MapReduce ile METASEZGİSEL OPTİMİZASYON

Muhammed Emre ÇOLAK

Yüksek Lisans Tezi

Yazılım Mühendisliği Anabilim Dalı
Danışmanı: Prof. Dr. Asaf VAROL

NİSAN- 2015

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MapReduce ile METASEZGİSEL OPTİMİZASYON

YÜKSEK LİSANS TEZİ

MUHAMMED EMRE ÇOLAK
(132137105)

Tezin Enstitüye Verildiği Tarih : 28.04.2015
Tezin Savunulduğu Tarih : 24.04.2015

Tez Danışmanı : Prof. Dr. Asaf VAROL (F.Ü.)
Diğer Jüri Üyeleri : Doç. Dr. Bilal ALATAŞ (F.Ü.)
Doç. Dr. Ali KARCI (İ.Ü.)

NİSAN-2015

ÖNSÖZ

Tez çalışmasında, optimizasyon, optimizasyon tarihi, metasezgisel algoritmalar, MapReduce sistemi ve optimizasyon konusundaki güncel çalışmalar incelenmiş, "Dairesel Su Dalgaları" adlı yeni ve özgün bir optimizasyon algoritması tasarlanmış ve bu algoritma MapReduce platformunda pseudo-distributed olarak işletilmiştir.

Yenilikçi fikirlerin daima destekçisi olan ve fikirleri ile de her zaman bana destek veren, yardımlarını esirgemeyen çok değerli ve saygıdeğer hocam, danışmanım Sayın Prof. Dr. Asaf VAROL'a ebedi teşekkürlerimi sunuyorum.

Yardımlarıyla beni yönlendiren ve destek veren saygıdeğer hocam Doç. Dr. Bilal ALATAŞ'a candan teşekkürlerimi sunuyorum.

Muhammed Emre ÇOLAK
ELAZIĞ-2015

İÇİNDEKİLER

Sayfa No

ÖZET

Gelişen dünyada, iyileştirme ihtiyacı duyulan fonksiyon sayısı ve bu fonksiyonlara ait parametreler gün geçtikçe çoğalmakta ve bu parametrelerin optimal değerlerinin bulunması büyük önem arz etmektedir. Bu hususta metasezgisel optimizasyon algoritmaları geniş kullanıma sahip algoritmalardır. Ancak metasezgisel algoritmaların, tek bilgisayar üzerinde çalışan algoritmalar olmaları vesilesiyle, dağıtık olarak işletilmesi halinde etkinliğinin arttırılabileceği öngörülmektedir. Bu noktada çeşitli dağıtık uygulamalar bulunmasına rağmen, MapReduce kullanılarak yapılmış bir uygulamanın bulunmadığı gözlemlenmiştir. MapReduce ile dağıtık olarak metasezgisel optimizasyon uygulamasının gerçekleştirilmesi halinde optimizasyon algoritmalarının daha büyük problemleri daha az sürede çözme kabiliyetine kavuşacağı öngörülmektedir.

Bu tez çalışmasında optimizasyon konusu incelenmiş, metasezgisel optimizasyon algoritmalarına birkaç örnek verilmiş, yeni bir optimizasyon algoritması tasarlanmış, MapReduce altyapısına giriş yapılmış ve son olarak da MapReduce ile yeni tasarlanan optimizasyon algoritmasının uygulanmasına yönelik bir çalışma verilmiştir.

Anahtar Kelimeler: MapReduce, Metasezgisel Algoritmalar, Optimizasyon, Dağıtık İşleme

SUMMARY

Metaheuristic Optimization with MapReduce

In today's world, number of the functions that needs to be optimized and parameters of these functions are increasing and optimization of these parameters has significant importance, in this regard; a metaheuristic optimization algorithm has broad usage. However these algorithms can be more effective if they can work in distributed environment. It has been noticed that such a distributed optimization applications exist but an application in MapReduce platform has not be done yet. In this regard MapReduce framework has been chosen to work with. With MapReduce framework bigger problems may be solved with lesser cost and time.

In this thesis, optimization is inspected, some of the metaheuristic algorithms are explained, a novel optimization algorithm is designed, an introduction to MapReduce framework and finally an optimization application with MapReduce framework is provided.

Key Words: MapReduce, Metaheuristic Algorithms, Optimization, Distributed Work.

ŞEKİLLER LİSTESİ

Sayfa No

TABLOLAR LİSTESİ

Sayfa No

KISALTMALAR

ABC	:	Artificial Bee Colony – Yapay Arı Kolonisi
ACO	:	Ant Colony Optimization- Karınca Koloni Optimizasyonu
CoV	:	Calculus of Variations – Değişkenlerin Kalkülüsü
CWW		Circular Water Waves – Dairesel Su Dalgaları
WWO		Water Wave Optimization – Su Dalga Optimizasyonu
GFS	:	Google File System – Google Dosya Sistemi
HDFS	:	Hadoop File Sistem – Hadoop Dosya Sistemi
PSO	:	Particle Swarm Optimization – Parçacık Sürü Optimizasyonu
YARN	:	Yet Another Resource Negotiator – Hadoop Kaynak Dağıtıcısı

SEMBOLLER LİSTESİ

v	**:** Hız
j	**:** Boyut
i	**:** Parçacık
c	**:** Bilişsel pozitif ivma sabiti
w	**:** [0,1] aralığında uniform dağılımla oluşturulan rassal sayılar
x	**:** Asil üyenin yönelme öncesi değeri
t	**:** Zaman, işlem adımı
S	**:** Sürü
y	: Global en iyi konum
y_g	: Lokal en iyi konum
N	: Komşuluk
τ	: Feromon yoğunluk fonksiyonu
g	: Görüş fonksiyonu
ρ	: Feromon buharlaşma katsıyısı
D	: Optimize edilecek parametre sayısı
p	: Kaynağın seçilme ihtimali
r	: Mesafe
I	: Işık yoğunluğu
$ß$	: Ateş böceği çekimi
K	: Anahtar
V	: Değer
n	: Değişken sayısı
d	: Bir sonraki değerden farkı
r	: Yarıçap
C	: Dalga daire sayısı
M	: Maksimum dalga sayısı
b	: Yeni başlangıç noktalarının sayısı
a	**:** Daire dalga sırası
h	: Dalga boyutu

EKLER LİSTESİ

Sayfa No

1. GİRİŞ

Optimizasyon günümüz dünyasında önemli yere sahip bir alandır. Optimizasyon demek; zaman, mekân, maliyet, hammadde ve bunun gibi sınırlı kaynaklardan tasarruf edilmesi anlamını taşımaktadır. Yapılacak işlere ait fonksiyonların ve bu fonksiyonlara ait parametrelerin artışı, en uygun değerlerin kişilerce bulunmasını imkânsız hale getirmiş ve böylece çeşitli hesaplama algoritmalarının kullanımının önü açılmıştır. Bu hususta metasezgisel optimizasyon algoritmaları geniş kullanıma sahip algoritmalardır. Ancak metasezgisel algoritmaların, tek bilgisayar üzerinde çalışan algoritmalar olmaları vesilesiyle, dağıtık olarak işletilmesi halinde etkinliğinin arttırılabileceği öngörülmektedir. Bu noktada çeşitli dağıtık uygulamalar bulunmasına rağmen MapReduce kullanılarak yapılmış bir uygulama bulunmadığı gözlemlenmiştir.

MapReduce ile dağıtık olarak metasezgisel optimizasyon uygulamasının gerçekleştirilmesi halinde, optimizasyon algoritmalarının daha büyük problemleri daha kısa sürede çözme kabiliyetine kavuşacağı öngörülmektedir.

2. OPTİMİZASYON

Bu bölümde optimizasyon kavramı, optimizasyonun tarihsel gelişimi, bazı metasezgisel algoritmalar ve metasezgisel algoritmalara yönelik bazı çalışmalara yer verilmiştir.

2.1. Optimizasyon Nedir?

Optimizasyon ya da kısıtlanmış optimizasyon ya da matematiksel programlama; kıt kaynakların en iyi ve kazançlı şekilde kullanım yönteminin belirlenmesi olarak tanımlanabilir. Optimizasyon konusu eski çağlardan günümüze kadar ilim dünyasında kendine yer edinmiş bir konudur. Günümüzde artık metasezgisel algoritmalar hemen hemen her alanda optimizasyon işleri için kullanılmaktadır.

2.2. Optimizasyonun Tarihsel Gelişimi

Beyin Metasezgisel algoritmaların bulunmasına kadar optimizasyonun tarihçesine ait süreç aşağıda sunulmuştur [1].

Eski Uygarlıklarda:

Yunan matematikçiler geometrik çalışmalarıyla ilgili bazı optimizasyon problemlerini çözmüşlerdir.

- M.Ö 300 Euclid nokta ile çizgi arasındaki minimum uzaklığı bulmuş ve dikdörtgenin toplam kenar boyutu aynı olan dörtgenler arasında en geniş yüz ölçümüne sahip olduğunu ispatlamıştır.
- M.Ö. 200 Zenodorus (Pappus ve Theon'un belirttiğine göre) Dido problemini (Dido's problem) çalışmıştır.
- M.Ö. 100 Heron Catoprica'sa ışığın aynadan yansırken iki nokta arasındaki en kısa mesafeden gittiğini ispatlamıştır.

17. ve 18. Yüzyıllarda:

Kalkulus ve varyasyonların icadından önce sadece bazı ayrık optimizasyon problemleri araştırılmıştır.

- 1615 J. Kepler şarap varilleri için en iyi boyutları bulmuştur. Ayrıca yeni bir eş arayışına başladığı sırada sekreter probleminin (dinamik programlamanın klasik bir uygulaması) ilk versiyonunu formüle etmiştir.

- 1638 G. Galileo asılan zincirin şeklini bulmaya çalışmış ancak, başarışız olmuştur.
- 1646 P. de Fermat ekstrem noktalarda fonksiyonların eğimlerinin kaybolduğunu göstermiştir. 1657 yılında Fermat ışığın iki nokta arasını en az sürede (en kısa yoldan) geçtiğini göstermiştir.
- Newton (1660) ve G. W. Von Leibniz (1670) kalkülüs ve varyasyonların (calculus of variations - CoV) temellerini atmışlardır. Bazı sonlu ayrık optimizasyon problemlerini de düşünmüşlerdir. 1687 yılında Newton en az dirençli gövdeyi çalışmıştır.
- 1696 yılında Johann ve Jacob Bernoulli Brachistochrone problemini çalışmış ve böylece varyasyonların kalküsü (calculus of variations-CoV) doğmuştur.
- 1712 yılında J.S. König bal peteği şeklinin en iyi şekil olduğunu göstermiştir. Fransız Bilim Akademisi (The French Academy of Sciences) bu olayı ilahi yol gösterme olarak beyan etmiştir.
- 1740 yılında L. Euler bir yayını ile varyasyonların kalkülüsünün genel teorisi üzerine araştırmalar başlatmıştır.
- 1746 yılında P.L.M. de Maupertuis fiziksel olayları açıklamak için en az hareket prensibini (principle of least action) formüle etmiştir.
- 1754 yılında J.-L. Lagrange CoV dâhilinde ilk buluşlarını 19 yaşında yapmıştır. 1760 yılında Plateau'nun problemini (Plateau's problem) yani en az yüzeyler problemini formülize etmiştir.
- 1936 yılında J. Douglas bir problem çözümü ile Fields Madalyası almıştır, 1974 yılında E. Bombieri de bu konudaki çalışması ile Fields Madalyası almıştır.
- 1784 yılında G. Monge ulaşım problemini (combinatorial optimization problem known as the transportation problem) incelemiştir.

19. Yüzyılda:

İlk optimizasyon algoritmaları 19. Yüzyılda sunulmuştur. K.T.W. Weierstrass, J. Steiner, W.R. Hamilton ve C.G.J. Jacobi CoV'yi daha da geliştirmişlerdir.

