaplanır. Alg kolonilerinin her bir helisel hareketi belli bir enerji harcar. Bu enerji kaybı (e) başlangıcında belirlenen bir kriterdir. Eğer bir koloni bulunduğu konumdan helisel hareket sonucunda daha uygun bir konuma gitmiş ise bu enerji kaybının yarısı kadar enerji harcar. Daha kötü bir konuma gitmesi durumunda ise enerji kaybının tamamını harcar.

AAA'nin sözde kodu Çizelge 2.3'de ve akış diyagramı Şekil 2.7'de verilmiştir.

Çizelge 2.3. AAA sözde kodu.

GİRDİ:

n : Alg kolonisi sayısı, **D :** Arama uzayı boyutu, $f_{(x)}$: Uygunluk fonksiyonu, **Iter :** Maksimum İterasyon Sayısı, Δ **:** Kesme kuvveti, e **:** Enerji kaybı, A_p **:** Adaptasyon

ÇIKTI:

A_{eniyi} **:** En uygun sonucu veren alg kolonisi

1: Alg kolonilerinin oluşturulması , x_i $(i = 1, 2, \dots, n)$

2: Kolonilerin büyüklüklerini 1(bir), açlık değerlerini 0(sıfır) olarak tanımla

3: $f_{(x)}$ fonksiyonu kullanılarak uygunluk değerlerinin hesaplanması

4: A_{eniyi} belirle

5: while(t < Iter)

6: Alg kolonilerinin enerjilerini(E) ve sürtünme yüzeylerini(τ) hesapla

7: for i = 1:n

8: **iStarve = 1**

9: **while($E_{(i)} > 0$)**

10: Turnuva metodunu kullanarak j alg kolonisini seç

11: Rastgele k, l **ve** m boyutlarını seç

12: Denklem 2.25 – 2.27 'i kullanılarak yeni alg kolonisi oluştur

13: $f_{(x)}$ kullanarak yeni koloninin uygunluk değerini hesapla

14: **if($f_{(x_{yeni})} <$ $f_{(x_i)}$)**

15: $x_i = x_{yeni}$

16: $E_{(i)} = E_{(i)} - (\frac{e}{2})$

17: **iStarve =0**

18: **else**

19: $E_{(i)} = E_{(i)} - e$ **end if**

20: **end while**

20: **if(iStarve = 1)** açlık değerinin bir arttır **end if**

21: end for

22: Alg kolonilerinin büyüklüklerini (G) hesapla

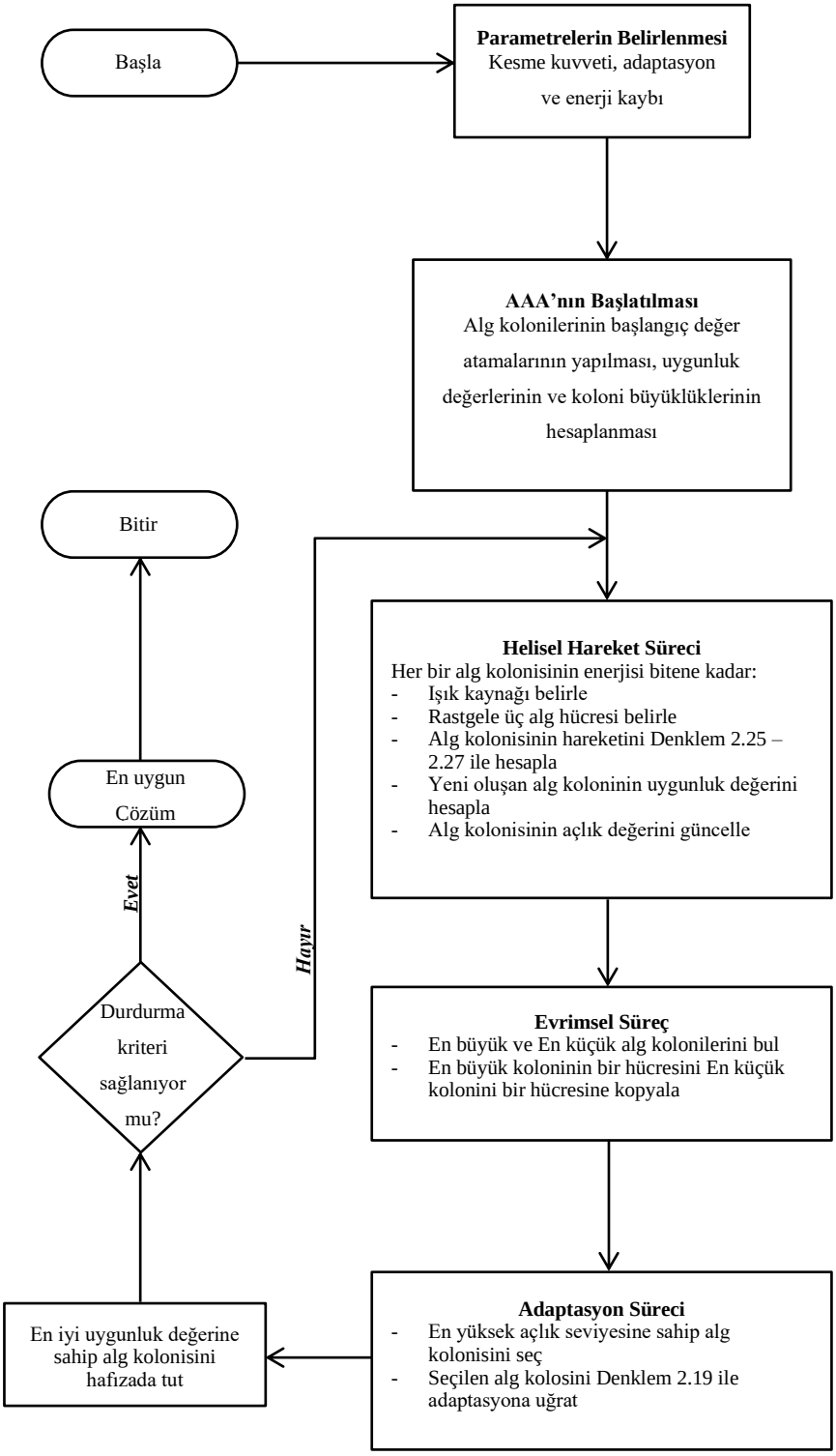
23: Rastgele belirlenmiş bir boyuta Denklem 2.24'u uygula

24: if($rand < A_p$)

25: En aç alg kolonisine Denklem 2.19'u uygula **end if**

26: A_{eniyi}'yi bul

27: **end while**

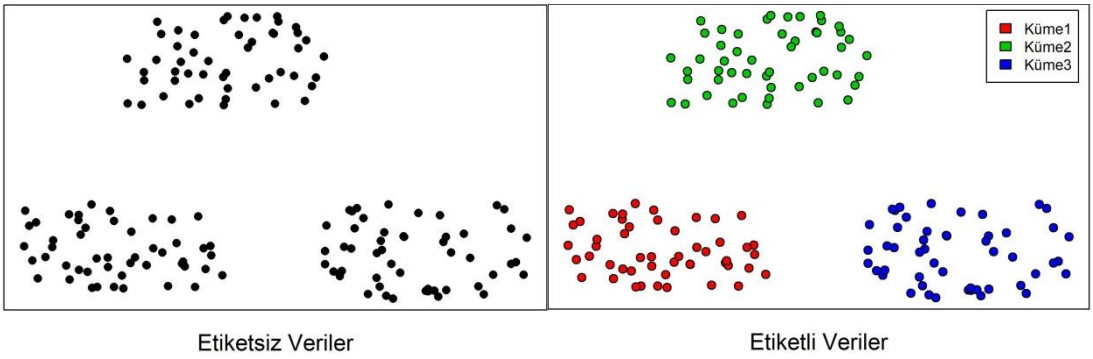


Şekil 2.7. AAA akış diyagramı.

BÖLÜM 3

KÜMELEME

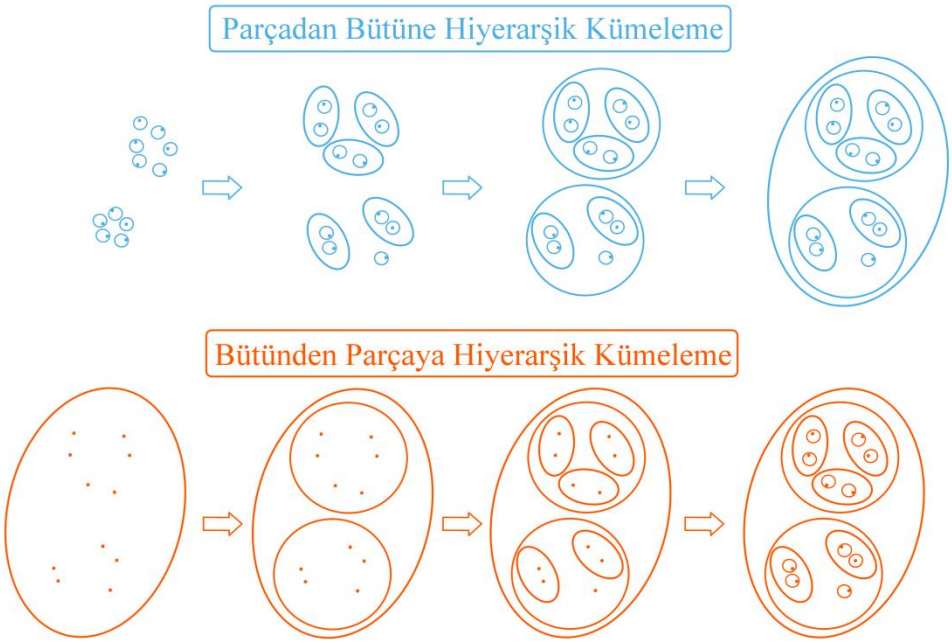
Kümeleme, hiçbir etikete sahip olmayan verilerin bir öğretici olmadan gruplandırılması işlemedir [24]. Bu gruplandırmanın temel amacı, oluşturulan grupların kendi içerisinde olabildiğince benzer olması, diğer gruplarla ise benzer olmaması gerekmektedir. Kümeleme, veri madenciliğinin temel problemlerinden birisi ve istatiksel veri analizinde kullanılan bir yöntemdir [25]. Veri madenciliğinin yanı sıra, makine öğrenmesi, görüntü analizi, veri sıkıştırma vb. alanlarda yaygın olarak kullanılmaktadır. Kümelemenin temel amacı, n elemanlı bir veri setindeki verileri k tane kümeye ayırmaktır. Bu kümeleme probleminin yapılabilmesi için iki farklı yaklaşım kullanılır. Bu yaklaşım modelleri, hiyerarşik ve bölümsel kümeleme modelleridir. Şekil 3.1'de etiketlenmemiş verilerden oluşan bir veri setinin küme işlemi gösterilmiştir.



Şekil 3.1. Veri kümeleme işlemi.