- 1806 yılında A.-M. Legendre küçük kareler (least square) metodunu sunmuştur ki J.C.F. Gauss kendinin bulduğunu iddia etmiştir. Legendre aynı zamanda CoV alanında da katkılarda bulunmuştur.
- 1815 yılında T.R. Malthus, R. Torrens, E. West ve D. Ricardo eş zamanlı olarak üretim fonksiyonları için "Azalan Dönüşler Kanunu" ("the Law of Diminishing

Returns") ortaya atmış ve (quasi) konkav (iç bükey) fonksiyonları ekonomide görünmeye başlamıştır.

- 1826 yılında J.B.J. Fourier mekanik ve ihtimal teorisinde ortaya çıkan problemleri çözmek için LP-problemlerini formül haline getirmiştir.
- 1847 yılında A.L. Cauchy gradient metodunu sunmuştur.
- 1857 yılında J.W. Gibbs kimyasal dengenin enerjisinin minimumu olduğunu göstermiştir.
- 1870'li yıllarda ekonomideki marjinalist devrim sonucunda optimizasyon ekonomik teorinin bir iç parçası haline gelmiştir.

20. yüzyılda:

20. yüzyılda CoV daha da geliştirilmiştir. Başlıca geliştiricileri; O. Bolza, C. Caratheodory ve G.A. Bliss'dir.

- 1902 yılında J. Farkas, Karush-Kuhn-Tucker teoremini de ispat eden meşhur teoremini sunmuştur.
- Konveksite konsepti oluşturulmuştur: J.L.W.V. Jensen 1905 yılında konveks fonksiyonları tanıtmış, J. S. Hadamard 1889 yılında yaptığı çalışmalarında bu fikirden bahsetmiştir.
- 1917 yılında H. Hancock optimizasyon üzerine ilk kitabı olan "Theory of Minima and Maxima" kitabını yayınlamıştır.
- Biyomatematikçi D.W. Thompson "on Growth and Form" kitabını 1917 yılında yayınlamıştır. Bu kitapta canlı organizmaların formlarını analiz etmek için optimizasyon uygulamıştır.
- 1925 yılında H.C.M. Morse CoV'yi genelleştiren teorisini sunmuştur.
- 1928 yılında F.P.P Ramsey optimal ekonomik büyüme çalışmasında CoV'yi uygulamış, Ramsey'in çalışması 1950'li yıllarda yeniden canlanarak optimal büyüme teorisi alanı haline gelmiştir.
- 1931 yılında J. Viner Viner-Wong torba (envelope) teoremini sunmuştur.
- 1932 yılında K. Menger gezgin satıcı probleminin genel formülasyonunu sunmuştur.
- 1939 yılında L.V. Kantorovich LP-modeli ve çözümü için algoritma sunmuştur.
- 1975 yılında Kantorovich ve T.C. Koopmans LP-problemine katkılarından dolayı Nobel Ekonomi Ödülünü kazanmıştır.

II. Dünya savaşından sonra optimizasyon eşzamanlı olarak operasyon araştırmalarında geliştirilmiştir. J. Von Neumann operasyon araştırmalarının ardındaki önemli bir karakterdi. Elektronik hesaplamaların geliştirilmesiyle algoritmik araştırma genişlemiştir.

- 1944 yılında J. Von Neuman ve O. Morgenstern dinamik programlama (DP) fikrini kullanarak ardışıl karar problemlerini çözmüşlerdir. A. Wald (1947) bununla ilişkili bir araştırma yapmıştır. Bir başka erken DP uygulaması P. Massé (1944) tarafından rezervuar yönetimi için sunulmuştur.
- 1947 yılında Amerikan hava kuvvetleri için çalışan G. Dantzig, LP problemlerinin çözümü için Simplex metodunu sunmuş, Von Neumann LP-problemleri için duallik teoremini (theory of duality) vermiştir.
- 1949 yılında ilk uluslararası kongre "International Symposium on Mathematical Programming, on optimization" Chicago'da yapılmıştır.
- 1950'li yıllarda:
- 1951 yılında H.W. Kuhn ve A.W. Tucker lineer olmayan problemler için en iyilik şartlarını yeniden bulmuşlardır. F. John 1948 yılında ve W. Karush 1939 yılında benzer şartlar sunmuşlardır.
- 1951 yılında H. Markowitz portföy teorisini sunmuştur. Bu teori kuadratik optimizasyona dayanıyordu. 1990 yılında Markowitz ekonomi alanında Nobel ödülü almıştır.
- 1954 yılında L.R. Ford'un ve D.R. Fulkerson'ın ağ problemleri üzerindeki araştırmaları kombinatoryal optimizasyonun başlangıç noktası olmuştur.

Sınırsız problemler için (örneğin quasi-Newton ve conjugate gradient method) algoritmalar geliştirilmiştir.

Optimal kontrol teorisi CoV 'den ayrı bir disiplin olarak gelişmeye başlamıştır. Uzay yarışı optimal kontrol teorisinin araştırılmasına ek bir hız vermiştir.

- 1954 yılında IEEE Control Systems Society kurulmuştur.
- 1956 yılında L.S. Pontryagin'in araştırma grubu maksimum prensibini sunmuştur.
- 1957 yılında R. Bellman en iyilik prensibini sunmuştur.

1960'lı yıllarda:

- Zoutendijk (1960) fizibil yönler metodunu (the methods of feasible directions) sunarak lineer olmayan problemler için Simplex metodunu genelleştirmiştir. Rosen, Wolfe ve Powell benzer fikirler geliştirmişlerdir.

- 1963 yılında Wilson, 1975 yılında Han ve 1977 yılında Powell tarafından sıralı kuadratik programlama metodu keşfedilmiştir.

1970 ve sonrasında:

- 1973 yılında Mathematical Programming Society kurulmuştur.
- 1984 yılında N. Karmarkar'ın LP-problemleri için polinomal zaman algoritması ortaya çıkmıştır. İlk LP polinomal zaman algoritması; elipsoit metodu Khachiyan tarafından 1979 yılında sunulmuştur.

1960 ve 70'li yıllarda geliştirilen komplekslik analizi, optimizasyon teorisini etkilemeye başlamıştır.

1980'li yıllarda bilgisayarların daha verimli olmasıyla birlikte evrensel optimizasyon için sezgisel algoritmalar ve geniş ölçekli problemler popülerlik kazanmaya başlamıştır.

1990'lı yıllarda iç nokta (interior point) metodu kullanımı yarı tanımlı (semi definite) optimizasyona genişlemiştir.

Optimizasyonun tarihsel gelişimi genel olarak incelendiğinde; optimizasyonun spesifik problemler üzerinde iyileştirme çabalarıyla başlayıp, problemlerin matematiksel formülizasyonuna ve genelleştirilmesine doğru ilerlediği görülmektedir. Mevcut problemlere matematiksel çözüm yollarının üretilmesinden bu çözümlerin algoritmik hale getirilmesine ve zamanla sezgisel yaklaşımların kullanılmasına doğru yöneldiği gözlemlenmektedir. Bilgisayarın yüksek işlem yapabilme kapasitesi bu sezgisel yaklaşımların kullanılarak kabul edilebilir hata paylarıyla çözüme ulaşılmasına imkân vermiştir. Artık işlem kapasitelerinin dağıtık işletme yollarıyla arttırarak daha büyük problemleri çözmenin önü açılmıştır.

2.3. Metasezgisel Optimizasyon Algoritmaları

Pek çok metasezgisel algoritma ilim dünyasına sunulmuş durumdadır. Her algoritmanın güçlü ve zayıf yönleri mevcut olmakla birlikte henüz her şartta en iyi sonucu veren bir algoritma bulanabilmiş değildir. Bununla birlikte probleme uygun algoritmaların seçimiyle günümüz problemlerinin hemen hemen hepsine çözüm bulunabilmektedir. Günümüzde genel kabul gören metasezgisel algoritmalardan bazıları aşağıda açıklanmıştır.

2.3.1. Parçacık Sürü Optimizasyon Algoritması

Kennedy, J. ve Eberhart, R. tarafından 1995 yılında Parçacık Sürü Optimizasyonu (Particle Swarm Optimization) adıyla Neural Networks, 1995. Proceedings., IEEE

International Conference on (Volume:4) 1942 – 1948 sayfalarında yayınlanmıştır [2]. Kısaca PSO olarak adlandırılır. Kuş sürülerinin besin arayışlarının gözlemlenmesi sonucunda ortaya çıkmıştır. Programda kuş yerine fonksiyon değerlerini taşıyan parçacıklar kullanılmıştır. Bu parçacıkların iteratif olarak ve birbiriyle ilişkili olarak (evrensel en iyi parçacık değeri tüm parçacıklara bildirilir) en iyi değere doğru ulaşmaya çalışmaktadırlar.

Kullanılan formülizasyon oldukça açık ve kolay anlaşılır bir yapıya sahiptir. Her parçacığın bir konumu ve bir hareket hızı vardır. Parçacıkların ilk konumları rassal olarak atanır. Daha sonra parçacığın *t* zamanından sonraki konumunun bulunması için hız ve önceki konum bilgileri kullanılır. Hızın hesaplanması PSO'daki belki de en önemli noktadır. PSO üzerinde yapılan çalışmalarda parçacık sayısı, gruplanması üzerinde de durulmakla birlikte yapılan geliştirmelerin üzerinde durduğu önemli bir nokta hız fonksiyonudur. PSO hız fonksiyonu için;

$$v_{ij}(t+1) = v_{ij}(t) + c_1 w_{1j}(t)\,[\,y_{ij}(t) - x_{ij}(t)\,] + c_2 w_{2j}(t)\,[\,y_{gij}(t) - x_{ij}(t)\,] \qquad (1)$$

formülü kullanılır. Burada v_{ij} (t), t zamanında i. parçacığın j. boyuttaki hızıdır. c_1 ve c_2 evrensel ve lokal parçacıkların katkısını ölçeklemeye yönelik bilişsel pozitif ivme sabitleridir. w_{1j} ve w_{2j} [0,1] aralığında üniform dağılımla üretilen rasgele sayılar olup algoritmaya stokastik özellik sağlamaktadır. y iterasyonların başından itibaren parçacığın bulduğu en iyi pozisyonu ifade etmektedir.

Hız formülü uygulandıktan sonra konumun bulunması için;

$$x_i(t+1) = x_i(t) + v_i(t+1) \qquad (2)$$

formülü kullanılır. Burada x konum fonksiyonu, i parçacık numarası, v hız fonksiyonu ve t de zaman yani işlem adımı olarak kabul edilir. Dikkat edilirse bu formül temel fizik hareket formülüdür.

PSO genel olarak iki ana amaca yönelik olarak özelleşmiştir. Birincisi evrensel en iyiyi bulma (Global En iyi PSO ya da kısaca Gbest-PSO), ikincisi lokal en iyiyi bulma (Lokal En iyi PSO ya da Lbest-PSO). Gbest-PSO tüm parçacıkların bulduğu en iyi konumu (1) formülündeki y_g olarak kabul eder. Öte yandan Lbest-PSO (1) formülündeki y_g için kendi sürüsünün bulduğu en iyi değeri kabul eder, yani sürünün tamamının alt sürülere ayrıldığı

ve her alt sürünün kendi içinde ayrı sürüymüş gibi davrandığı düşünülür. Yani Lbest-PSO'da parçacığın komşulukları mevcuttur ve komşulukların bulduğu en iyi değer formülde y_g olarak ifade edilir. Söz konusu algoritmalar için sözde kodlar;

Tablo 2.1. PSO algoritmasının sözde kodları [2]

Gbest-PSO	**Lbest-PSO**
1. n_x boyutlu S sürüsü oluşturulup parametrik değerler atanır. *Tekrar:* *2. Her parçacık için (i= 1,2,3,….,n);* *Parçacığın en iyi pozisyonunu belirle:* *Eğer $f(S.x_i) < f(s.y_i) \Rightarrow S.y_i = S.x_i$* *Küresel en iyi pozisyonu belirle:* *Eğer $f(S.x_i) < f(s.y_g) \Rightarrow S.y_g = S.x_i$* *$y_g$ evrensel en iyi y değeri.* *3. Her parçacık için (i= 1,2,3,….,n);* *Denklem 1 ile hız güncellemesini gerçekleştir.* *Denklem 2 ile pozisyon güncellemesini gerçekleştir.* *4. Durdurma Koşulunu kontrol et sağlanmazsa tekrar et.*	*1. n_x boyutlu S sürüsü oluşturulup parametrik değerler atanır.* *Tekrar:* *2. Her parçacık için (i= 1,2,3,….,n);* *Parçacığın en iyi pozisyonunu belirle:* *Eğer $f(S.x_i) < f(s.y_i) \Rightarrow S.y_i = S.x_i$* *Küresel en iyi pozisyonu belirle:* *Eğer $f(S.x_i) < f(s.y_g) \Rightarrow S.y_g = S.x_i$* *$y_g$ komşulukların en iyi y değeri.* *3. Her parçacık için (i= 1,2,3,….,n);* *Denklem 1ile hız güncellemesini gerçekleştir.* *Denklem 2 ile pozisyon güncellemesini gerçekleştir.* *4. Durdurma Koşulunu kontrol et sağlanmazsa tekrar et.*

Lbest-PSO'da y_g değerinin ve N_i komşuluğunun hesaplanması;

$$y_g(t+1) \in \{ Ni \mid f\left(yg_i\ (t+1)\right) = min\{f(x)\}, \forall x\ \in\ N_i \tag{3}$$

$$N_i = \{\ y_{i-nN_i}(t), y_{i-nN_i+1}(t), \dots, y_{i+N_i-1}(t), y_{i+nN_i}(t) \tag{4}$$

Dikkat edilirse N_i değeri için n_s alınırsa, komşuluk tüm parçacık sürüsü haline gelir ki bu durumda Lbest-PSO Gbest-PSO haline gelir.

PSO algoritması sürekli fonksiyonlarda optimizasyona yönelik, başarılı bir algoritmadır. Kesikli optimizasyon problemlerinde uygulanmasına yönelik, çeşitli uygulamalar mevcut olmasına rağmen, sürekli problemler için daha uygun bir algoritma olduğu düşünülmektedir.

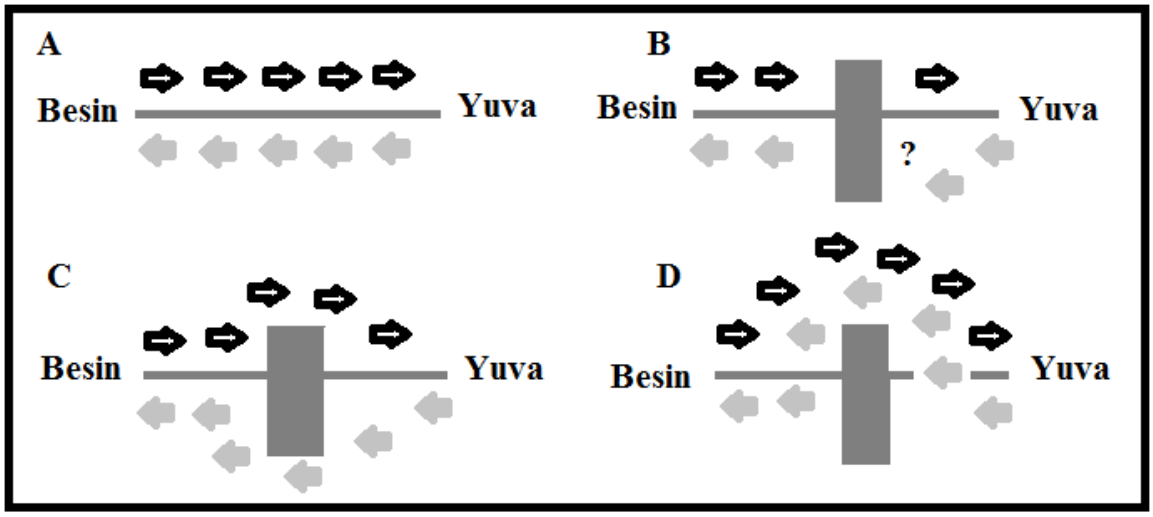
2.3.2. Karınca Koloni Optimizasyon Algoritması

Karınca Koloni Optimizasyon Algoritması (Ant Colony Optimization Algorithm) karınca kolonisinin davranışlarından esinlenilerek geliştirilmiş bir algoritmadır. Drigo tarafından 1991 yılında ortaya konulmuştur [3]. Karıncaların feromon kullanarak besin yollarını bulmaları üzerine dayalı bir mantığa sahiptir. Doğada karıncaların davranışları şöyle özetlenebilir;

- Karıncalar başta rassal bir şekilde hareket ederek gezer ve bir feromon izi bırakırlar.
- Eğer belli bir yoğunluktan daha yoğun bir feromon izi bulurlarsa bu izi takip ederler.
- Eğer bir yiyecek bulurlarsa yiyeceği yuvaya götürürler ki bu da bir feromon izi bırakmalarına yol açar. Yiyeceği taşırken ki hızları normal gezintilerinden daha yavaş olduğundan, bıraktıkları feromon yoğunluğu artar.
- Yuvaya girdikten sonra tekrar yukardaki adımları takip ederler.

Karıncalar geçtikleri yollarda doğal olarak feromon bırakırlar. Karınca bir besin bulduğunda en kısa yoldan geri döner ve tekrar besini almak üzere en kısa yoldan besine gider. Bu gidiş gelişleri yol üzerindeki feromon miktarını arttırır ve karıncalar yol tercihi yaparken en yüksek feromon değerine sahip yolu tercih etme eğilimindedirler.

Feromonun önemli bir özelliği zamanla azalmasıdır. Aksi takdirde bir yiyecek kaynağında yiyecek kalmadığı zaman dahi karıncalar o kaynağa yönelirler. Ancak, zamanla feromon buharlaştığı için artık kıymeti kalmayan (bitmiş) yiyecek kaynaklarına giden yollarda unutulmaktadır. Feromonların buharlaşmasının başka faydaları da bulunmaktadır, aşağıdaki şekil göz önüne alınacak olursa;



Şekil 2.1. Karıncalarda yuvaya yemek taşıma davranışları [4]

Şekil 2.1.'de; A durumu bir yiyecek kaynağı ile yuva arasında gidip gelen karıncaları göstermektedir. B durumunda yol üzerine bir engel yerleştirilmiştir. Bu durumda karıncalar hangi yolun daha kısa olacağını bilmediklerinden; karıncaların bir kısmı engelin sağ tarafından geçerken, diğer kısmı sol tarafından geçmektedir. Dikkat edilirse sol (yani üst) taraf daha kısadır. Buradan karıncalar daha hızlı geçeceklerdir. Böyle belli bir zaman içinde bu yolu tamamlayan karınca sayısı daha fazla olacağından, buradaki feromon miktarının yoğunluğu daha yüksek olacak ve zamanla karıncaların hepsi yoğun olan tarafta ilerlemeye başlayacaklardır. Mesafe arttıkça feromon miktarının düşmesi hem daha yakın kaynaklara yönelmeyi sağlarken hem de kaynağa giden en kısa yolun bulunmasına imkân sağlamaktadır.

Karınca koloni algoritması graflar için tanımlanmış bir algoritmadır ve graflar üzerinde çalışır. Sınırlı sayıda çözümün olduğu ayrık problemler için etkindir ve bu tarz graflara yapı grafı (construction graph) adı verilir. Karıncalar bu graf üzerinde düğümden düğüme kenarlar üzerinden dolaşarak çözüm bulmaya çalışırlar.

Tablo 2.2. Karınca algoritması sözde kod [3]

1. *Yapı grafını oluştur.*
2. *Feromon için ilk değerleri ata*
3. *Durma şartına kadar devam eden adımları tekrarla*
4. *Tüm karınca çözümlerini oluştur.*
5. *Feromon değerlerini güncelle*

Tablo 2.2. ile gösterilen sözde kodda bulunan çözüm oluşturma adımı (4 nolu adım) probleme özel olup mevcut karıncanın mevcut durumuyla bulduğu sonucu ifade etmektedir. Kenar seçimi;

$$ihtimal(e\ kenarının\ seçimi) = \frac{\tau(e).g(e)}{\sum_{seçilebilecek\ e\ kenarları} \tau(e')n(e')} \quad (5)$$

formülüyle yapılır. τ(e) önceki eylemlerden oluşan feromon yoğunluğunu ifade ederken, *g(e)* verilen problem için görüş fonksiyonudur (örneğin mesafe).

Feromon değerlerinin güncellenmesi ise yapılan karınca eylemleri sonucunda bıraktıkları feromonların yollara eklenmesini ve geçen zamandan dolayı belli miktardaki Feromonun buharlaştırılmasını içerir.

$$\tau(e) \coloneqq \begin{cases} (1-\rho).\tau(e), eğer\ karınca\ kenardan\ geçmemişse \\ (1-\rho).\tau(e) + \Delta\ yeni\ feromon\ , eğer\ karınca\ kenardan\ geçmişse \end{cases} \quad (6)$$

$0<\rho<1$ aralığındaki parametre buharlaşma katsayısı olarak kabul edilir.

Feromonlar; karınca kolonisinin hafızasını taşımaktadır. Eğer ρ küçük tutulursa, düşük seviyede bir buharlaşma oluşur ki bu yeni bulgulara karşı yavaş adaptasyona sebebiyet verir. Eğer ρ büyük olursa buharlaşma hızlı olur, bu da hızlı adaptasyonu sağlar.

Karınca koloni algoritması graflar üzerinde etkin çalışan ve genel kabul görmüş bir algoritma olup sınırlı sayıda çözümün olduğu ayrık problemler için kullanılmaktadır.

2.3.3. Yapay Arı Koloni Algoritması

Yapay Arı Kolonisi (Artificial Bee Colony – ABC) gerçek arıların yiyecek arama davranışlarından yola çıkılarak geliştirilmiştir. Derviş Karaboğa tarafından 2005 yılında önerilmiştir [5]. Gerçek dünyada arılar temelde 3 kategoridedir. Bunlar doğurgan dişi arı yani kraliçe, dişi arıyla çiftleşmekle görevli erkek arı ve kısır dişi arı yani işçi arılardır. İşçi arılar besin bulma konusunda görevlidirler ve yaptıkları işlere göre sınıflandırılmakla birlikte bir işçi arı şartlara bağlı olarak herhangi bir görevi yüklenebilir. Temelde besin kaynağı aramak için kovan dışında gezen arılara kâşif arı denilir, bir besin kaynağından yiyecek getiren arıya görevli arı denilir, kâşif arılar iyi bir besin kaynağı bulduklarında bu besin kaynağını kovanda bekleyen arılara dans yoluyla öğretirler, kovanda bekleyen ve dansı izleyen arılara gözcü arılar denir.

Gözcü arılar uygun bir besin kaynağı öğrendiklerinde görevli arı haline gelerek besin kaynağından besin taşımaya başlarlar. Algoritmada görevli, kâşif ve gözcü arıların oluşturularak bu arılara farklı görevlerin yüklenmesi öngörülmüştür. Arıların görevleri değişebilmektedir, örneğin bir kâşif arı, işçi arıya dönüşebilmektedir. Kâşif arı yiyecek kaynağı aramakla görevliyken (optimal nokta içermesi muhtemel alan), işçi arılar kâşif arının bulduğu yiyecek kaynağında (optimal nokta içermesi muhtemel alanda) yiyecek toplamakla (optimal noktayı aramakla) görevlidirler. Gözcü arılar mevcut durumu gözleyerek ihtiyaç halinde kâşif arı ya da işçi arı olarak görev yapmaktadırlar. ABC algoritmasında bazı ön tanımlar mevcuttur. Bu ön tanımlar her yiyecek kaynağında sadece bir işçi arının görevli olması, işçi ile gözcü arı sayısının eşit olmasıdır.