Hiyerarşik kümeleme, parçadan bütüne (agglomerative) ve bütünden parçaya (divisive) olmak üzere iki farklı yaklaşımla kümeleme işlemini yapar. Şekil 3.2'de de gösterildiği gibi bunlar adlarından anlaşılacağı üzere, veri setindeki bütün veri parçalarının hepsini ya bir bütün olarak alır ve benzer özellikte olmayanları ayırarak

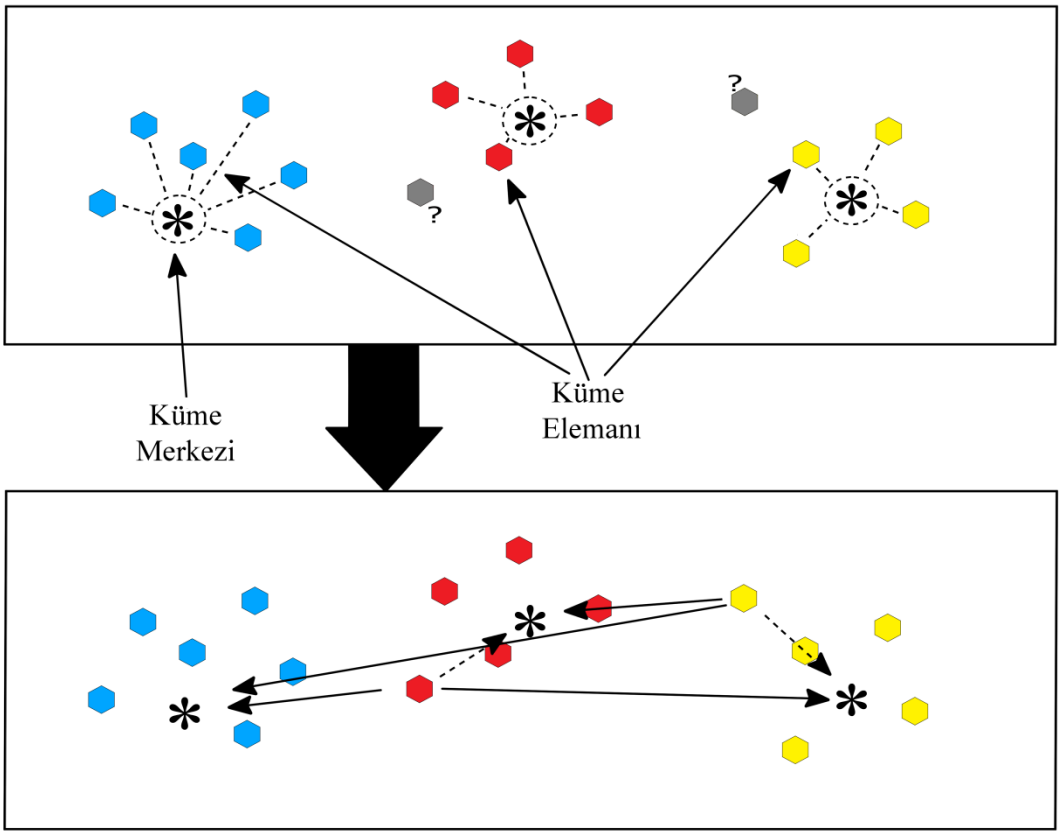
kümeleme yapar ya da ayrı kümeler olarak alır ve benzer özellikte olanları birleştirerek kümeleme işlemini gerçekleştirir [26]. Bu kümeleme yönteminin en önemli avantajı küme sayısının önceden bilinmesine gerek duymamasıdır. Dezavantajı ise bir kümeye atanan bir elemanın sonradan kümesinin değiştirilememesi ve iç içe girmiş veri kümelerinde ayrım yapamamasıdır.



Şekil 3.2. Hiyerarşik kümeleme çeşitleri.

Bölümsel kümelemede ise veri seti birbirinden bağımsız istenilen sayıda kümeye bölünebilir. Bu işlemi yinelemeli bir şekilde belirlenen bir ölçüte göre (örneğin küme içi kare hatalarının toplamı) yapar ve uygun küme merkezlerini bulmayı amaçlar. Bu yinelemeli ve bir ölçüte göre yapılan işlem, kümeleme problemini optimizasyon problemine dönüştürür [27].

Bölümsel kümelemede, veri setindeki verilerin ait olacakları kümeyi belirlemede iki farklı yaklaşımla yapılmaktadır. Bunlar bulanık (fuzzy) ve kesin (crisp) yöntemleridir. Bulanık kümeleme yaklaşımında, veri belirlenen bulanıklık derecesine göre birden fazla kümeye ait olabilir. Kesin kümeleme yaklaşımında ise veri ancak bir kümeye ait olabilir. Bu iki yaklaşımda da her kümede en az bir eleman olmak zorundadır [28, 29]. Şekil 3.3'te veri seti içerisindeki iki elemanın uygun kümeye atanma işlemi gösterilmiştir.



Şekil 3.3. Kümeye eleman atama işlemi.

Kümeleme problemlerinin çözümünde birçok farklı algoritma kullanılabilir. Bunlardan K-Means [30, 31], Fuzzy C-Means [32, 33] gibi geleneksel algoritmalar çok fazla şekilde kullanılmasına rağmen genel optimum küme merkezlerini bulma sürecinde yerel optimum küme merkezlerinde takılma riski yüksektir. Son yıllarda bu sorunun üzerinden gelmek için sıklıkla meta-sezgisel algoritmalar kullanılmaya başlanmıştır.

Maulik ve Bandyopadhyay [34] GA'nin kümeleme problemlerinde uygulanabilir olduğu göstermek amacıyla yapay olarak ürettikleri 4 veri seti ile genel olarak kullanılan İris, Hint Telugu ünlü sesi ve Ham petrol veri setleri üzerinde K-means algoritması ile karşılaştırmışlardır. Bu karşılaştırmalarda GA tüm veri setlerinde kümeleme başarımı olarak K-Means'den daha başarı olduğu gösterilmiştir. Chen ve Ye'nin [35] yapmış olduğu PSO ile kümeleme çalışmasında, yapay olarak üretilmiş üç farklı veri setini K-Means ve Fuzzy C-Means algoritmaları ile karşılaştırılmış ve kümeleme başarımı olarak iki algoritmayı da geride bırakmıştır. Shelokar ve

arkadaşlarının [36] yapmış olduğu çalışmada ACO ile kümeleme probleminin çözülebileceği gösterilmiştir. 5 farklı veri setinde yapılan testler sonucunda ACO algoritması GA, Tabu Arama (TS) ve benzetilmiş tavlama algoritmalarına göre başarılı sonuçlar vermiştir. Verma ve arkadaşlarının[37] yaptığı çalışmada Fuzzy C-Means algoritması ile PSO algoritmasını birleştirerek geliştirdikleri algoritmayı beyin görüntülerinin bölümlenmesinde kullanmıştır ve bu algoritmaların tek başlarına kullanıldığında elde ettikleri başarımdan daha üstün bir başarım elde etmişlerdir. Kaushik ve arkadaşlarının [38] yapmış olduğu çalışmada ateş böceği algoritmasına, genetik algoritmanın çaprazlama modelini entegre ederek kümeleme başarımının arttırıldığını göstermişlerdir. Ortakcı'nın [39] yapmış olduğu çalışmada PSO ile Fuzzy C-Means algoritmasını birleştirerek normalde bilinmesi gereken ayrılacak küme sayınının belirlemesi işleminde dinamik olarak hesaplanmasını sağlamıştır. Hatamlou [12] tarafından yapılan çalışmada BH algoritmasının kümeleme başarımının K-Means, PSO, Yerçekimsel Arama (GSA), BB-BH algoritmalarına göre daha başarılı olduğu gösterilmiştir. Deeb ve arkadaşları [40] tarafından değiştirilen Karadelik (BH) algoritmasında kara delik tarafından yutulan yıldızların yeniden oluşturma işleminde yenilikler yapılmıştır. Bu yapılan yenilik ile BH algoritmasının kümeleme başarımını kontrol etmek için sıklıkla kullanılan beş veri seti kullanılmış ve birçok algoritma ile karşılaştırılmıştır. Bu karşılaştırmalar sonucunda BH algoritması tüm veri setlerinde tüm algoritmalar içerisinde standart BH algoritmasında dahil en iyi sonuçları verdiği gösterilmiştir. Kushwaha ve arkadaşları [41] tarafından manyetik güçten ilham alınarak geliştirilen manyetik optimizasyon algoritması kümeleme problemlerinin çözümünde kullanılmıştır. UCI veri setleri üzerinde yapılan test sonuçlarında saflık (purity) ve doğruluk (accuracy) değerlerine bakıldığında genel olarak diğer algoritmalardan daha iyi sonuçlar verdiği gösterilmiştir. Karaboğa ve Öztürk [42] tarafından yapılan çalışmada ABC algoritmasının kümeleme başarımı UCI veri setleri ile PSO, BayesNet, RBF, KStar gbi birçok algoritma ile karşılaştırılmıştır. Tüm veri setleri sonuçları karşılaştırıldığında diğer algoritmalar göre daha iyi sonuç verdiği gözlendiği gösterilmiştir. Kapoor ve arkadaşların [43] yapmış olduğu çalışmada uydu görüntülerinin kümelenmesinde GWO algoritması kullanılmıştır. Bu çalışma da GWO algoritması ile GA, Diferansiyel Evrim (DE) ve PSO algoritmaları ile karşılaştırıldığında Davies-Bouldin indeksi, işlem süresi, küme içi mesefa ve kümeler arası mesafe durumlarında daha iyi sonuçlar vermiştir. Ji ve diğerleri [44] tarafından

K-Modes algoritması ile birleştirilen ABC algoritmasının kategorik veri setlerinde K-Modes, Fuzzy C-Means algoritmalarına göre daha başarı olduğu gösterilmiştir. Özbakır ve Turna'nın [45] yapmış olduğu çalışmada son 10 yılda çıkmış olan meta-sezgisel algoritmaların kümeleme başarımlarının karşılaştırıldığı çalışmada UCI'nın 10 farklı veri seti üzerinde testler yapılmıştır. Bu testler sırasında PSO, ABC, İyon Hareketi (IMO), Ağırlıklı Süper Pozisyon Çekimi (WSA) algoritmaları kullanılmıştır. Sonuçlar incelendiğinde zaman içerisinde geliştirilen algoritmaların başarımlarının arttığı gözlenebilmektedir.

Bu tez çalışmasında bölümlemeli kümeleme probleminin çözümü için son 20 yıl içerisinde farklı zamanlarda geliştirilmiş üç algoritma kullanılmaya karar verilmiştir. Bu algoritmalar geliştirilme zamanlarına göre sıralanmış şekilde HS, GWO ve AAA meta-sezgisel algoritmalarıdır. Bu algoritmalar kullanılarak yöneticisiz kesin bölümlemeli kümeleme yöntemiyle büyük veri setlerinin kümelenmesi amaçlanmaktadır.

3.1. META-SEZGİSEL ALGORİTMALARIN KÜMELEME PROBLEMİNE UYGULANMASI

3.1.1. Kümeleme Probleminin Tanımı

n tane elemandan oluşan veri seti $V = \{\overrightarrow{V_1}, \overrightarrow{V_2}, \overrightarrow{V_3}, \dots, \overrightarrow{V_n}\}$ şekilde ifade edilsin. Veri setindeki her bir eleman $\vec{V_i}$, d boyutlu bir vektördür.$\vec{V_i} = \{v_{i,1}, v_{i,2}, v_{i,j}, \dots, v_{i,d}\}$ ve $v_{i,j}$, veri setindeki i. noktanın j. boyutundaki gerçel değerini ifade eder.