Tablo 2.3. ABC algoritması için sözde kod [5]

1. *Başlangıç yiyecek kaynaklarının oluşturulması*
2. *Durma şartına kadar devam eden adımları tekrarla*
3. *Görevli arıların yiyecek kaynaklarına gönderilmesi*
4. *Görevli arılardan gelen verilerle olasılıksal seleksiyonların belirlenmesi*
5. *Gözcü arıların olasılık değerlerine göre yiyecek kaynağı bölgesi seçmesi*
6. *Yetersiz kaynak bölgelerinin bırakılması ve kâşif arı oluşturulması*

Başlangıç yiyecek kaynaklarının oluşturulması:

$$x_{ij} = x_{jmin} + rand(0,1)(x_{jmax} - x_{jmin}) \quad (7)$$

Burada i=1,…,SN , j=1,…,D olup SN kaynak sayısını D ise optimize edilecek parametre sayısını göstermektedir. *min.* ve *max.* ile ifade edilen parametrenin alt ve üst sınırlarıdır. (7) formülü kullanılarak ilk yiyecek kaynakları oluşturulur.

Görevli arıların yiyecek kaynaklarına gönderilmesi:

Yeni kaynağın belirlenmesi için formül (8) kullanılır.

$$y_{ij} = x_{ij} + \theta_{ij}(x_{ij} - x_{kj}) \quad (8)$$

x_i i. kaynağı, j ise bu kaynağa ait rastgele seçilen j. parametreyi ifade eder. Böylece x_i kaynağının komşuluğundaki y_i kaynağı elde edilir. (8) formülündeki j [0, D] aralığında rastgele seçilen bir tamsayıyı ifade etmektedir. θ_{ij} [0, 1] aralığındaki rastgele bir katsayıdır.

v_i kaynağı için uygunluk fonksiyonu;

$$uygunluk_i = \begin{cases} 1/(1+f_i) & , f_i \geq 0 \\ 1 + abs(f_i) & , f_i < 0 \end{cases} \quad (9)$$

formülüyle hesaplanır. Burada f söz konusu problem için v_i kaynağının maliyetidir. Eğer v_i kaynağı daha düşük maliyete sahipse görevli arı eski kaynağı hafızasından siler ve v_i kaynağını hafızasına alır, eğer daha yüksek çıkarsa görevli arı x_i kaynağında çalışmaya devam eder ancak, x_i kaynağı için başarısızlık parametresinin değerini bir arttırır. Eğer bir kaynağın başarısızlık parametresi belli bir değere ulaşırsa o kaynak terk edilir. O kaynakta görevli arı kâşif arı olur.

$$p_i = \frac{uygunluk_i}{\sum_{j=1}^{SN} uygunluk_j} \quad (10)$$

Gözcü arıların seçeceği kaynakların belirlenmesi:

Formülde uygunluk fonksiyonu; söz konusu kaynağın kalitesini, SN görevli arı sayısını p_i ise kaynağın seçilme ihtimalini ifade etmektedir. Orijinal algoritmada hesaplanan p_i değerleri kullanılarak rulet tekerleği yöntemiyle arılar kaynaklara atanır.

Tükenen kaynakların terk edilmesi:

Algoritmanın bir döngüsü tamamlandıktan sonra, başarısızlık parametreleri kontrol edilir. Eğer başarısızlık parametresi belirlenmiş limitin üzerine çıkan bir kaynak var ise bu kaynak terk edilir ve kaynakta görevli arı kâşif arı statüsüne geçerek (7) eşitliği yardımıyla yeni kaynak aramaya başlar.

ABC algoritması için tavsiye edilen parametre değerleri; koloni büyüklüğü 20 ila 50 arası limit için parametre sayısı ile koloni büyüklüğünün çarpımıdır. ABC algoritması literatürde kendine yer edinmiş etkin bir metasezgisel algoritma olup pek çok uygulamada kendine kullanım alanı bulmuştur.

2.3.4. Ateşböceği Algoritması (Firefly Algorithm)

Ateşböceği Algoritması (Firefly Algorithm) Xin-She Yang tarafından, 2008 yılında ateşböceği davranışlarından esinlenilerek geliştirilmiştir [6]. Ateşböceklerinin çiftleşme döneminde birbirlerine olan çekimlerinin ateşböceğinin yaydığı ışığın parlaklığı ve yanıp sönme ritminin ilintili olmasından yola çıkılarak tasarlanmıştır. Arama uzayına rasgele dağıtılan ateşböceklerinin (yani arama uzayındaki rastgele noktaların), fonksiyon değerlerinin hesaplanarak, en iyi değere sahip ateş böceğinin en parlak ışık değerine sahip olması ve diğer ateşböceklerinin bu noktaya yönelmesi üzerine kurulu bir metasezgisel algoritmadır.

Her iterasyonda ateşböcekleri arasında en iyi uygunluk değerine sahip ateşböceği bulunur ve geriye kalan tüm ateşböcekleri bu en iyi noktaya doğru hareket ettirilir.

Işık yoğunluğu ve çekim:

Ateşböceği algoritmasında iki önemli nokta vardır: Işık yoğunluğunun varyasyonu ve çekimin formüle edilmesi, basitlik sağlanması için çekimin parlaklıkla tanımlanması ve parlaklığın da amaç fonksiyonu ile belirlenmesi kullanılabilir.

Tablo 2.4. Ateşböceği algoritması için sözde kod [6]

Amaç fonksiyonu f(x), $x = (x_1,...,x_d)^T$
Ateş böcekleri için ilk popülasyonun oluşturulması x_i (i=1,2,...,n)
x_i noktasındaki ışık yoğunluğu I_i; $f(x_i)$ fonksiyonuyla belirlenir.
Işık emme katsayısı γ tanımlanır.
***while** (t<Maksimum Jenerasyon)*
***for** i = 1: n tüm n ateşböcekleri*
    ***for** j = 1 : n tüm n ateşböcekleri (iç döngü)*
    if**($I_i<I_j$), i. ateşböceğini j. ateşböceğine doğru hareket ettir; **end if
        r uzaklığı için exp(-γr) fonksiyonu ile çekiciliği çeşitlendir.
        Yeni çözümleri değerlendir ve ışık yoğunluğunu güncelle.
    ***end for** j*
***end for** i*
Ateşböceklerini derecelendir ve bu adım için evrensel en iyiyi (g) bul.
end while
Sonuçları çıktı olarak ver

Ateşböceklerinin ışık üretmeleri, ışıkların şiddetleri ve yanıp sönme frekansları erkek ve dişilerde ve değişik türlerde farklılık göstermektedir. Ateşböcekleri ışıklarını sadece çiftleşmek amacıyla değil, savunma ya da avlanma gibi amaçlarla da kullanabilmektedirler. Bu sebeple algoritma için bazı idealleştirmeler yapılmalıdır.

- Tüm ateşböcekleri uniseks olarak kabul edilecektir, yani ateşböcekleri birbirlerini cinsiyetten bağımsız olarak çekeceklerdir.
- Çekim ışık parlaklığıyla orantılıdır, yani daha az parlayan ateşböceği daha parlak olana doğru yönelecektir. Çekim parlaklıkla orantılıdır ve çekim ile parlaklık aradaki mesafe arttıkça azalır. Eğer bir ateşböceğinden daha parlak bir ateşböceği mevcut değil ise bu ateşböceği rassal olarak hareket edecektir.
- Ateşböceğinin parlaklığı amaç fonksiyonundan etkilenir ya da amaç fonksiyonu ile belirlenir.

Formülizasyonlar:

En basit formuyla *r* mesafeyi ifade etmek üzere ışık yoğunluğu *I*(*r*) ters kare kuralına göre hesaplanır. Formülü:

$$I(r) = \frac{I_s}{r} \tag{11}$$

I_s kaynaktaki ışık yoğunluğudur ve:

$$I = I_o e^{-\gamma r} \tag{12}$$

Burada I_0 orijinal ışık yoğunluğudur. Işığın emilimi ve ters kare kuralının ortak etkisini sağlamak için aşağıdaki Gaussian formuyla yaklaşım sağlanabilir.

$$I(r) = I_o e^{-\gamma r^2} \tag{13}$$

Ateşböceklerinin çekimleri, gördükleri yakın ateşböceklerinin ışık yoğunluğuyla orantılı olduğundan, ateşböceğinin çekim yani β değeri;

$$\beta(r) = \beta_o e^{-\gamma r^2} \tag{14}$$

formülüyle hesaplanabilir. Burada β_o, r=0 noktasındaki çekim değeridir.

2.3.5. Su Dalga optimizasyonu (Water wave optimization-WWO)

Sığ su dalga modelinden (shallow water wave model) esinlenilerek, Yu-Jun Zheng tarafından 2015 yılında geliştirilmiştir [7]. Su dalga optimizasyonu olarak adlandırılan metasezgisel algoritmada; dalga akımı ile dip etkileşiminin, dalga hareketleri üzerindeki etkilerinden esinlenilmiş ve çok boyutlu global optimizasyon problemleri için bir arama metodu dizayn edilmiştir.

WWO optimizasyon problemlerinin çözümü için sığ su dalga modelinden esinlenmiştir. WWO algoritmasın f amaç fonksiyonuna ait çözüm uzayı X, deniz tabanı ile eş anlamlı görülür ve $x \in X$ noktasının uygunluğu deniz tabanına olan uzaklığıyla ters orantılı olarak hesaplanır yani; su seviyesi ne kadar alçak ise uygunluk fonksiyon değeri o kadar yüksektir. Burada 3 boyutlu deniz tabanının daha büyük boyutlu problemlerinden çözümü amacıyla n boyutlu bir uzaya genelleştirildiğine dikkat edilmelidir.

WWO algoritması belli bir çözüm popülasyonu oluşturur. Bu popülasyonun her bir üyesi dalga olarak adlandırılır ve bir yükseklik, $h \in Z$ ve dalga boyu $\lambda \in R$ değerine sahiptir. İlk atamada her dalga için h değeri sabit bir h_{max} ve λ değeri 0,5 olarak atanır. Problemin

çözüm sürecinde dalgaların üç tip operasyonu göz önüne alınır: Yayılma (Propagation), Kırılma (Refraction), ve Bozulma (Breaking).

Yayılma:

Her jenerasyonda, her dalga bir kere yayılır. Bu operasyon orijinal *x* dalgasını her boyutta kaydırarak yeni bir *x'* dalgasını (15) formülüyle hesaplar.

$$x'(d) = x(d) + w\lambda L(j) \quad (15)$$

Burada; *w* [-1,1] aralığında uniform dağılımla oluşturulmuş rassal sayılar, *L(j) j.* boyutun uzunluğudur. Eğer yeni pozisyon arama uzayının dışındaki bir noktaya gelirse; arama uzayı dâhilinde yeni bir rassal noktaya yeniden ayarlanır.