Kümeleme algoritmalarında veri setindeki her bir eleman k farklı kümeye ayrılır, bu kümeler $C = \{C_1, C_2, \dots, C_k\}$ olarak ifade edilsin. Bu kümeler işlem sonucunda kendi içerisinde mümkün olabildiğince benzer, her bir küme diğer kümeler ile mümkün olabildiğince farklı olmalıdır. Kümeleme işlemin sonucunda oluşan kümeler şu özellikleri taşımalıdır [46]:

1. Her kümede en az bir eleman olmalıdır. $C_i \neq \emptyset, i = \{1, 2, \dots, k\}$

2. Her eleman yalnızca bir kümeye atanmalıdır. $C_i \cap C_j = \emptyset, i \neq j \; ve \; i,j = \{1,2,\dots,k\}$
3. Her eleman bir kümeye atanmalıdır. $\bigcup_{i=1}^{k} C_k = V$

Kümeleme işleminde veri setindeki elemanlar arası benzerlik oranı genel olarak Öklid uzaklık ölçütü kullanılarak belirlenir. Örneğin $\vec{V_i}$ ve $\vec{V_j}$, veri seti içerisindeki d boyutlu iki eleman olsun. Bunların benzerlik miktarı şu şekilde hesaplanır [47]:

$$\left\|\vec{V_i}, \vec{V_j}\right\| = \sqrt{\sum_{\dim=1}^{d} (v_{i,dim} - v_{j,\dim})^2} \tag{3.1}$$

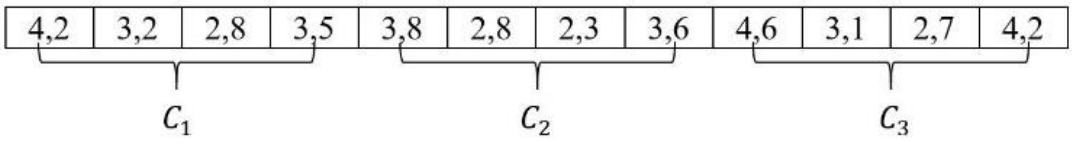
3.1.2. Meta-Sezgisel Algoritmaların Kümeleme İşlemine Uygulanması

n adet veriden oluşan bir veri setinde istenilen sayıda kümeye ayırma probleminde çok farklı şekillerde kümeler oluşturulabilir. Bu oluşan kümelerden kendi içerisinde en benzer ve diğer kümelerde en az benzer şekilde oluşacak kümeleri bulmak bir optimizasyon problemidir. Bu problemin çözümde popülasyon tabanlı meta-sezgisel algoritmaların kullanılması en uygun kümelerin oluşturulmasında kullanılabilir.

Önceki bölümde de anlatıldığı gibi bu tez çalışmasında kullanılan meta-sezgisel algoritmalar için popülasyondaki her bir birey (armoni, kurt, alg kolonisi) problemin olası bir çözümünü göstermektedir. Kümeleme probleminde ise popülasyondaki her bir birey oluşturulacak kümeler için muhtemel küme merkezlerini tutarlar. Bu merkez değerlerine göre veri setindeki bütün elemanları uygun kümelere yerleştirir ve en uygun küme merkezi noktalarını bulmayı amaçlar.

Bu tez çalışmasında kullanılan bütün meta-sezgisel algoritmalar için popülasyondaki her bir bireyin başlangıç popülasyonlarını $X = \{\vec{X}_1, \vec{X}_2, \dots, \vec{X}_i, \dots, \vec{X}_{pop}\}, i = 1,2,\dots,pop$ olarak ifade edilir. pop, popülasyondaki birey sayıdır. Kümeleme işleminde her bir birey küme merkezlerini temsil eden vektördür. Her bir birey, $\vec{X}_i = \{\vec{c}_1, \vec{c}_2, \dots, \vec{c}_j, \dots, \vec{c}_k\}, j = 1,2,\dots,k \; ve \; k,$ oluşturulacak küme sayısını, $\vec{c}_j$, j. kümenin ağırlık merkezini ifade eder. Her bir küme merkezi $\vec{c}_j = \{c_{j,1}, c_{j,2}, \dots, c_{j,d}\}$ d boyutlu

bir vektördür. Küme merkezlerinin boyutu veri setindeki elemanların boyutuyla aynıdır. Her bir boyut gerçel değerlerden oluşur. Örneğin, veri setindeki elemanlarının boyutu dört olan ve üç kümeye ayrılacak bir problemde popülasyondaki bir bireyin gösterimi Şekil 3.4'de gösterilmiştir.



Şekil 3.4. Küme merkezlerinin temsili gösterimi.

3.1.3. Kümeleme Doğruluk İndeksi

Kümeleme doğruluk indeksi (KDİ), kümeleme algoritmaları için kümeleme başarımlarını sayısal değere dönüştürmek amacıyla geliştirilmiş uygunluk fonksiyonlarıdır. Meta-sezgisel algoritmalar da temel olarak bu tür fonksiyonları minimize veya maksimize yapmayı amaçlamaktadır. Bu çalışmada da bu fonksiyonun minimize edilmesi amaçlanmaktadır. Kümeleme problemlerinin çözüme ulaşması için iki temel başarımın sağlanması gerekir:

1. Oluşturulan kümelerin içerisindeki elemanlar kendi aralarında olabildiğince birbirine yakın olmalı,
2. Oluşturulan farklı kümeler arasında olabildiğince ayrışma olmalıdır.

Literatürde kümeleme başarımı ölçmek için farklı KDİ fonksiyonları kullanılmaktadır [48, 49]. Bu tez çalışmada KDİ olarak Küme içi hata karelerinin toplamı (WSSSE) kullanılmıştır [50]. Bu KDİ fonksiyonun da amaç en küçük değere ulaşmaktır. Buda kullanılan meta-sezgisel algoritmaların minimizasyon işlemi yapacağı anlamı gelir. Popülasyondaki bir bireyin uygunluk değerini hesaplama formülü aşağıda verilmiştir:

$$WSSSE = \sum_{j=1}^{K} \sum_{i=1}^{N_j} \left\| v_{i,j} - c_j \right\|^2 \quad (3.2)$$

burada $v_{i,j}$, veri seti içerisindeki j. kümeye atanan i. elemanı, c_j, j. küme merkezini göstermektedir.

3.1.4. Meta-Sezgisel Algoritmalar ile Kümeleme Gerçekleştirilmesi

Bu tez çalışmasında kullanılacak meta-sezgisel algoritmalar ile kümeleme işleminin yapılabilmesi için yapılacak genel işlemler aşağıdaki sözde kodda verilmiştir.

Çizelge 3.1. Meta-Sezgisel algoritmalar ile kümeleme sözde kodu.

Girdi:

$\boldsymbol{V} = \{\overrightarrow{\boldsymbol{V_1}}, \overrightarrow{\boldsymbol{V_2}}, \overrightarrow{\boldsymbol{V_3}}, \ldots, \overrightarrow{\boldsymbol{V_n}}\}$: veri seti, $\boldsymbol{pop}$ **:** popülasyon sayısı, $\boldsymbol{k}$ **:** küme sayısı, $\boldsymbol{Iter}$ **:** iterasyon sayısı, $\boldsymbol{n}$ **:** veri setindeki eleman sayısı

Çıktı:

$\boldsymbol{X_{eniyi}}$ **:** en iyi sonucu veren birey

1: $\boldsymbol{pop}$ kadar bireyinin başlangıç değerlerini ata, $\boldsymbol{X} = \{\vec{\boldsymbol{X}}_{\boldsymbol{1}}, \vec{\boldsymbol{X}}_{\boldsymbol{2}}, \ldots, \vec{\boldsymbol{X}}_{\boldsymbol{i}}, \ldots, \vec{\boldsymbol{X}}_{\boldsymbol{pop}}\}$
2: **for t = 1:Iter**
3: **for each** $\boldsymbol{X}$
4: **for i =1:**$\boldsymbol{n}$
5: **for j = 1:**$\boldsymbol{k}$
6: Denklem 3.1'e göre $\|\vec{\boldsymbol{V}}_{\boldsymbol{i}}, \vec{\boldsymbol{c}}_{\boldsymbol{j}}\|$ değerini hesapla
7: **end for**
8: Eğer $\vec{\boldsymbol{c}}_{\boldsymbol{j}}$, $\vec{\boldsymbol{V}}_{\boldsymbol{i}}$'ye en yakın ağırlık merkezi ise $\vec{\boldsymbol{V}}_{\boldsymbol{i}}$'yi $\vec{\boldsymbol{c}}_{\boldsymbol{j}}$'i kümeye ata
9: **end for**
10: Denklem 3.2'e göre bireyin uygunluk değerini hesapla
11: **end for**
12: $\boldsymbol{X_{eniyi}}$'yi bul
13: Kullanılan meta-sezgisel algoritmaya göre ağırlık merkezlerini güncelle
14: end for

BÖLÜM 4

APACHE SPARK

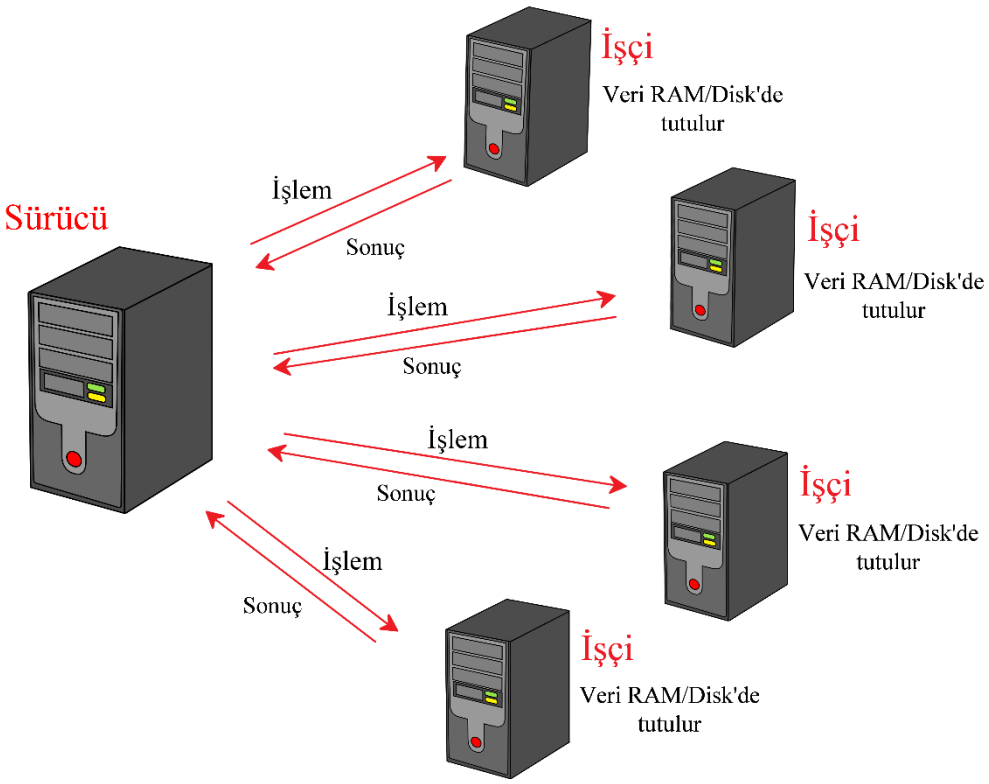
Apache Spark, yüksek veri boyutuna sahip verilerin işlenmesinde dağıtık hesaplama sistemine sahip açık kaynak kodlu bir sistemdir. Kaliforniya Üniversitesi'nden Matei Zaharia tarafından geliştirilmiştir [51]. Spark, hesaplama yapacağı verileri önbellekte tuttuğu ve optimize edilmiş sorgu yöntemleri sayesinde daha önceden geliştirilmiş olan Apache Hadoop gibi sistemlere göre çok daha hızlı işlem yapmaktadır. Spark ayrıca R, Python, Java ve Scala gibi programlama dillerine doğrudan destek verdiği gibi üçüncü parti yazılımlar ile .NET ve Julia dillerini de desteklemektedir [52]. Spark gelişmeye ve daha geniş bir kullanım alanına sahip olmaya devam etmektedir. Sistem kullanımını daha hızlı hale getirmeye, makine öğrenmesi kitaplığını genişletmeye ve daha farklı harici veri kaynaklarına erişebilmeyi sağlama yolunda ilerlemektedir.

Spark yapılan işlemleri dağıtarak yaptığından dolayı düğüm sayısının artması işlemlerin hızlandırılması sağlanmaktadır. Spark, Hadoop'a rakip olmaktan daha çok Hadoop'da yapılan işlemlerin performansını arttırmak için geliştirilmiştir. Spark'da bir veri depolama birimi olmadığı için veriyi Hadoop'un HDFS dosya yapısını kullanır. Bu yüzden Hadoop'tan bağımsız değildir.

Apache Spark'ın getirdiği bu dağıtık hesaplama modelinde bir küme oluşturulur. Bu oluşan kümede işlemler birbirinden bağımsız olarak, paralel hesaplama yöntemi ile gerçekleştirilir. MapReduce sistemlerinde veri kümelere dağıtılır ve toplanarak işlem yapılır. Ancak Apache Spark'da bu dağıtım işlemi yinelemeli işlemler için kullanışlı olması için program sonlanana kadar toplanmaz. Bu şekilde veriyi tekrar tekrar kümeye dağıtma işlemi yapılmaz ve buradan kaynaklanan zaman kaybı en aza indirilmiş olur. Bu dağıtık hesaplama modeli, geliştirici kaynaklı oluşabilecek hataları azaltmak ve işçi düğümlerindeki yük dengelemesini sağlamak için katı kurallara sahiptir.

4.1. APACHE SPARK KÜMELEME SİSTEMİ

Büyük veri işleme mimarileri genel olarak aynı yapıdadır. Bu yapıda işlemleri kontrol eden bir sürücü (master) programı ve yapılacak işlemleri dağıtık olarak hesaplayacak olan işçi (worker) düğümlerinden oluşan bir küme yapısına sahiptir. Spark, dağıtık hesaplama yapabilmesi için iki ana soyutlamaya ihtiyaç duymaktadır. Bunlar: esnek dağıtılmış veri kümeleri ve bu veri kümelemelerine uygulanacak işlemlerdir. Bunlara ek olarak kümede çalışan işlemlerde kullanılabilen iki kısıtlı paylaşılan değişken türünü de destekler. Şekil 4.1'de Apache Spark küme yapısı gösterilmiştir.



Şekil 4.1. Apache Spark mimarisi.

Apache Spark, özellikle açık kaynak kodlu olması, her sürüm ile yeni özelliklerin gelmesi ve mevcut olan özelliklerin daha da verimli hale gelmesi sebebiyle büyük veri işleyen kişiler tarafından sıkılıkla tercih edilmektedir. Bunun en büyük nedeni petabayta ulaşan verilerin bu yapı sayesinde olabildiğince kısa sürede işlenebilmesi ve

sonuçların alınabiliyor olmasıdır. Spark'ın diğer büyük veri işleme sistemlerinde hızlı olmasının nedeni veriyi önbellekte tutmasıdır. Spark, işlem başlangıcında veriyi tüm küme elemanlarına dağıtır. Bu dağıtma işleminde öncelikli olarak veri önbelleğe kaydeder, eğer önbellek yeterli gelmezse disk içerisine veriyi kaydeder. Veri dağıtıldıktan sonra sürücü tarafından gelen görev yerine getirilir ve sonuç sürücüye iletilir.

Spark, çoğu konuda avantajlı olsa bile en büyük dezavantajı bir depolama biriminin olmamasıdır. Genel olarak Spark, Hadoop'un Hadoop Dağıtılmış Dosya Sistemi (Hadoop Distributed File System – HDFS) yapısını kullanır. Bu sayede dosyaları HDFS'den okuyup yazabilir. Ne kadar büyük verilerin işlenmesi büyük önem arz etse bile bunların depolanmasında büyük öneme sahiptir.

Temel olarak bir Spark uygulaması sürücü ve işçiler olarak iki ana bileşenden oluşurlar. Sürücü, kullanıcıdan gelen kodu (işi) işçi düğümlere dağıtmak ve çalışmalarını denetlemek ve toplamaktır. İşçi düğüm ise, gelen iş parçacığını en uygun şekilde yürütüp sonuç üretmektir.

4.1.1. Spark DataFrame

RDD'lerde ortaya çıkan performans ve ölçeklendirme sınırlarını çözmek ve RDD'lerde nesneleri tanımlayan bir veri tabanı yapısına sahip olmamasından dolayı Apache Spark 1.3 sürümüyle birlikte Spark DataFrame yapısı tanıtıldı. Bu yapı sayesinde gelen başlıca yenilikler:

1. Girdi Optimizasyon Motoru: DataFrame'ler, verileri daha verimli bir şekilde işlemek için Catalyst Optimizer gibi giriş optimizasyon motorlarını kullanırlar.
2. Yapılandırılmış Veri: Verilerin şematik bir görünümü olmasını sağlar. Buda verileri depolarken anlamlı gösterimler sağlar.
3. Özel Bellek Yönetimi: Veri RDD'lerde olduğu gibi yığın olarak depolanmaz bu sayede çöp toplayıcı yükünde azalma olur.

4.1.2. Paralelleştirilen İşlemler

DataFrame'lerde birçok paralel işlem gerçekleştirilebilir. Bu tez çalışmasında kullanılan fonksiyonlar aşağıda verilmiş ve açıklanmıştır.

1. agg: Her sütun kendi içerisinde gruplandırmaya yarayan ve sonuç olarak DataFrame üreten işlevdir.
2. collect: Veri kümesi içerisindeki tüm verileri sürücü programına gönderir. Örneğin; bir diziyi paralelleştirmek, eşlemek ve toplamak için kullanılabilir.
3. foreach: Her satırı kullanıcı tarafından belirlenen bir işlev uygular. Burada yalnızca veri içerisinde işlemler yapılır. Verinin içeriği değiştirilmez.

4.1.3. Paylaşılan Değişkenler

Paralelleştirme veya dağıtık hesaplama işlemlerinde bazen işlemlerin gerçekleştirilebilmesi için dışarıdan alınması gereken değerler vardır. Bu değerler dağıtık hesaplamada kullanılan global değişkenler olabildikleri gibi işlem sonuçlarının tutulması gereken sonuçlar için de kullanılabilir. Bu tez çalışmasında popülasyondaki bireylerin değerlerini dağıtık hesaplamada kullanmak için yayma (broadcast) işlemi uygulanmıştır. Ayrıca WSSSE skorlarını tutmak içinse akümülatör (accumulator) kullanılmıştır. Bu iki değişkenin genel tanımları aşağıda verilmiştir.

1. Yayın değişkeni (broadcast): İstenilen değişkenin her bir düğüme sadece okunur bir şekilde aktarılmasını sağlayan bir nesne oluşumudur.
2. Akümülatör (accumulator): Dağıtık hesaplama sürecinde eğer bir işlem sonucunun alınması gerekiyorsa bu sonuç akümülatör sayesinde hesaplanabilir. İşçi düğümler akümülatöre sadece yazma işlemi yapabilirler, sürücü düğüm ise akümülatör üzerinde tam yetkiye sahiptir.

4.1.4. Kümeleme Problemine Apache Spark Entegrasyonu

Apache Spark gibi büyük veri setlerinin hesaplama süresini düşürmek için tasarlanan sistemler genel olarak problemin tamamına uygulanmak yerine hesaplama süresinin en fazla olduğu noktalara uygulanmaktadır.

Al-Sawwa ve Ludwig [53] yapmış oldukları çalışmada PSO algoritmasını sınıflandırma problemin çözümü için güncellemişlerdir. Daha sonrasında bu sınıflandırma yapabilen PSO algoritmasını Apache Spark sistemine entegre ederek dağıtık hesaplama modelinden yararlanmışlardır. Burada Spark entegrasyonu sırasında problemin tamamını dağıtık hesaplama modeline uygun şekle getirmek yerine hesaplama maliyetinin en yüksek olduğu verileri uygun sınıflara atama işleminde Spark mimarisi kullanılmıştır. Bu çalışmada kullanılan veri setleri Apache Spark'ın RDD nesne yapısında okunmuş ve Map ve Reduce işlemleri ile uygunluk fonksiyonu sonuçları hesaplanmıştır. Higgs veri seti için sınıflandırma yani denetimli öğrenme metodu uygulamalarına rağmen %53'luk bir doğruluk oranı yakalanmıştır. Hızlanma (speed up) değerleri işçi düğüm sayısı artması işlemin süresini ters orantılı bir şekilde azaltmaktadır. Benzer bir şekilde Wang ve Qian [44] da yapmış oldukları çalışmada ABC algoritmasını kümeleme amacıyla kullanmışlar ve büyük ölçekli verilerin kümelenmesinde kullanmışlardır. Aynı şekilde bu çalışmada da verilerin uygun kümeye yerleşmesi ve uygunluk fonksiyonu sonucunun hesaplama kısmında Spark'ın dağıtık hesaplama modeli kullanılmıştır. Ayrıca artan işçi düğüm sayısının artması işlem süresini kısalttığı gözlemlenmiştir. Qi ve arkadaşlarının [54] yapmış olduğu çalışmada ise GA'nın ikili testlerde (pairwise test) kullanımı sırasında sadece uygunluk fonksiyonunun hesaplanması sırasında değilde tüm problemde uygulamışlardır. Bu sayede GA'daki her bir bireyin dağıtık hesaplama modelinde kullanılabilmesi için bireyleri RDD nesne yapısı içerisinde tanımlayarak dağıtık hesaplama yapısının içerisine entegre etmişlerdir. Alınan sonuçlar incelendiğinde ise işlem süresinin ciddi düşüşler olduğu gözlemlenmiştir. Tripathi ve arkadaşlarının [55] yaptığı GWO ile kümeleme probleminin çözümü uygulamasında yine Hadoop MapReduce işlemi kullanılmıştır. Ama bu çalışmada kullanılan veri setini işçi düğümlere dağıtmak yerine popülasyondaki bireyleri işçi düğümlere dağıtarak hızlanma sağlamayı amaçlamışlardır. Çalışmanın sonuçlarına bakıldığında yaklaşık olarak %40 civarında bir hızlanma gözlemlenmiştir.

Yapılan bu araştırmalarda Apache Spark dağıtık hesaplama modeli meta-sezgisel algoritmalar ile kümeleme probleminin çözümünde kullanıldığı gözlemlenmiştir. Burada genel olarak hesaplama maliyeti yüksek olan yerlerde Apache Spark sisteminin kullanıldığı görülmüştür. Bu tez çalışmasında yapılacak olan yöneticisiz kesin bölümlemeli kümeleme probleminin meta-sezgisel algoritmalar yardımıyla çözülmesi işleminde hesaplama maliyeti en yüksek yer olan veri setindeki elemanların uygun kümelere atanma işlemini ve WSSSE uygunluk fonksiyonunun hesaplanmasında Apache Spark kullanılmıştır.