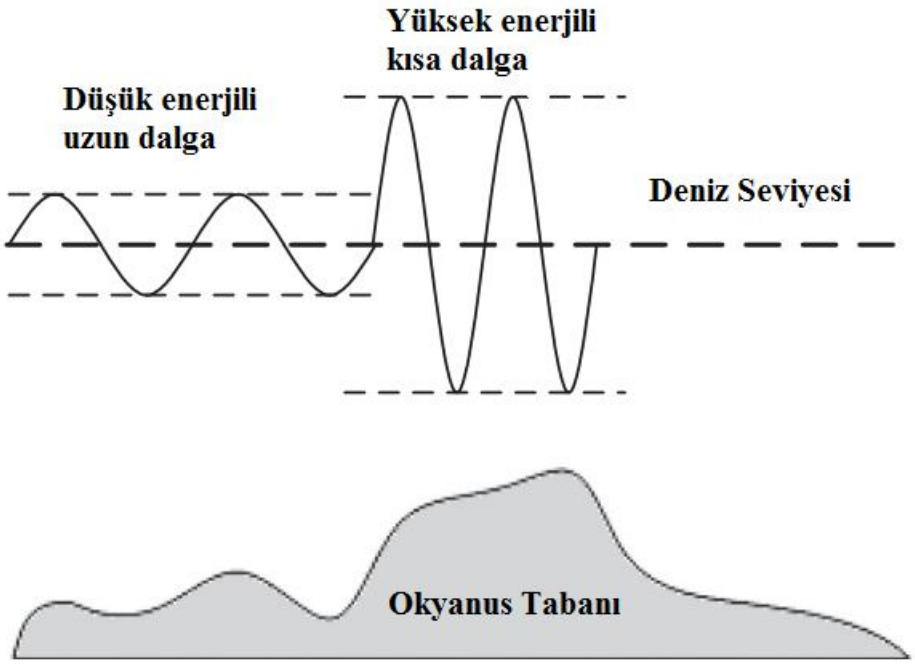
Bir dalga derin sulardan (düşük uygunluk fonksiyon değerli), sığ sulara (yüksek uygunluk fonksiyon değerli) geçtiğinde; dalga yüksekliği artar ve dalga uzunluğu azalır. Şekil 2.2.'de bu durumun temsili bir görüntüsü sunulmuştur.

Yayılmanın ardından yeni bulunan *x'* dalgasının uygunluk değeri hesaplanır. Eğer uygunluk değerinde iyileşme oluşmuş ise popülasyonda *x* dalgası yerine *x'* dalgası geçer ve yüksekliği h_{max} olarak güncellenir. Aksi takdirde *x* dalgası popülasyonda kalır ancak, yükseklik değeri h bir düşürülür.

Her jenerasyonun ardından her *x* dalgasının dalga boyu denklem 16 ile güncellenir.

$$\lambda = \lambda * \alpha^{-(f(x)-f_{min}+e)/(f_{max}-f_{min}+e)} \quad (16)$$

f_{max} ve f_{min} o anki popülasyonun en büyük ve en küçük uygunluk değerlerini ifade ederken, α dalga azaltma katsayısı ve *e* sıfıra bölme hatasıyla karşılaşmamak için kullanılan çok düşük bir sayıdır. Denklem (16), yüksek uygunluk değerli dalgaların daha küçük dalga boyutlarına sahip olmasını ve böylece daha düşük uzaklıklarda yayılmalarını garanti altına almaktadır.



Sekil 2.2. Derin ve sığ sulardaki farklı dalgalar [7]

Kırılma:

Dalganın yayılması esnasında; eğer dalga parçaları eş derinliğe dik değil ise, yönü saptırılacaktır. Bu duruma örnek Şekil 2.3.'de sunulmuştur.

Şekil 2.3. Dalga kırılması [7]

WWO algoritmasında yüksekliği sıfıra düşen dalgalarda kırılma uygulanmaktadır. Kırılmanın ardından oluşacak yeni pozisyonu hesaplamak için denklem (17) kullanılır.

$$x'(j) = N\left(\frac{x^*(j)+x(j)}{2}, \frac{|x^*(j)-x(j)|}{2}\right) \quad (17)$$

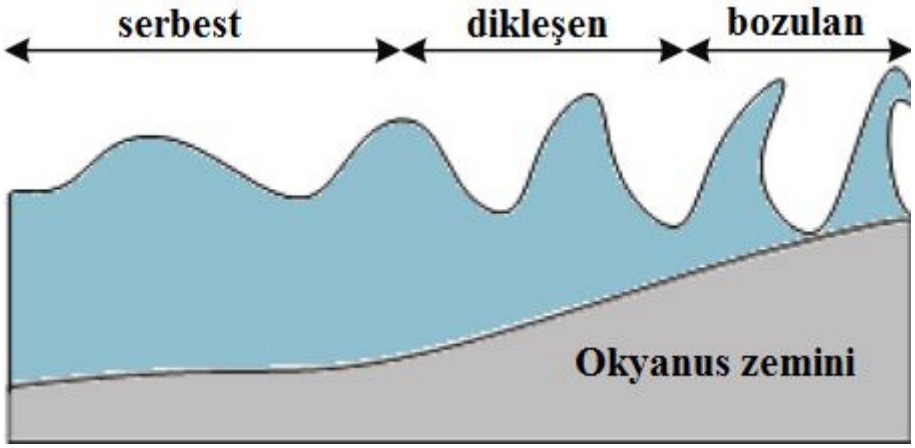
Burada x^* şu ana kadar bulunan en iyi pozisyonu ifade ederken N Gaussian rassal sayısıdır.

Kırılmanın ardından x' dalgasının yüksekliği gene h_{max} değerine eşitlenir ve dalga boyu (18) denklemiyle belirlenir.

$$\lambda' = \lambda f(x)/f(x') \quad (18)$$

Bozulma:

Dalga eşik değerin altındaki derinliğe sahip bir noktaya ulaştığında; dalga tepesinin ivmesi hızını aşar ve ardından; dalga tepesi gittikçe dikleşir ve Şekil 2.4. de gösterildiği gibi dalga sıralı yalnız dalgalar şeklinde bozulur.



Şekil 2.4. Dalganın bozulması [7]

WWO algoritmasında sadece yeni en iyi pozisyonu bulan x dalgasında bozulma operasyonu gerçekleştirilir ve bu noktada lokal arama yapılarak bozulma simüle edilir. Bu operasyon için denklem (19)'dan faydalanılır.

$$x'(j) = x(j) + N(0,1).\beta L(j) \quad (19)$$

Burada; β bozulma katsayısıdır. Eğer yeni oluşturulan yalnız dalgalardan hiçbiri x dalgasından daha iyi değil ise, x dalgası popülasyonda kalır, ancak daha iyi bir nokta bulunur ise, bu nokta x dalgasının yerine popülasyona geçer.

Tablo 2.5. WWO algoritması için sözde kod [7]

N adet dalga için P popülasyonunu rassal olarak oluştur;
while durdurma kriterine kadar **do**
for each $x \in P$ do
Yayılımı denklem (15)'e göre hesapla
if $f(\mathbf{x'}) > f(x)$ then
if $f(x') > f(x^*)$ then
Bozulma operasyonunu x' üzerinde denklem (19)'a göre gerçekleştir.
x^*'ı x' ile güncelle
x'i x' ile değiştir.
else
x 'e ait h değerini bir düşür
if $x.h$==0 **then**
denklem (18)ve (17)'ye göre x'i yeni x''lara kır.
Denklem (16)'ya göre dalga boylarını güncelle
x^*'ı geri dönder.

2.3.6. Güncel Çalışmalar

Optimizasyon alanında yapılan çalışmalar güncelliğini korumakta ve revaçta bir alan olarak günümüzde yer almaktadır. Yeni bazı optimizasyon algoritmaları ;

- "Enhanced leader PSO" (Geliştirilmiş lider PSO) [8],
- "An efficient hybrid Particle Swarm and Swallow Swarm Optimization Algorithm" (Verimli bir hibrit Parçacık Sürü ve Yutulmuş Sürü Optimizasyon Algoritması) [9],
- "Enhanced Compact Artificial Bee Colony" (Geliştirilmiş Kompakt Yapay Arı Kolonisi) [10],
- "the Intelligent Water Drops" (Zeki Su Damlacıkları) [11],
- "Modified Bat Algorithm" (Modifiye Yarasa Algoritması)[12].

Optimizasyon metodları kullanılarak bazı problemlerin çözümüne yönelik yapılmış çalışmalar;

- "Distributed DOA estimation using clustering of sensor nodes and diffusion PSO algorithm" (Algolayıcı Düğümlerinin kümesinin kullanılmasıyla, dağıtılmış DOA tahmini ve yayılan PSO algoritması)[13],

- "Hybridizing ant colony optimization with firefly algorithm for unconstrained optimization problems" (Kısıtsız optimizasyon problemleri için hibritleştirilmiş karınca koloni optimizasyonu ile ateşböceği algoritması) [14],
- "An ant colony algorithm for the multi-compartment vehicle routing problem" (Çok bölmeli araç yönlendirme sorunu için bir karıca koloni algoritması) [15],
- "A quantum-behaved particle swarm optimization with memetic algorithm and memory for continuous non-linear large scale problems" (Sürekli, doğrusal olmayan, geniş ölçekli problemler için, memetik algoritma ve hafıza ile quantum davranışlı parçacık sürü optimizasyonu) [16],
- "Minmax robustness for multi-objective optimization problems" (Çok amaçlı optimizasyon problemleri için minimum maksimum sağlamlığı) [17],
- "Improved seeker optimization algorithm hybridized with firefly algorithm for constrained optimization problems" (Kısıtlı optimizasyon problemleri için; ateşböceği algoritmasıyla hibritleştirilmiş, gelişmiş arama optimizasyon algoritması) [18]
- "A survey on nature-inspired optimization algorithms with fuzzy logic for dynamic parameter adaptation" (Dinamik parametre adaptasyonu için; bulanık mantık içeren, doğadan esinlenilmiş optimizasyon algoritmalarının incelemesi) [19].

Yapılan çalışmalar genel olarak incelendiğinde; yeni algoritmalar, var olan algoritmaların geliştirilmesi ve/veya var olan algoritmaların çeşitli problemlerin çözümünde kullanımlarına yönelik çalışmalar oldukları gözlemlenmektedir.

Optimizasyon algoritmalarının meta sezgisellerinin, güncelleme fonksiyonlarının ya da popülasyonlarının kullanım şekillerinde yapılacak iyileştirmeler ile daha etkili sonuçlar elde edilmektedir. Bu alandaki çalışmaların hız kesmeden devam etmekte olduğu her yıl yayınlanmakta olan pek çok çalışmadan açıkça görülmektedir.

Optimizasyon yöntemlerinin farklı alanlardaki spesifik problemlerin çözümü için yapılan araştırmalarda yapılan yayınlarda önemli bir kesimi oluşturmaktadır. Güncel problemler için özelleştirilmiş metasezgisel algoritmaların etkili bir çalışma alanı olduğu görülmektedir.

3. BULUT BİLİŞİM

Bulut bilişim (Cloud computing) veya işlevsel anlamıyla çevrim içi bilgi dağıtımı; bilişim aygıtları arasında ortak bilgi paylaşımını sağlayan hizmetlere verilen genel addır. Bulut bilişim bu yönüyle bir ürün değil, hizmettir; temel kaynaktaki yazılım ve bilgilerin paylaşımı sağlanarak, mevcut bilişim hizmetinin; bilgisayarlar ve diğer aygıtlardan elektrik dağıtıcılarına benzer bir biçimde bilişim ağı (tipik olarak İnternet'ten) üzerinden kullanılmasıdır. Kelime anlamı olarak;

1-Kişisel bilgisayara ya da ofis sunucularına (servers) yüklenmiş programlardan ziyade, sıklıkla web tarayıcı (browser) gibi araçlarla internet üzerindeki uygulamalara ve bilgisayar verilerine ulaşmak ve kaydetmek,

2-İnternet-temelli hesaplama; bilgi, IT kaynakları ve yazılım uygulamalarının istek temelli olarak mobil cihazlara sunmak,

3-Web-tabanlı uygulamalara, web servislerine ve IT altyapılarına servis olarak internet üzerinden ulaşmaktır.

Birbirinden farklı nitelikteki pek çok cihazın ortak olarak çalıştığı bir alan olan bulutun kontrolü ve yönetimi büyük bir karmaşıklığa sahiptir. Böylesi bir karmaşıklığın üstesinden gelebilmek için ölçeklenebilir ve yönetilebilir bir sistem kurulmasını sağlamak amacıyla Bulut İşletim Sistemi kavramı ortaya atılmıştır. Michael Park, Corporate Vice President of Marketing in the Server & Tools Business at Microsoft, Bulut İşletim Sistemini şöyle tanımlamıştır: "En yüksek katmanda, Bulut İşletim Sistemi geleneksel işletim sistemlerinin yaptığı şeyi yapar – uygulama ve donanımları yönetir- ancak bunu bulut hesaplama kapsam ve ölçeğinde yapar "[20].

OpenStack OS, bulut bilişim sistemleri için güçlü bir açık kaynak platformudur. Openstack OS Compute, Storage ve Image Services servislerini sunan sistem, özel bulut bilişim kümeleri oluşturmak için oldukça gelişmiş özellikler sunan açık bir platformdur. OpenStack, özellikle bulut bilişim üzerinde sanallaştırma çözümleri kullanarak hızlı sistemler kurmak ve bunları etkin bir şekilde yönetmek için uygun bir çözümdür. OpenStack'ın kullanım alanlarından bazıları:

- Kümelendirilmiş hesaplama uygulamaları (Hadoop gibi…)
- Kümelendirilmiş veritabanı ve nosql uygulamaları
- Caching sistemleri
- Yük dengeleme sistemleri ve bu sistemlerin arkasında çalışan uygulama sunucuları

3.1. Büyük Veri

Büyük veri, çok büyük ve karmaşık olduklarından eldeki veritabanı yönetim araçlarıyla ya da geleneksel veri işleme algoritmalarıyla işlenmeleri güç olan veri kümeleri koleksiyonlarına yönelik bir terimdir. Güçlükleri veriyi elde etme, depolama, arama, paylaşma, transfer etme, analiz etme ve görselleştirme alanlarında kendini göstermektedir.

"Büyük verileri çoğu ilişkisel veritabanı yönetim sistemleri ve masaüstü istatistik ve görselleştirme paketleriyle çalışmak güçtür. Bunun yerine devasa paralel yazılımların onlarca, yüzlerce hatta binlerce sunucunun üzerinde çalıştırılmasını gerektirir" [21].

The Apache™ Hadoop® projesi açık kaynaklı, güvenli, ölçeklenebilir, dağıtık hesaplama yazılımı geliştirmektedir.

Hadoop yazılım kütüphanesi; basit programlama modelleri kullanılarak, bilgisayar kümeleri üzerinde büyük verilerin dağıtık olarak işlenmesine imkân vermektedir. Her biri lokal hesaplama ve depolama hizmeti veren bir serverdan binlerce makineye kadar ölçeklenebilir olacak şekilde dizayn edilmiştir. Donanıma güvenerek yüksek uygunluk beklemek yerine, kütüphanenin kendisi uygulama katmanında hataları bulmak ve gidermek üzere dizayn edilmiştir, yani her biri hataya meyilli bilgisayar kümelerinin üzerinde yüksek uygunluklu servis sunar.

3.2. MapReduce

2004 yılında Google MapReduce raporunu yayınlamıştır [22]. MapReduce yeni bir dağıtık programlama modeli sunmuş olup bu modelle sıradan sunucular üzerinde kolaylıkla büyük veriler üzerinde yüksek performanslı paralel programların yazılmasına olanak sunmuşlardır.

Basitçe MapReduce programları iki temel modülden oluşmaktadır; mapper ve reducer. Tipik bir MapReduce görevi şu adımlardan oluşur:

- Veri bloklara parçalanarak mapper modüllerine gönderilir,
- Mapperlar gelen veriden (anahtar, değer) çiftleri çıkarır,
- Daha sonra program anahtar, değer çiftlerini listeler,

  $$\text{Map } (k1,q1) \rightarrow \text{list } (k2,q2)$$

- Anahtar/değer listesi oluşturulunca, liste reducer modüllerine gönderilir,
- Reducer modülleri gruplama görevi olarak görülebilecek, toplam anahtar/değer çıktısını oluşturur.

Reduce (*k2,* list (*q2*)) → list (*q*3)

Mapper ve reducer modülleri kullanıcı tarafından tanımlanan programlar olup, MapReduce API'si kullanılarak (implement edilerek) yazılır.

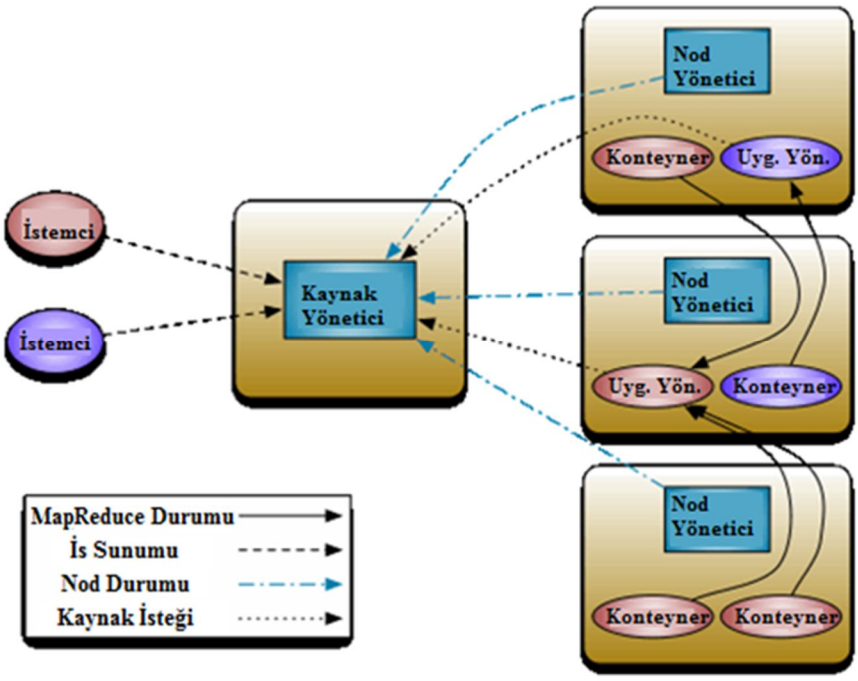
3.3. Apachi Hadoop

Apache Hadoop Google'un Mapreduce frameworkunun en popüler açık kaynak kodlu uygulamasıdır. Hadoop büyük verilerin MapReduce programlama modeliyle depolanmasına ve işlenmesine imkân sağlar.

Hadoop master-slave mimarisini kullanmaktadır. Şu an güncel versiyonu 2.5.1 olup yeni nesil MapReduce olarak geçmektedir, MapReduce 2.0 (MRv2) ya da YARN olarak anılmaktadır. Global bir Kaynak Yöneticisi (ResourceManager) birimi ve uygulama başına bir Uygulama Yöneticisi birimi (ApplicationMaster) bulunmaktadır.

ResourceManager ve nod slaveleri ile Nod Yöneticisi (NodeManager) veri-işleme altyapısını oluşturur. ResourceManager sistemdeki tüm uygulamaların kaynaklarının yönetimini üstlenen nihai karar mevkiidir.

Her uygulamanın sahip olduğu ApplicationMaster, etkisel olarak, ResourceManager ile müzakere ederek kaynak alan ve NodeManager ile birlikte görevi çalıştıran ve izleyen birimdir.



Şekil 3.1. YARN mimarisi [23]

ResourceManager iki ana bileşene sahiptir: Scheduler (Zamanlayıcı) ve ApplicationManager.

Scheduler çalışan çeşitli uygulamalara alışıldık kısıtlara bağlı (kapasite, kuyruk vs.) kaynakların bölüştürülmesinden sorumludur. Scheduler uygulamaların izlenmesi ya da takip edilmesi gibi işlemler yapmadığı için mantıksal olarak saf bir zamanlayıcı olarak çalışır. Ayrıca uygulama ya da donanım hatalarından ötürü başarısız olan görevlerin yeniden başlatılması konusunda da herhangi bir garanti sunmaz. Scheduler fonksiyonunu uygulamanın kaynak ihtiyacına dayalı olarak gerçekleştirir.

ApplicationManager görev isteklerini kabul etmek, uygulamaya has ApplicationMaster'ı çalıştırmak ve hata durumundan ApplicationMasterı yeniden başlatma servisi sunmakla yükümlüdür.

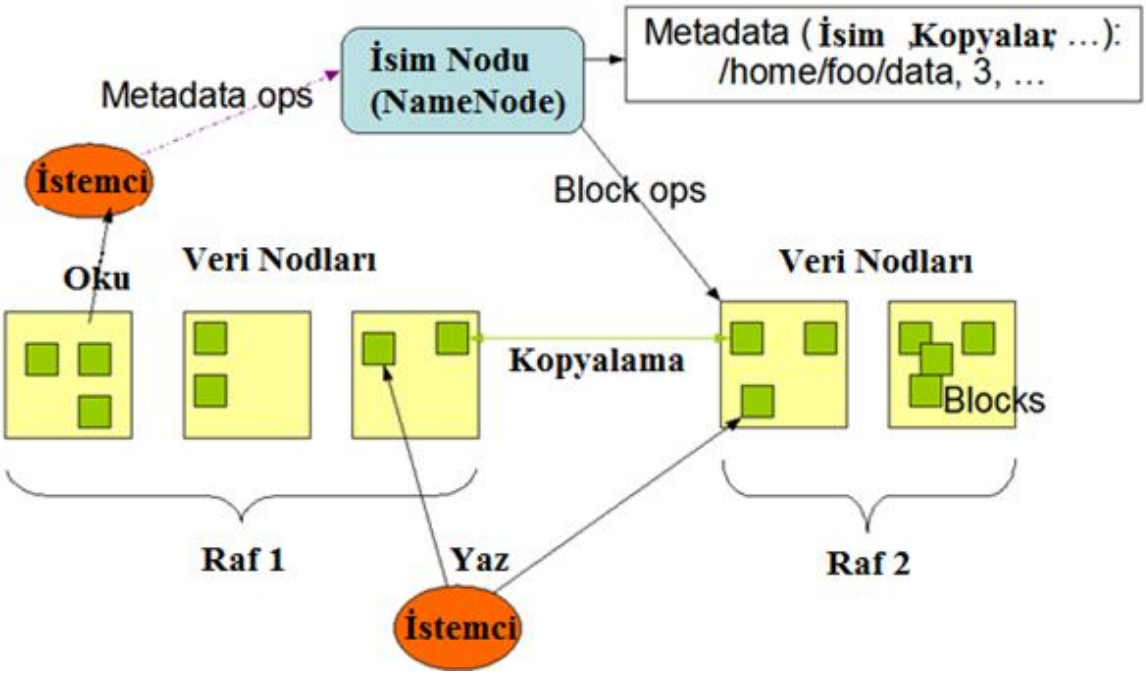
NodeManager makine başına oluşturulan altyapı ajanıdır. Konteynırlardan ve konteynırların kaynak kullanımının (cpu, memory, disk, network) gözlemlenmesinden ve ResourceManager/Scheduler'a rapor sunulmasından sorumludur [23].

3.3.1. Hadoop Dosya Sistemi

Google'un ilk çıkardığı MapReduce altyapısında dosyaları saklamak ve yönetmek için GFS (Google File System) kullanılmıştır. Daha sonra Hadoop kendi MapReduce altyapısında bu dosya sistemini geliştirerek HDFS (Hadoop File System) adını vermiştir.

HDFS, Hadoop uygulamaları tarafından kullanılan temel dağıtık depolama birimidir. Bir HDFS kümesi temelde, dosya sisteminin meta dataları yöneten NameNode ve gerçek verileri depolayan DataNode'dan oluşur. Client'ler dosya meta datalarına ulaşmak ya da dosya modifikasyonları ve DataNode'larla I/O işlemleri yapmak için NameNode ile iletişim kurar [23].

HDFS sıradan donanımlarla dağıtık depolama yapmak için hata toleranslı, ölçeklenebilir, güvenli bir yapı sunar [23].