Şekil 4.2'de Apache Spark'ın DataFrame yapısının dağıtık hesaplama modelinin akümülatör kullanılarak bu tez çalışması içerisindeki kullanım şekli gösterilmiştir. Bu çalışmada kullanılan veri seti DataFrame yapısı olarak ele alınıp işçi düğümlere dağıtılacak ve popülasyon ise yayın dağıtımı (broadcast) olarak tüm düğümlere gönderilerek verilerin uygun küme merkezlerine atanması sağlanacaktır. Uygun küme merkezine atanan verilerin uygunluk fonksiyonu değerleri ise akümülatöre eklenerek tüm veri seti için uygunluk değeri hesaplanabilecektir. Bu anlatılan işlemleri sahte kodu Çizelge 4.1'de gösterilmiştir.

Çizelge 4.1. WSSSE değerinin hesaplanması.

GİRDİ:

Veri : $\boldsymbol{d}$**-boyutlu veri seti** $\boldsymbol{X = \{\overrightarrow{x_1}, \dots, \overrightarrow{x_j}, \dots, \overrightarrow{x_N}\}}$

Kmerkezleri: Küme merkezleri

ÇIKTI:

sonuc : uygunluk fonksiyonu sonuç değeri

```
1: Fonksiyon WSSSEHesapla(Veri, Kmerkezleri)
2:    sonuc = 0.0
3:    enkısauzaklık = INF
4:    for each Veri
5:       for merkez in Kmerkezleri do
6:          uzaklık = ÖklidUzaklıkHesapla(Veri, merkez)
7:          if(enkısauzaklık > uzaklık)
8:             enkısauzaklık = uzaklık
```

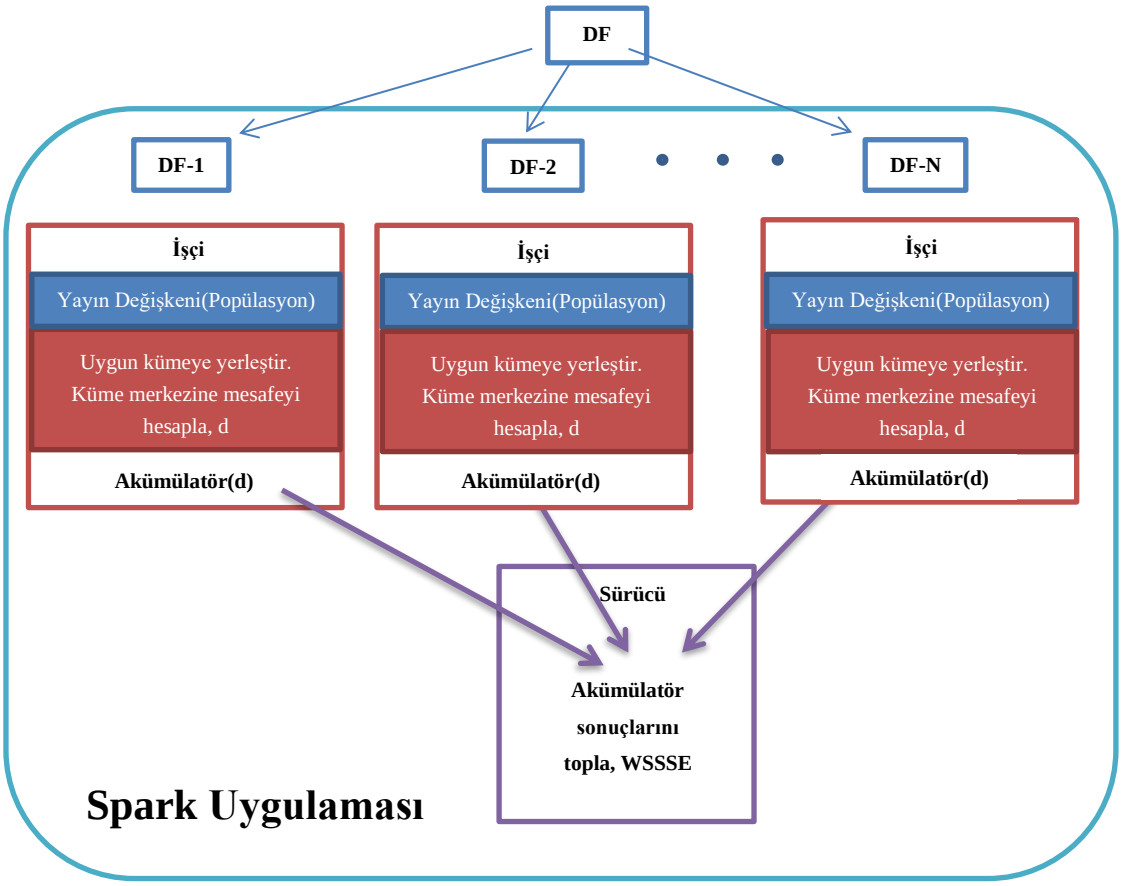
9: **end if**

10: sonuc = sonuc + enkısauzaklık

11: **end for**

12: **end for**

13: **return sonuc**



Şekil 4.2. Apache Spark'ın kümeleme problemine uygulanması.

BÖLÜM 5

DENEYSEL ÇALIŞMALAR

Bu bölümde Apache Spark kümeleme mimarisinin tasarım aşamaları, kullanılan veri seti ve meta-sezgisel algoritmaların kümeleme sonuçlarından elde edilen bilgiler anlatılmaktadır. Meta-sezgisel algoritmaların işlem süreleri ve kümeleme başarımları çizelgeler ve şekillerle değerlendirilmiştir. Yapılan tüm deneyler Scala programlama dilinde yazılmıştır.

5.1. APACHE SPARK KÜMELEME MİMARİSİ TASARIMI

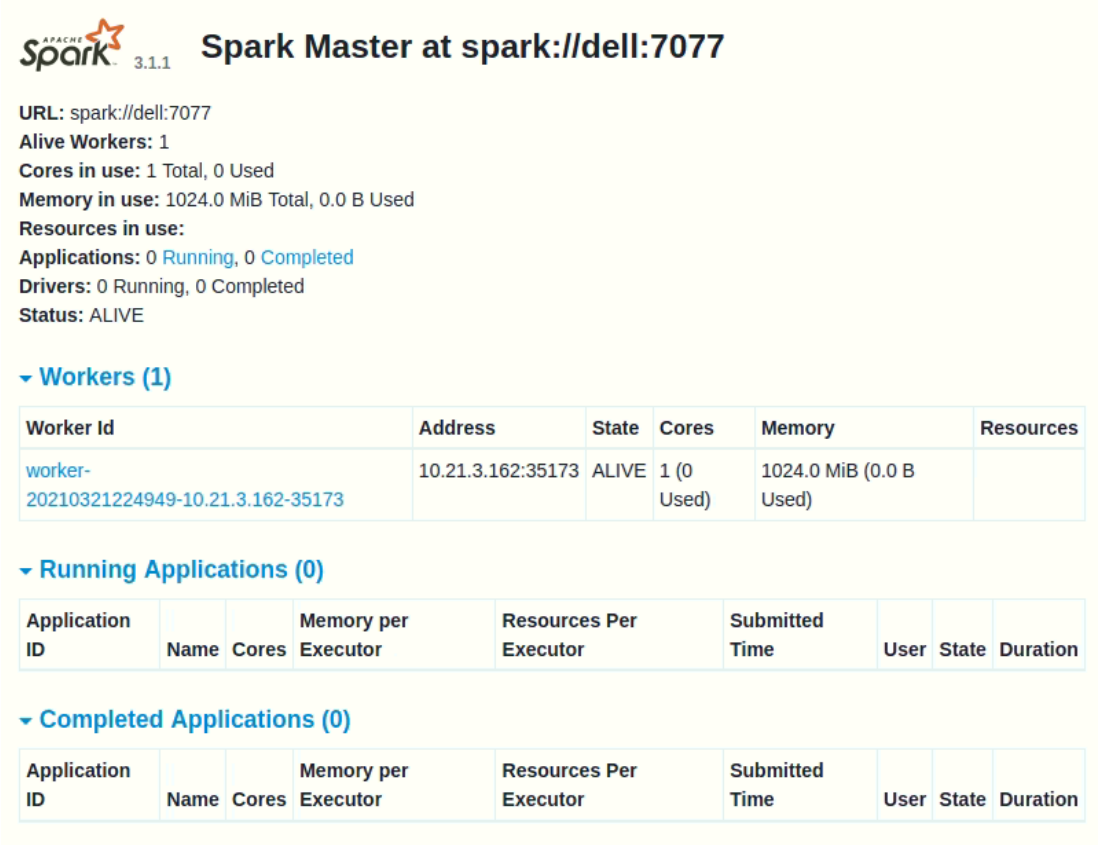
Apache Spark'ın küme yapısı kullanılarak yapılan deneylerde veri seti ile Apache Spark'ın hız performansı test edilmiştir. Apache Spark kümeleri sürücü (master) ve işçi (worker) düğümlerinden oluşur. Burada sadece sürücü düğümünden oluşan kümelere Standalone adı verilir. İşçi düğümlerinin ise sayısının artması dağıtık hesaplama süresinin kısalmasında temel oluşturmaktadır. Bu çalışmada 1 sürücü 1 işçi, 1 sürücü 2 işçi ve 1 sürücü 3 işçi'den oluşan Apache Spark kümeleri oluşturularak yöneticisiz kesin bölümlemeli kümeleme probleminin işlem süresi analiz edilmiştir. Yapılan testlerde işçi düğüm sayısının artması, işlem süresini kısalttığı gözlemlenmiştir. Spark küme yapısı Ubuntu 20.04 işletim sistemi ve Apache Spark 3.1.1 sürümü üzerinde oluşturulmuştur. Terminalden Spark üzerinde 1 master 1 işçi düğüm oluşturulduğunda Web GUI (Graphic User Interface) üzerinde Şekil 5.1'deki gibi görünür.

Şekil 5.1'deki Spark kümesini oluşturabilmek için öncelikle Apache Spark klasörünün içerisindeki conf klasörü açılır ve içindeki spark-env.sh.template dosyası ismi spark-env.sh yapılarak kaydedilir. Daha sonra dosya içerisine;

export SPARK_DRIVER_CORES=1

export SPARK_DRIVER_MEMORY=1g

export SPARK_WORKER_INSTANCES=1



Şekil 5.1. Web GUI üzerinden Spark ile tanımlı 1 master 1 işçi.

export SPARK_WORKER_CORES=1

export SPARK_WORKER_MEMORY=1g

satırları girilerek dosya kaydedilir. Burada 1 işlemciye ve 1 GB önbelleğe sahip işçi ve sürücü tanımlanmıştır. Bu tanımlamalar yapılırken kullanılan cihazın özellikleri göz önünde bulundurulmalıdır.

$ /usr/local/spark-3.1.1/sbin/start-all.sh komutu çalıştırılarak tanımlanmış olan sürücü ve işçi aktif edilir.

$ /usr/local/spark-3.1.1/sbin/stop-all.sh komutu ise aktif edilmiş olan tüm düğümleri kapatma işlemini gerçekleştirir. Özellikle Spark küme mimari yapısında değişiklik yapılacağında aktif olan tüm düğümlerin kapalı olmasına dikkat edilmelidir.