Şekil 3.2. HDFS mimarisi [23]

3.3.2. Örnek MapReduce Programı

Ek-1'de kodları verilmiş olan kodlar; bir dosyadaki kelimeleri sayan MapReduce uygulamasına ait kodlar olup Hadoop'un resmi sitesinde [23] örnek kod olarak sunulmuştur. Koddan da görüleceğe üzere Mapper sınıfından türeyen bir map sınıfı (TokenizerMapper) ile reducer sınıfından türeyen bir reduce (IntSumReducer) sınıfı oluşturularak main fonksiyonu içinde konfigürasyon (configuration) ve iş (job) sınıflarının oluşturulup gerekli parametrelerin ve map ve reduce sınıflarının verilmesi ile program çalışmaya hazır hale gelmektedir.

MapReduce platformunda problemler map ve reduce fonksiyonlarına bölünerek çözülür. Problemin çözümü için birden fazla map ve reduce fonksiyonu kullanılması ya da aynı map veya reduce fonksiyonunun tekrar tekrar çalıştırılması gerektiği hallerde; eskiden mevcut bir yöntem olmadığından, ilk map-reduce ikilisi çalıştırılarak, çıktısı alındıktan sonra manuel olarak (kullanıcı tarafından) ikinci kez istenilen map-reduce ikilisiyle tekrar koşturulmaktaydı. Kullanıcının da müdahil olduğu bu yöntem istenilen kadar tekrar edilmekteydi. Ancak, kullanıcının müdahil olduğu bu yöntem artık chainMapper sınıfıyla ortadan kalkmış ve map-reduce ikililerinin otomatik olarak zincirleme kullanılmasının önü açılmıştır.

Tablo 3.1. Zincirleme harita kullanımı, chainMapper sınıfı [23]

```
...
 Job = new Job(conf);
Configuration mapAConf = new Configuration(false);
...
ChainMapper.addMapper(job, AMap.class, LongWritable.class, Text.class,
Text.class, Text.class, true, mapAConf);

Configuration mapBConf = new Configuration(false);
...
ChainMapper.addMapper(job, BMap.class, Text.class, Text.class,
LongWritable.class, Text.class, false, mapBConf);
...
job.waitForComplettion(true);
```

Chainmapper sınıfı kullanılarak mevcut Job (iş) için kullanılacak map sınıfları addMapper metoduyla sırayla eklenir. addMapper sınıfı parametreleriyle aşağıda verilmiştir.

Tablo 3.2. addMapper sınıfına ait parametreler [23]

```
addMapper(Job job, Class<? extends Mapper> xclass,
 Class<?> inputKeyClass, Class<?> inputValueClass,
Class<?> outputKeyClass, Class<?> outputValueClass,
Configuration mapperConf)
```

Hadoop içinde anahtar ve değer olarak kullanılabilecek çeşitli sınıflar tanımlanmıştır. Çalışmamızda sürekli bir optimizasyon problemini çözmek için, hadoop'u kullanacağımızdan ihtiyacımız olan sınıflar; değer için long bir değişken ve anahtar olarak ise integer bir değişken olacaktır. Hadoop'da bu değerleri taşıyacak olan sınıflara bir örnek Ek-2'de sunulmuştur.

Oldukça etkin ve kısa zamanda program yazmayı ve uygulamayı çalıştırmayı sağlayan hadoop büyük veriler için standart cihazlar üzerinde çalışan güvenilir, hızlı ve etkin bir platformdur.

3.3.3. MapReduce Araştırmaları

MapReduce platformu kullanılarak <*değer,anahtar*> çiftlerine bölünebilen ve text olarak giriş çıkışı sağlanabilen her türlü problem çözülebilmektedir. Burada ki önemli nokta; <*değer,anahtar*> çiftleri kullanılarak map ve reduce fonksiyonlarıyla problemin çözülebilir yapıda olmasıdır.

Bu alanda yapılan bazı güncel çalışmalar;

- A MapReduce-based approach for shortest path problem in large-scale networks (Geniş ölçekli ağlarda en kısa yol problemi için MapReduce temelli yaklaşım) [24],
- An efficient MapReduce-based rule matching method for production system (Üretim sistemi için verimli bir MapReduce temelli kural uydurma metodu) [25],
- Parallel online sequential extreme learning machine based on MapReduce (MapReduce temelli paralel, çevrimiçi, sıralı, ekstrem öğrenme makinası) [26],
- Hierarchical attribute reduction algorithms for big data using MapReduce (MapReduce kullanarak, büyük veriler için hiyerarşik özellik indirgeme algoritması) [27],
- A MapReduce based parallel SVM for large-scale predicting protein–protein interactions (Proteinler arası etkileşimin büyük ölçekte öngörülmesi için MapReduce temelli paralel SVM) [28],
- MapReduce optimization algorithm based on machine learning in heterogeneous cloud environment (Heterojen bulut ortamında makine öğrenmesine dayalı, MapReduce optimizasyon algoritması) [29],
- Distributed Extreme Learning Machine with kernels based on MapReduce (MapReduce temelli çekirdekler ile Dağıtık Ekstrem Makine Öğrenmesi) [30],
- Parallel attribute reduction algorithms using MapReduce (MapReduce kullanılarak, paralel özellik indirgeme algoritmaları) [31].

Yapılan çalışmaların geneli incelendiğinde büyük işlem kapasitesine ihtiyaç duyulan veriler üzerinde paralel çalışarak zaman ve masraftan tasarruf etmek amacıyla map ve reduce fonksiyonlarıyla üretilen çözümler sunulmuştur.

Üzerinde çalışan problemlerin büyük bölümü hâlihazırda var olan problemler olmasına rağmen geniş ölçekli problemler üzerinde etkin çözüm yolları bulma amacıyla bu

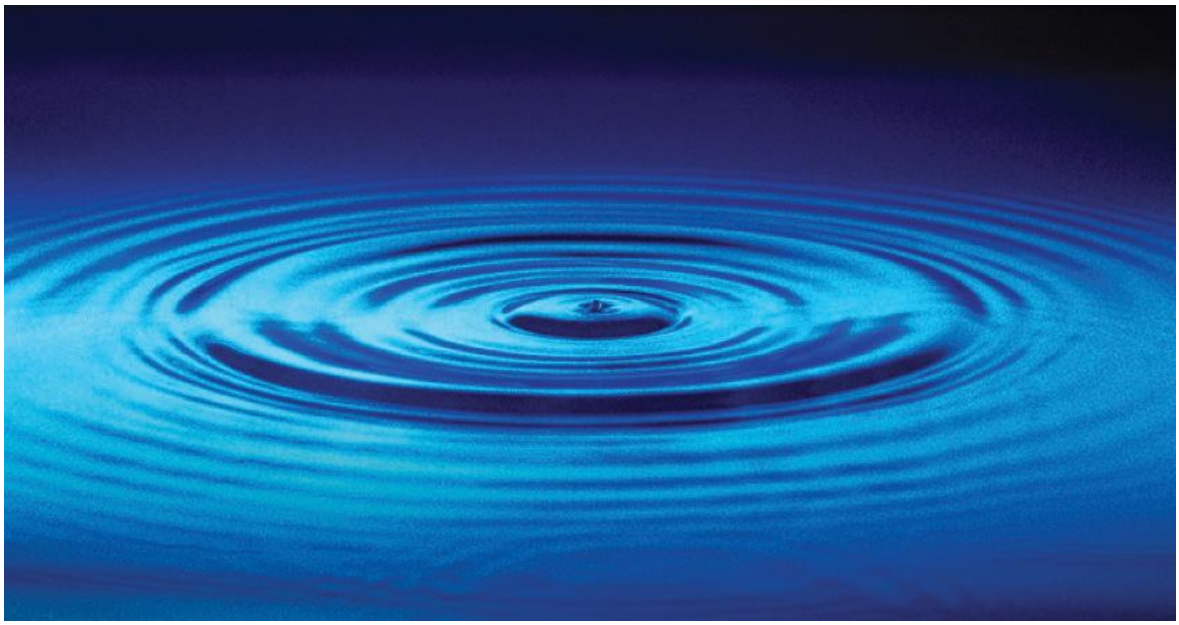
problemlerin MapReduce platformunda etkili çözümleri araştırılmış ve bulunan map ve reduce temelli yöntemler ilim dünyasına sunulmuştur.

4. UYGULAMA

MapReduce platformunda (Hadoop) metasezgisel optimizasyon algoritmasının dağıtık olarak gerçekleştirilmesi amaçlanmıştır. Bu amaçla özel olarak yeni bir optimizasyon algoritması tasarlanmıştır ve kodlamasının kolaylığı ve dağıtık olarak çalıştırılmaya müsait yapısı sebebiyle bu yeni tasarlanan algoritma seçilmiştir.

4.1. Dairesel Su Dalgaları Algoritması (Circular Water Waves-CWW)

Durgun su yüzeyine bir damla suyun damlatılması halinde ortaya çıkan dairesel dalgalardan esinlenilerek geliştirilmiştir [32]. Örnek bir görüntü Şekil 3.3.'de sunulmuştur.



Şekil 3.3. Dairesel su dalgaları[33]

n değişkenli bir f fonksiyonunu göz önüne alalım. f fonksiyonun değişkenlerini x_j ($1 \leq j \leq n$) olarak ifade edelim ve f fonksiyonunun minimum ya da maksimum y değerlerini bulmayı amaçladığımızı varsayalım ki y;

$f(x_0, x_1, \ldots, x_{n-1}, x_n) = y$ olarak tanımlansın.

Arama uzayının ortasından bir S noktası seçilsin. Bu noktanın suya damlatılan su damlası olduğunu düşünülsün. Bu noktanın çevresinden dalgaların geçtiği gibi çevresindeki alandaki tüm noktaların fonksiyon değerlerini hesaplamak mümkün olamayacağından; r_j yarıçap değeri ile c_a dalga dairesi sayısında noktalar hesaplanacaktır. ($1 \leq a \leq M$ M= maksimum dalga daire sayısı).

$$d_j = r_j \times w \times c_a \tag{15}$$

(15) denkleminden yararlanılarak j. boyuttaki d fark değerleri hesaplanır. Burada d_j S başlangıç noktasından j. boyuttaki farkını, r_j j. boyuttaki yarıçapını, w rassal olarak üretilmiş (her seferinde yeni bir w değeri oluşturulur) [0,1] aralığındaki bir sayıyı, c_a ise kaçınca daire olduğunu ifade etmektedir. d farklarının hesaplanması ile yeni noktalar bulunarak arama gerçekleştirilir.

Akıllı bir arama yapmak adına; her boyut için yeni fonksiyon değerleri hesaplanır. Örneğin 3 boyutlu bir problem için şu noktaların uygunluk fonksiyonları hesaplanıp kontrol edilmelidir;

Nokta 1: (x_0+d_0, x_1, x_2), Nokta 2: (x_0-d_0, x_1, x_2),

Nokta 3: $(x_0, x_1+d_1, , x_2)$, Nokta 4: $(x_0, x_1-d_1, , x_2)$,

Nokta 5: (x_0, x_1, x_2+d_2), Nokta 6: (x_0, x_1, x_2-d_2),

S başlangıç noktasından daha iyi uygunluk değerine sahip olan noktalar kullanılarak; yeni bir muhtemel en iyi nokta hesaplanır ve bu noktadan yola çıkılarak rassallaştırılmış muhtemel en iyi noktalar hesaplanır. Örneğin Nokta 2 ve Nokta 4 S noktasından daha iyi uygunluk değerine sahip ise;

Nokta 7: (x_0-d_0, x_1-d_1, x_2),

Nokta 8: $(x_0-w\times d_0, x_1-w\times d_1, x_2)$.

noktaları hesaplatılır. Rassal kaç adet nokta hesaplanacağı probleme göre kararlaştırılabilir. Genel kullanım için rassal nokta sayısının fonksiyonun değişken sayısıyla eşit olması uygun görülmektedir.

Yarıçap değeri olan r_j için başlangıç değeri olarak j. boyutun uzunluğunun yarısının kullanılması tavsiye edilmektedir. Uygunluk değerinin iyileşmemesi halinde;

$$r_j = r_j/M \tag{16}$$

(16) denklemi ile yarıçap güncellenir.