Terminal üzerinden Spark kümesinin oluşturulması bu şekilde yapılmaktadır. Fakat uygulamamızı da bu küme yapısında çalışması için aşağıdaki komut girilmelidir.

$ ./usr/local/spark-3.1.1/bin/spark-shell --master spark://bilgisayarAdı:7077

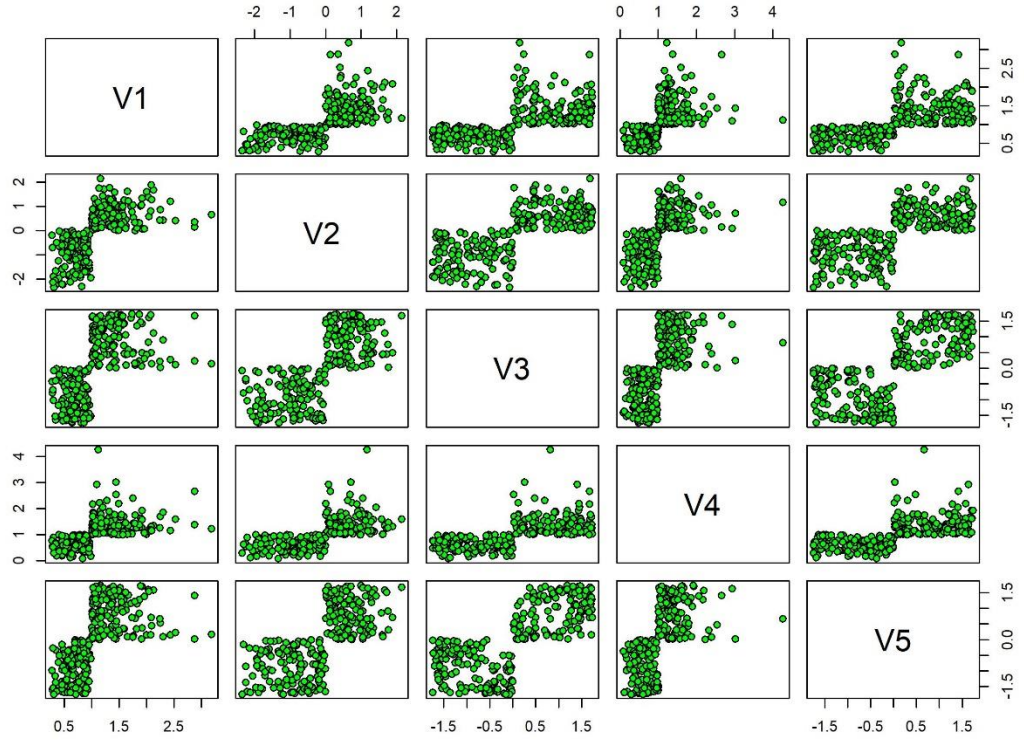
Bu komutta sürücü makine olarak WEB GUI'in sol üst köşesinde görülen URL adresi girilmelidir. Bu komut çalıştırıldığında Scala komut sistemi açılmaktadır.

Bu çalışmada, sürücü ve işçi düğümler tanımlanırken her bir düğüm için 1 GB önbellek ve 1 işlemci (CPU) ataması yapılmıştır. Ayrıca Apache Spark dağıtık hesaplama modeli kullanılan bütün deneyler Scala komut satırında test edilmiştir.

5.2. KULLANILAN VERİ SETİ

Bu çalışmada Baldi ve arkadaşları [56] tarafından yapılan derin öğrenme modellemesi çalışmasından kullanılmak üzere Monte Carlo simülasyonu kullanılarak üretilmiş UCI veri setlerinden Higgs [57] veri seti kullanılmıştır. Bu veri seti 28 nitelikten ve 2 sınıftan oluşmaktadır. İlk 21 nitelik parçacık hızlandırıcı tarafından ölçülen düşük seviyeli kinematik değerlerdir. Diğer 7 nitelik ise ilk 21 niteliğin fonksiyonel sonuçlarıdır. Şekil 5.2'de bu veri setinin ilk 5 niteliğinin ikişerli olarak gösterilmiştir. Şekilde V1, V2, V3, V4 ve V5 nitelikleri temsil ederken, bu gösterimlerden çizilecek yatay ve dikey çizgilerin kesiştikleri yerler bu niteliklerin ikili olarak gösterimleridir. Grafikler, tek renk gözükmesine rağmen veri içeriğinde iki farklı sınıfı tanımlayan iki farklı renk vardır ancak veriler iç içe olduğundan dolayı tek renk gözükmektedir. Geriye kalan niteliklerinde dağılımı aynı şekilde devam etmektedir. Bu çalışmada işlem süresini kısaltmak amacıyla bu veri setinden rasgele 12.500 (2.6 MB), 25.000 (5.3 MB) ve 50.000 (11 MB) satırlık alt veri setleri oluşturulmuştur. Bu alt veri setleri sırasıyla 1 sürücü 1 işçi, 1 sürücü 2 işçi ve 1 sürücü 3 işçi'den oluşan Spark kümeleri üzerinde her bir meta-sezgisel algoritma için çalıştırılarak geliştirilen algoritmaların

ölçeklenebilirliği test edilmiştir. Ayrıca 50.000 satırlık alt veri seti 1 sürücü 1 işçi, 1 sürücü 2 işçi, 1 sürücü 3 işçi ve 1 sürücü 4 işçi'den oluşan Spark kümeleri ile her bir meta-sezgisel algoritma için çalıştırılarak problemin hızlanma eğilimi test edilmiştir.



Şekil 5.2. Higgs veri seti.

5.3. META-SEZGİSEL ALGORİTMALARIN UYGULANMASI

Genel olarak, meta-sezgisel algoritmalar parametreli ve parametresiz olmak üzere ikiye ayrılırlar. Bu çalışmada kullanılan algoritmalardan HS ve AAA parametreli algoritmalardır. Çizelge 5.1'de kullanılan meta-sezgisel algoritmaların parametreleri verilmiştir.

Çizelge 5.1. Meta-Sezgisel algoritma parametreleri.

Parametre Adı	HS	GWO	AAA
Popülasyon Boyutu	30	30	30
İterasyon	400	400	400
AHDO	0.9	-	-

TAO	0.4	-	-
BG	0.2	-	-
Enerji Kaybı	-	-	0.3
Adaptasyon	-	-	0.2
Kesme Kuvveti	-	-	2.0

Hızlanma [58], tek düğümdeki çalışma süresinin birden fazla düğüm sayısındaki çalışma süresine oranıdır. Bu oran paralelleştirme başarımı ölçmek için kullanılır. Bu tez çalışmasında meta-sezgisel algoritmaların büyük veri setlerinin kümelenmesi problemindeki işlem süresinin düğüm sayısının artmasıyla kısaldığını görülmektedir. Hızlanma aşağıdaki gibi hesaplanır [59].

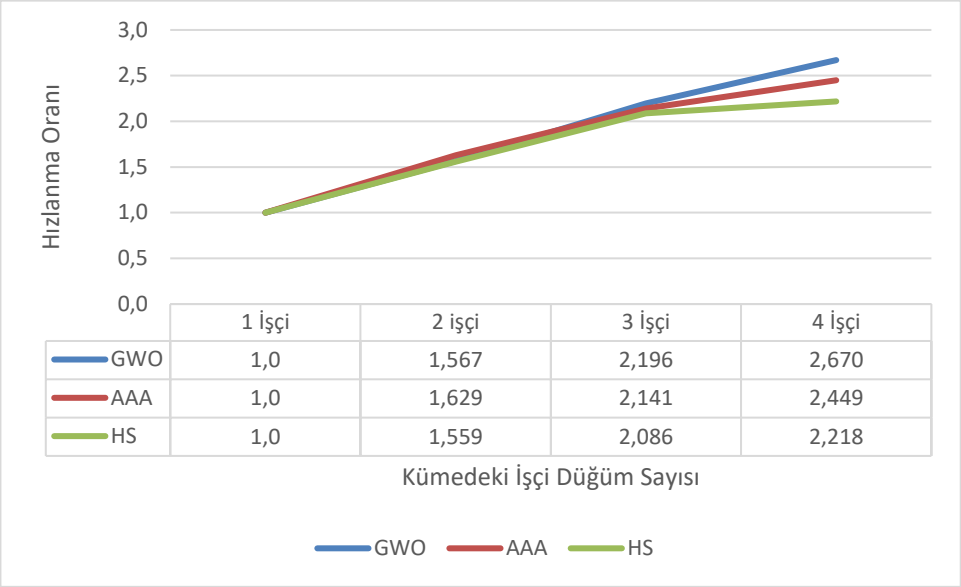
$$Hızlanma = \frac{T_1}{T_n} \tag{5.1}$$

burada T_1, tek düğümdeki işlem süresi, T_n, n adet düğümdeki işlem süresidir.

Öncelikle 50.000 satırdan oluşan veri setini sırasıyla 1 sürücü 1 işçi, 1 sürücü 2 işçi, 1 sürücü 3 işçi ve 1 sürücü 4 işçi'den oluşan Spark kümelerinde algoritmaların hızlanmaları incelenmiştir. Çizelge 5.2'de kullanılan meta-sezgisel algoritmaların işlem süreleri saniye cinsinden gösterilmiştir. Şekil 5.2'de ise hızlanma oranlarını göstermektedir.

Çizelge 5.2. Meta-sezgisel algoritmaların işlem süreleri (Saniye).

Algoritma	1 sürücü 1 İşçi	1 sürücü 2 işçi	1 sürücü 3 İşçi	1 sürücü 4 İşçi
GWO	8.448,30	5.391,04	3.846,94	3.164,23
AAA	15.490,78	9.508,51	7.234,72	6.324,06
HS	317,86	203,88	152,36	143,33



Şekil 5.2. Meta-sezgisel algoritmaların hızlanma oranları.

Yukarıdaki şekil ve çizelge incelendiğinde artan işçi düğüm sayısının işlem süresini kısalttığı gözlemlenebilmektedir. En yüksek hızlanma oranına sahip GWO algoritması için 1 sürücü 1 işçi ile 1 sürücü 4 işçi arasında 2,6 kat hızlanma olduğu gözlemlenmektedir. Diğer algoritmalar için de 1 sürücü 1 işçi ile 1 sürücü 4 işçi arasında GWO'ya yakın bir şekilde işlem sürelerinden hızlanma olduğu gözlemlenmiştir. İşlem sürülerini göz önüne aldığımızda ise bu sürelerdeki farklılıklar algoritmaların çalışma prensiplerinden kaynaklanmaktadır. HS'nin işlem süresinin kısa çıkmasının temel sebebi ilk iterasyonda popülasyonun tamamı ile işlem yapılması, geri kalan iterasyonlarda ise tüm popülasyondan sadece tek bir yeni birey üretilerek üretilen bu bireyin kümeleme performansına bakılmasıdır. Yani 20 iterasyonluk bir problemde toplamda 39 birey oluşturulmuş ve sadece bunların uygunluk değerleri hesaplanmıştır. GWO'nun işlem süresinde ise her iterasyonda tüm popülasyonun küme merkezleri güncellenir ve kümeleme performansları yeniden hesaplanır. Genel olarak çoğu meta-sezgisel algoritmada bu durum gerçekleşmektedir. AAA'nin işlem süresinin en fazla çıkmasının sebebi, her iterasyonun başlangıcında tüm bireylere o anki uygunluk değerlerine göre [0,1] arasında enerji değerleri atanır. Bu enerji değerleri bireyin iterasyondaki hareket sayısına etki etmektedir. Bu durum da her iterasyonda yaklaşık olarak popülasyon sayısının 3 katı kadar küme merkezi güncellemesi yapmasına yol açmaktadır. Bu algoritmaların problem çözüm yaklaşımlarının farklı olması sadece işlem sürelerini değil aynı zamanda uygunluk

fonksiyonu sonuçları ve doğruluk oranları arasında da farklılık olmasını sağlamaktadır. Çizelge 5.3'de kullanılan algoritmaların Apache Spark'ın dağıtık hesaplama modülü kullanılarak ve Apache Spark'ın dağıtık hesaplama modülü kullanılmadan elde edilen WSSSE skorları ve doğruluk değerleri verilmiştir.

Çizelge 5.3. Meta-sezgisel algoritmaların WSSSE skorları ve doğruluk oranları.

Algoritma	Doğrulama Kriteri	Seri Çalıştırma	1 Sürücü 1 İşçi	1 Sürücü 2 İşçi	1 Sürücü 3 İşçi	1 Sürücü 4 İşçi
GWO	**WSSSE**	**210.561,65**	**226.923,23**	**235.902,40**	**237.064,45**	**234.623,90**
	Doğruluk %	**73,16**	**70,02**	**79,42**	**77,41**	**71,23**
AAA	**WSSSE**	367.860,50	361.526,66	384.223,95	384.392,46	454.030,44
	Doğruluk %	49,99	51,55	50,48	50,52	49,99
HS	**WSSSE**	277.920,00	274.892,86	289.944,39	261.582,87	266.971,04
	Doğruluk %	50	50	49,99	49,99	50,01

Yukarıdaki çizelgede algoritmaların 50.000 satırlık veri seti ile 1 sürücü 1 işçi, 1 sürücü 2 işçi ve 1 sürücü 3 işçi 'den oluşan Spark kümelerinde ve Apache Spark kullanılmadan aynı algoritmaların Java programlama dilinde geliştirilmiş olan seri çalıştırma sonucunda elde edilen WSSSE skorları ve doğruluk oranları gösterilmiştir. Çizelgede de görüldüğü gibi Apache Spark'ın dağıtık hesaplama modülü ile seri çalıştırılma arasında WSSSE skorlarına ve doğruluk değerlerine bakıldığında anlamlı bir farklılık görülmemektedir. Bu sonuçta bize kullanılan Apache Spark dağıtık hesaplama modülünün hesaplamalarda sapmaya sebep olmadığını göstermiştir. Bu çizelgede başarım en düşük WSSSE skoruna ve en yüksek doğruluk oranına göre ele alınmalıdır. Bu durumlar göz önüne alındığında GWO algoritması tüm deneylerde diğer algoritmalar göre HIGGS veri seti için en başarılı algoritmadır. Bunun başlıca sebebi algoritmalar arasındaki problem çözüm yaklaşımlarının farklı olmasıdır. GWO en uygun sonucu veren bireylerin etrafında toplanması Higgs veri seti için onu en başarılı algoritma yapmıştır. HS'nin en kötü sonucu veren bireyi eleyerek ilerlemesi ve AAA'nın ise alg kolonisi içerisinden rastgele seçtiği iki koloniden daha iyi olanına göre küme merkezi değiştirmesi, bu veri setinde yeterli başarıya ulaşamamasına sebep olmuştur. Ancak WSSSE skorları göze alındığında GWO ve HS'nin sonuçları birbirine yakın olmasına rağmen doğruluk oranları arasında %20'lik bir farklılık gözlenmektedir. Benzer bir durum AAA ve HS içinse doğruluk oranlarının neredeyse

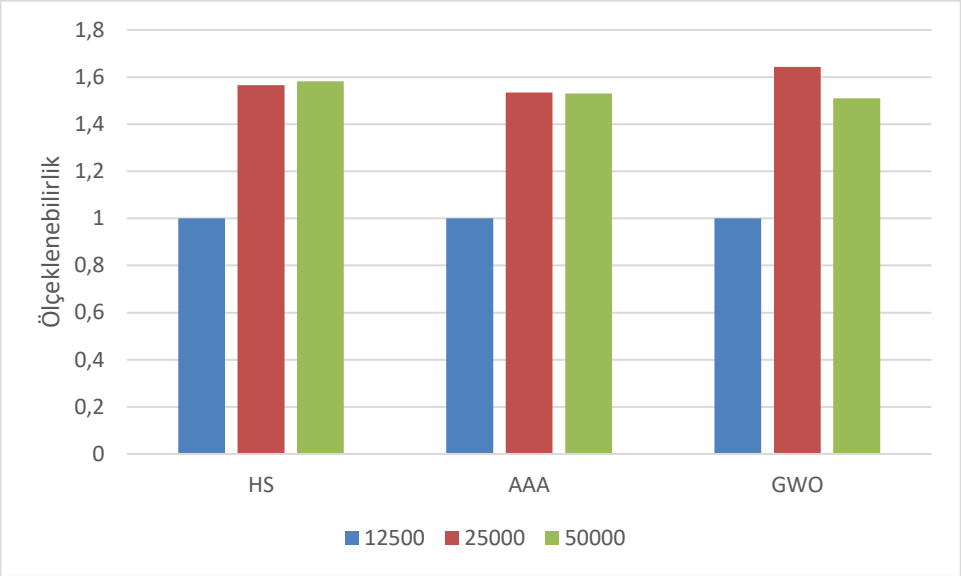
aynı olmasına rağmen WSSSE skorları arasındaki fark çok fazla olarak gözlenmektedir. Bu durumun başlıca sebebi Higgs veri setinin iç içe geçmiş ancak farklı kümelere ait değerlerden oluşmasından kaynaklanmaktadır. Buda bize HS ve AAA algoritmalarının iç içe değerlerden oluşan veri setlerinin kümelenmesinde dezavantajlı durumda kaldıklarını göstermiştir. Bunun yanında GWO algoritması ise en iyi sonuçlarının etrafında toplanmasından dolayı bu tür veri setlerinin kümelenmesinde avantajlı konumda kaldığı görülmektedir.

Bu tez çalışmasında algoritmaların başarımlarını ölçmek için kullanılan bir diğer yöntem ise ölçeklenebilirliktir. Ölçeklenebilirlik, Spark kümelerinin dağıtık hesaplama modeli tarafından verimli kullanılıp kullanılmadığını ölçmek için kullanılır. Ölçeklenebilirlik aşağıdaki gibi hesaplanır ve veri seti boyutu ve düğüm sayısı aynı oranda arttırılarak incelenir [59].

$$Ölçeklenebilirlik = \frac{T_n}{T_1} \tag{5.2}$$

burada T_n, n adet düğümdeki çalışma süresi, T_1, 1 adet düğümdeki çalışma süredir.

Ölçeklendirilebilirliği ölçmek için Higgs veri 12.500, 25.000 ve 50.000 satırlık veri alt kümeleri alınmıştır ve bunlar sırasıyla 1 sürücü 1 işçi, 1 sürücü 2 işçi ve 1 sürücü 4 işçi ‘den oluşan Spark kümeleri ile her bir meta-sezgisel algoritma çalıştırılarak incelenmiştir. HS, GWO ve AAA algoritmasının ölçeklenebilirliği Şekil 5.5’de gösterilmiştir.



Şekil 5.5. Meta-Sezgisel algoritmaların veri boyutu ölçeklenebilirlik grafiği.

Yukarıdaki şekil incelendiğinde kümeleme problemin çözümü için geliştirilen algoritmanın belirli bir veri boyutundan sonra ölçeklenebilir olduğu gözlemlenmiştir. Bunun nedeni işçi düğümlere verilen işlemlerin yük dengesinin sağlanabilmesi için belli bir miktarda veri boyutunun aşılması gerektiği gözlemlenmiştir.

BÖLÜM 6

SONUÇLAR

Bu çalışmada, tamamen iç içe geçmiş verilerden oluşan büyük veri setinin kümelenmesinin zor olduğu görülmüştür. Ancak en uygun sonuca ulaşmak için farklı yaklaşımlar kullanan meta-sezgisel algoritmaların farklı başarımlar elde ettiği gözlemlenmiştir. Bu yaklaşımlar algoritmaları farklı veri setleri ve farklı problemlerin çözümünde avantajlı veya dezavantajlı durumda kalmasına sebep olmaktadır. Bu çalışma bize en uygun sonucu veren bireyin etrafında toplanan bir meta-sezgisel algoritma olan GWO'nun bu veri setinin kümelenmesinde avantajlı olduğu görülmüştür. HS'nin en kötü sonuçları eleyerek popülasyonun genelinde en iyileme yapmaya çalışması ve AAA'nın ise popülasyon içinden seçtiği rastgele iki koloniden daha iyi olanına göre konum değiştirmesi bu veri seti için onları dezavantajlı durumda bırakmıştır.

Çalışmadaki diğer bir zorluk ise büyük veri setlerindeki yüksek hesaplama süresinin en uygun şekilde kısalmasını sağlamaktır. Bu durumu sağlayabilmek için problem, dağıtık hesaplama modeline sahip olan Apache Spark sistemine entegre edilmiştir. Bu entegrasyon işleminde ise algoritmaların kümeleme başarımlarının yanı sıra iki farklı daha başarım elde edilmesi gerekmektedir. Bunlar;

1. Hızlanma : Artan düğüm sayısına bağlı olarak işlem süresinin kısaltılabilmesi,
2. Ölçeklenebilirlik : Eşit miktarlarda artan veri boyutu ile düğüm sayısının işlem süresinde olabildiğince az değişiklik göstermesini sağlayabilmektir.

Bu maddelerin gerçekleştirilebilmesi için geliştirilen algoritmalar incelenmiş ve hesaplama maliyeti en yüksek olan veri setindeki elemanların uygun kümelere atanması ve atandıkları küme merkezlerine uzaklıklarının hesaplanması işlemleri

dağıtık hesaplama modeli ile gerçekleştirilmesi durumunda yukarıda istenilen başarımların gerçekleştirilebileceği görülmüştür.

Bu maddelerden birincisi, yani hızlanma testi için yapılan deneylerde artan işçi düğüm sayısının işlem süresini kısalttığı gözlemlenmiştir. 1 sürücü 1 işçi ile 1 sürücü 4 işçi düğüm arasındaki hızlanmalara bakıldığında en iyi hızlanmayı 2,67 kat ile GWO algoritmasının sağladığı, diğer algoritmalarda ise bu oran HS için 2,2 kat , AAA içinse 2,45 kat olarak gözlemlenmiştir.

Ölçeklenebilirlik konusunda ise geliştirilen algoritmaların hemen hemen aynı sonuçları verdiği gözlemlenmiştir. Belirli bir veri boyutunun geçilmesi geliştiren algoritmalarda bir yük dengesinin oluşmasına ve işlem süreleri arasında farklılığın azalmasını sağlamıştır.

İleriye dönük yapılabilecek çalışmalar ise;

1. Apache Spark teknolojisinin dağıtık hesaplama modeli ile geliştirilen meta-sezgisel algoritmaların daha farklı problemlerin çözüm için kullanılabilir.
2. Meta-sezgisel algoritmaların kümeleme başarımını test etmek amacıyla farklı veri setlerinde kullanılabilir.
3. Meta-sezgisel algoritmaları kendi aralarında veya diğer kümeleme algoritmaları ile birleştirilerek hibrit bir algoritma geliştirilip başarımının artması sağlanabilir.

KAYNAKLAR

1. Mirjalili, S., S.M. Mirjalili, and A.J.A.i.e.s. Lewis, "Grey wolf optimizer", ***Advances in Engineering Software***, 2014, 69: p, 46-61.

2. Blum, C. and A.J.A.c.s. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison", ***ACM Computing Surveys***, 2003, 35(3): p. 268-308.

3. Ng, K., et al., "Review on meta-heuristics approaches for airside operation research", ***Applied Soft Computing***, 2018. 66: p. 104-133.

4. Khan, B. and P.J.I.A. Singh, "Selecting a meta-heuristic technique for smart micro-grid optimization problem: A comprehensive analysis", ***IEEE Access***, 2017. 5: p. 13951-13977.

5. Tarantilis, C.D., G. Ioannou, and G.J.J.o.F.E. Prastacos, "Advanced vehicle routing algorithms for complex operations management problems", ***Journal of Food Engineering***, 2005. 70(3): p. 455-471.

6. Wang, S.-C., "Genetic algorithm, in Interdisciplinary Computing in Java Programming", 2003, ***Springer***. p. 101-116.

7. Kennedy, J. and R. Eberhart. "Particle swarm optimization. in Proceedings of ICNN'95-International Conference on Neural Networks", 1995, ***IEEE***.

8. Rajabioun, R.J.A.s.c., "Cuckoo optimization algorithm". ***Applied soft computing***, 2011. 11(8): p. 5508-5518.

9. Karaboga, D. and B.J.A.s.c. Basturk, "On the performance of artificial bee colony (ABC) algorithm". ***Applied soft computing***, 2008. 8(1): p. 687-697.

10. Dorigo, M., M. Birattari, and T.J.I.c.i.m. Stutzle, "Ant colony optimization". ***IEEE computational intelligence magazine***, 2006. 1(4): p. 28-39.

11. Simon, D.J.I.t.o.e.c., "Biogeography-based optimization". ***IEEE transactions on evolutionary computation***, 2008. 12(6): p. 702-713.

12. Erol, O.K. and I.J.A.i.E.S. Eksin, "A new optimization method: big bang–big crunch". ***Advances in Engineering Software***, 2006. 37(2): p. 106-111.

13. Hatamlou, A.J.I.s., "Black hole: A new heuristic optimization approach for data clustering", ***Information sciences***, 2013. 222: p. 175-184.

14. Mucherino, A. and O. Seref. "Monkey search: a novel metaheuristic search for global optimization. in AIP conference proceedings". 2007. ***American Institute of Physics***.

15. Mirjalili, S. and A.J.A.i.e.s. Lewis, "The whale optimization algorithm", ***Advances in engineering software***, 2016. 95: p. 51-67.

16. Geem, Z.W., J.H. Kim, and G.V.J.s. Loganathan,"A new heuristic optimization algorithm: harmony search". ***Simulation***, 2001. 76(2): p. 60-68.

17. Yang, X.-S., "Harmony search as a metaheuristic algorithm, in Music-inspired harmony search algorithm". 2009, ***Springer***. p. 1-14.

18. Muro, C., et al., "Wolf-pack (Canis lupus) hunting strategies emerge from simple rules in computational simulations", ***Behavioural processes***, 2011. 88(3): p. 192-197.

19. Saremi, S., et al., "Evolutionary population dynamics and grey wolf optimizer", ***Neural Computing Applications***, 2015. 26(5): p. 1257-1263.

20. Uymaz, S.A., "Yeni bir biyolojik ilhamlı metasezgisel optimizasyon metodu: Yapay alg algoritması". 2015, **Selçuk Üniversitesi Fen Bilimleri Enstitüsü**.

21. Graham, L. and L.J.U.S.R. Wilcox, New Jersey, "Algae Prentice Hall". ***Advances in Microbiology***, 2000. 4.

22. Buhr, H. and S.J.W.R. Miller, "A dynamic model of the high-rate algal-bacterial wastewater treatment pond", ***Frontiers International Conference on Wastewater Treatment and Modelling***, 1983. 17(1): p. 29-37.

23. Liu, S.J.B.E.E., "Chapter 11: How Cells Grow. Bioprocess Engineering". ***Elsevier***, Amsterdam, 2013: p. 549.

24. Xu, R. and D.J.I.T.o.n.n. Wunsch, "Survey of clustering algorithms", ***IEEE Transactions on neural networks***, 2005. 16(3): p. 645-678.

25. Das, S., et al., "Automatic clustering using an improved differential evolution algorithm", ***IEEE Transactions on Systems Man and Cybernetics***, 2007. 38(1): p. 218-237.

26. Murtagh, F., P.J.W.I.R.D.M. Contreras, and K. Discovery, "Algorithms for hierarchical clustering: an overview", ***Wiley Interdisciplinary Reviews: Data Mining Knowledge Discovery***, 2012. 2(1): p. 86-97.

27. Van der Merwe, D. and A.P. Engelbrecht, "Data clustering using particle swarm optimization", ***Congress on Evolutionary Computation***, 2003.

28. Hussain, S.F., A. Pervez, and M.J.A.S.C. Hussain, "Co-clustering optimization using Artificial Bee Colony (ABC) algorithm", ***Applied Soft Computing***, 2020. 97: p. 106725.

29. Miyamoto, S., et al., "Algorithms for fuzzy clustering". 2008: ***Springer***.

30. Krishna, K., Murty M.N., "Genetic K-means algorithm*",* ***IEEE Transactions on Systems***, Man,Cybernetics, Part B, 1999. 29(3): p. 433-439.

31. Hartigan, J.A. and Wong M.A., "Algorithm AS 136: A k-means clustering algorithm", ***Journal of the royal statistical society***. series, 1979. 28(1): p. 100-108.

32. Bezdek, J.C., et al., "FCM: The fuzzy c-means clustering algorithm", ***Computers & Geosciences***, 1984. 10(2-3): p. 191-203.

33. Pal, N.R., et al., "A possibilistic fuzzy c-means clustering algorithm", ***IEEE transactions on fuzzy systems***, 2005. 13(4): p. 517-530.

34. Maulik, U. and S.J. Bandyopadhyay, "Genetic algorithm-based clustering technique", ***Pattern recognition***, 2000. 33(9): p. 1455-1465.

35. Chen, C.-Y. and F. Ye., "Particle swarm optimization algorithm and its application to clustering analysis", ***Proceedings of 17th Conference on Electrical Power Distribution***. 2012.

36. Shelokar, P., V.K. Jayaraman, and B.D.J.A.C.A. Kulkarni, "An ant colony approach for clustering", ***Analytica Chimica Acta***, 2004. 509(2): p. 187-195.

37. Verma, H., D. Verma, and P.K.J.E.S.w.A. Tiwari, "A population based hybrid FCM-PSO algorithm for clustering analysis and segmentation of brain image", ***Expert Systems with Applications***, 2020: p. 114121.

38. Kaushik, K. and V.J.P.C.S. Arora, "A hybrid data clustering using firefly algorithm based improved genetic algorithm", ***Procedia Computer Science***, 2015. 58: p. 249-256.

39. Ortakcı, Y., "Parçacık sürü optimizasyonu yöntemlerinin uygulamalarla karşılaştırılması". ***Karabük Üniversitesi***, 2011.

40. Deeb, H., et al., "Improved Black Hole optimization algorithm for data clustering", ***Journal of King Saud University - Computer and Information Sciences***, 2020.

41. Kushwaha, N., et al., "Magnetic optimization algorithm for data clustering", ***Pattern Recognition Letters***, 2018. 115: p. 59-65.

42. Karaboga, D. and C. Ozturk, "A novel clustering approach: Artificial Bee Colony (ABC) algorithm", ***Applied soft computing***, 2011. 11(1): p. 652-657.

43. Kapoor, S., et al., "A grey wolf optimizer based automatic clustering algorithm for satellite image segmentation", ***Procedia computer science***, 2017. 115: p. 415-422.

44. Ji, J., et al., "A novel artificial bee colony based clustering algorithm for categorical data", ***PloS one***, 2015. 10(5): p. e0127125.

45. Özbakır, L. and F.J. Turna, "Clustering performance comparison of new generation meta-heuristic algorithms", ***Knowledge-Based Systems***, 2017. 130: p. 1-16.

46. İnternet: ORTAKCI, Y., "Parallel Particle Swarm Optimization in Data Clustering", ***https://web.karabuk.edu.tr/yasinortakci/dokumanlar/yay%C4%B1nlar/ISRED2017_Mekke.pdf***,(2020).

47. Aljarah, I., et al., "Clustering analysis using a novel locality-informed grey wolf-inspired clustering approach", ***Knowledge and Information Systems***, 2020. 62(2): p. 507-539.

48. Halkidi, M., Y. Batistakis, and M. Vazirgiannis. "Clustering algorithms and validity measures", ***Thirteenth International Conference on Scientific and Statistical Database Management***, 2001.

49. Premalatha, K., A.J.C. Natarajan, and I. Science, "A New Approach for Data Clustering based on PSO with Local Search", ***Computer Information Science***, 2008. 1(4): p. 139-145.

50. León, J., et al., "A Multi-Objective Optimization Algorithm for Center-Based Clustering", ***Electronic Notes in Theoretical Computer Science***, 2020. 349: p. 49-67.

51. Zaharia, M., et al., "Spark: Cluster computing with working sets", ***HotCloud***, 2010. 10(10-10): p. 95.

52. İnternet: Apache Spark, ***http://spark.apache.org/docs/latest/#spark-overview,*** 2020.

53. Al-Sawwa, J., et al., "Parallel particle swarm optimization classification algorithm variant implemented with Apache Spark", ***Concurrency and Computation Practice and Experience***, 2020. 32(2): p. e5451.

54. Qi, R.-Z., et al., "A parallel genetic algorithm based on spark for pairwise test suite generation", ***Journal of Computer Science Technology***, 2016. 31(2): p. 417-427.

55. Tripathi, A.K., K. Sharma, and M.J. Bala, "A novel clustering method using enhanced grey wolf optimizer and mapreduce", ***Big data research***, 2018. 14: p. 93-100.

56. Baldi, P., P. Sadowski, and D.J. Whiteson, “Searching for exotic particles in high-energy physics with deep learning”, ***Nature Communications***, 2014. 5(1): p. 1-9.

57. İnternet: UCI repository of machine learning databases, University of California, Irvine. ***http://www. ics. uci.edu/mlearn/MLRepository. Html***(2020).

58. Aljarah, I. and S.A. Ludwig. “Parallel particle swarm optimization clustering algorithm based on mapreduce methodology”. ***Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC)***, 2012.

59. Barney, B.J., “Introduction to parallel computing”, ***Lawrence Livermore National Laboratory***, 2010. 6(13): p. 10.

ÖZGEÇMİŞ

Şüheda Semih AÇMALI ilk ve orta öğretimini Zonguldak ve Karabük'te tamamladı. Safranbolu Lisesi'nden 2010 yılında mezun oldu. 2011 yılında Selçuk Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü'nde öğrenime başlayıp 2017 yılında mezun oldu. 2018 yılında Karabük Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda başlamış olduğu yüksek lisans eğitimini, Karabük Üniversitesi Lisansüstü Eğitim Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı altında sürdürmektedir.