Her iterasyonda, dalga daire sayısı M, problemin boyutu n, yeni başlangıç nokta sayısı b olmak üzere; iterasyonun hesaplaması gereken nokta sayısı:

$$M(3n+1)b \tag{17}$$

(17) ile hesaplanır.

Tablo 4.1. CWW algoritmasının sözde kodu

Başlangıç noktalarını oluştur.
Do:
Her başlangıç noktası için
Her dalga dairesi için
-d_j değerlerini (15)'e göre hesapla
-Her yönde dalga noktaları oluştur ve uygunluklarını hesapla
-Muhtemel en iyi noktayı ve fitnesini hesapla
-Muhtemel en iyi noktayı rassallaştır
-Yeni noktaların uygunluklarını hesapla
Eğer uygunluk iyileşmemişse
-başarısızlık sayacını arttır ve r_i değerlerini (16)'ya göre güncelle
En iyi *b* adet noktayı yeni dalga başlangıç noktası olarak ata.
While(başarısızlık sayacı < 10)

4.2. Deneysel Çalışmalar

Deneysel çalışmalar kapsamında algoritma, sekiz kıyaslama fonksiyonu ile 2 ila 30 boyutlu problemler üzerinde; 3,5 ve 10 dalga daire sayılarıyla denenmiştir. Her arama için işlem 40 defa gerçekleştirilmiş olup ortalama, en iyi ve en kötü değerler sunulmuştur. Kullanılan fonksiyonlar ve deneysel sonuçlar sunulmuştur.

Algoritmanın denendiği fonksiyonlar;

F_1 Ackley's Function;

$$f(x,y) = -20\exp\left(-0.2\sqrt{0.5(x^2+y^2)}\right) - \exp\left(0.5(\cos(2\pi x)+\cos(2\pi y))\right) + e + 20$$

F_2 Beale's Function;

$$f(x,y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$$

F_3 Six Hump Camelback Function

$$f(x,y) = \left(4 - 2.1x^2 + \frac{x^4}{3}\right)x^2 + xy + (-4 + 4y^2)y^2$$

F_4 Goldstein-Price Function;

$$f(x,y) = \big(1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)\big)(30 + (2x - 3y)^2)(18 - 32x + 12x^2 + 48y - 36xy + 27y^2))$$

F_5 Lévi Function N.13;

$$f(x,y) = sin^2(3\pi x) + (x-1)^2\big(1 + sin^2(3\pi y)\big) + (y-1)^2(1 + sin^2(2\pi y))$$

F_6 Eggholder Function;

$$f(x,y) = -(y + 47)\sin\left(\sqrt{\left|y + \frac{x}{2} + 47\right|}\right) \text{-xsin}(\sqrt{|x - (y + 47)|})$$

F_7 Hölder Table Function;

$$f(x,y) = -\left|\sin(x)\cos(y)\exp\left(\left|1 - \frac{\sqrt{x^2 + y^2}}{\pi}\right|\right)\right|$$

F_8 Sphere Function;

$$f(x) = \sum_{i=0}^{n} x_i^2$$

Fonksiyonların arama uzayları ve gerçek minimum değerleri Tablo 4.3.'de verilmiştir.

Tüm fonksiyonlar Tablo 4.3.'de sunulan abstrack fonksiyon sınıfından türetilerek işlenir.

Tablo 4.2. Fonksiyonlar için abstrack sınıf.

```
  import java.util.Random;
  public abstract class function {
     public abstract double calculate(double[] degiskenler);
     public    abstract    double[]    limitations(double[]
degiskenler);
     public abstract double minimum();
     public abstract double[] defaultStartPoint();
     public abstract double fitness(double fonksiyondegeri);
     public abstract double[] getrange();
     public abstract int functype();
     public abstract double[] defaultstartpint(int boyut);
     public abstract double[] getrange(int boyut);
     public static int getSignal()
     {
          Random r=new Random();
          int temp=r.nextInt(100);
          if(temp<50)
               return -1;
          return 1;
     }
  }
```

Tablo 4.3. Fonksiyonlar

Fonksiyon	Arama Uzayı	Minimum
F_1	$-5 \le x,y \le 5$	f(0,0)=0
F_2	$-4.5 \le x,y \le 4.5$	f(3,0.5)=0
F_3	$-3 \le x_1 \le 3$ $-2 \le x_2 \le 2$	f(0,0898,-0,7126) =1,0316 f(-0,0898,0,7126) =1,0316
F_4	$-2 \le x,y \le 2$	f(0,-1) = 3
F_5	$-10 \le x,y \le 10$	f(1,1) = 0
F_6	$-512 \le x,y \le 512$	f(512, 404.2319) = -959.6407
F_7	$-10 \le x,y \le 10$	f(±8.05502, ±9.66459) =-19.2085
F_8	$-5.12 \le x_i \le 5.12$, $i = 1, 2, \ldots, n$.	x* = (0, …, 0), f(x*) = 0.

Algoritmaya ait deneysel sonuçlar Tablo 4.4-.4.11 tablolarında sunulmuştur.

Tablo 4.4. F_1 fonksiyonu için deneysel sonuçlar

n	m	b	Ortalama Değer	En iyi Değer	En kötü Değer	Ortalama İterasyon
2	3	3	2,04281036531029 E-15	0	3,5527136788005 E-15	53,95
2	5	3	9,76996261670138 E-16	0	3,5527136788005 E-15	49,35
2	10	3	1,15463194561016 E-15	0	7,105427357601E -15	45,67

Tablo 4.5. F_2 fonksiyonu için deneysel sonuçlar

n	m	b	Ortalama Değer	En iyi Değer	En kötü Değer	Ortalama İterasyon
2	3	3	0,035494064244 0729	9,9055602789 4557E-15	0,7620696508 92314	64,15
2	5	3	0,070479076167 4664	7,3471167496 2036E-20	1,1776327352 4131	64,7
2	10	3	0,038733822974 0361	1,1448858223 7879E-13	1,1580275296 0407	66,1

Tablo 4.6. F_3 fonksiyonu için deneysel sonuçlar

n	m	b	Ortalama Değer	En iyi Değer	En kötü Değer	Ortalama İterasyon
2	3	3	1,031628453489 88	1,0316284534 8988	1,0316284534 8988	35,37
2	5	3	1,031628453489 88	1,0316284534 8988	1,0316284534 8988	33,42
2	10	3	1,031628453489 88	1,0316284534 8988	1,0316284534 8988	31,45

Tablo 4.7. F_4 fonksiyonu için deneysel sonuçlar

n	m	b	Ortalama Değer	En iyi Değer	En kötü Değer	Ortalama İterasyon
2	3	3	47,59269243355 69	3	87,525665840 8539	51,975
2	5	3	44,17081813706 69	3	89,053044226 4534	50,125
2	10	3	39,30261647977 05	3	86,605156800 7117	46,2

Tablo 4.8. F_5 fonksiyonu için deneysel sonuçlar

n	m	b	Ortalama Değer	En iyi Değer	En kötü Değer	Ortalama İterasyon
2	3	3	1,34969464963992E-31	1,34969464963992E-31	1,34969464963992E-31	57,5
2	5	3	1,34969464963992E-31	1,34969464963992E-31	1,34969464963992E-31	52,05
2	10	3	1,34969464963992E-31	1,34969464963992E-31	1,34969464963992E-31	46,975

Tablo 4.9. F_6 fonksiyonu için deneysel sonuçlar

n	m	b	Ortalama Değer	En iyi Değer	En kötü Değer	Ortalama İterasyon
2	3	3	-915,692830785117	-959,640662720851	-633,842302239904	40,375
2	5	3	-929,747255522451	-959,640662720851	-704,805153877182	36,52
2	10	3	-947,689995274587	-959,640662720851	-888,949125269878	30,3

Tablo 4.10. F_7 fonksiyonu için deneysel sonuçlar

n	m	b	Ortalama Değer	En iyi Değer	En kötü Değer	Ortalama İterasyon
2	3	3	-19,2085025678867	-19,2085025678868	-19,2085025678867	38,15
2	5	3	-19,2085025678867	-19,2085025678868	-19,2085025678867	35,525
2	10	3	-19,2085025678867	-19,2085025678868	-19,2085025678868	33,45

Tablo 4.11 F_8 fonksiyonu için deneysel sonuçlar

n	m	b	Ortalama Değer	En iyi Değer	En kötü Değer	Ortalama İterasyon
2	3	3	0	0	0	470,3
2	5	3	0	0	0	426.875
2	10	3	0	0	0	368,1
10	3	3	0	0	0	548,2
10	5	3	0	0	0	548,2
10	10	3	0	0	0	548,075
30	3	3	1,26446320988337E-09	3,5752422224758E-129	2,84482586613925E-08	121.2
30	5	3	0	0	0	549,975
30	10	3	0	0	0	549,925

Deneysel sonuçlar çerçevesinde CWW algoritmasının başarılı bir algoritma olduğu gözlemlenmiştir. Düşük hata payına ve iterasyon sayısına sahip olup, yapısı gereği dağıtık olarak işlenmeye gayet uygundur. CWW algoritması bir kere iterasyonuna başlayınca diğer herhangi bir parametreye ihtiyaç duymamaktadır ki bu durum; dağıtık mimarideki argüman paylaşımından kaynaklı karmaşıklığı ortadan kaldırmaktadır.

4.2. Hadoop Uygulaması

Hadoop kullanılarak yukarıda bahsi geçen CWW metasezgisel optimizasyon algoritması ile problem çözümü gerçekleştirilecektir.

Optimizasyon algoritmalarının dağıtık olarak ve kolayca uygulanabilir bir yapı üzerinde gerçekleştirilmesinin metasezgisel optimizasyon algoritmalarının kullanımında yeni bir yol açacağı ve dağıtık işletme için bu algoritmaların özelleşmeleriyle; etkili hızlı ve düşük maliyetli çözümlerin ortaya çıkacağı öngörülmektedir.

Hadoop platformu verilerin nodlara dağıtılması amacıyla kullanılacaktır. Bu sebeple ayrı ayrı map ve reduce fonksiyonlarının tanımlanmasına gerek yoktur. Sadece map fonksiyonunun tanımlanarak bu fonksiyona verilerin gönderilmesiyle CWW algoritmasının çalıştırılması mümkündür. Text halinde yollanacak olan verilerin; text içindeki yapısı Tablo 4.11'de verilmiştir. Her ne kadar Word'de çoklu satırmış gibi gösteriliyor olsa da text olarak bu yazımın tek satır olduğuna dikkat edilmelidir. Birden çok değişken içerenler '%' ile ve farklı değişkenler '/' ile ayrılmıştır.

Tablo 4.12. Verilerin text halinde "%" ve '/' ile ayrılmış yazımı

```
degisken0% degisken1%.....%degiskenn/
radius0%radius1%....%radiusn/
hatasayacı/
fonksiyon degeri/
uygunluk deger/
dalgasayısı/
yapılacak iterasyon sayısı/
yapılmış iterasyon sayısı
```

Tablo 4.12'de verilen Map1 sınıfı CWW algoritmasının nasıl çalışacağına ışık tutmaktadır. Parçalara ayrılarak; nodlara gönderilen text dosyası içinden çıkarılan <*değer,anahtar*> çiftleri map fonksiyonuna iletilir. map fonksiyonu gelen *String* cinsindeki girişi *data* sınıfının yapılandırıcısına parametre olarak göndermekte ve bu data sınıfında değişkenler String içinden çıkarılmaktadır. Örnek olarak Sphere fonksiyonu kullanılmıştır. cww sınıfı CWW algoritmasını işleten sınıftır ve fonksiyon tipi ile data sınıfından giriş parametrelerini alır. Daha sonra cww algoritması calculate metodu ile yeni en iyi noktaları bularak bir data sınıfı listesi halinde geri dönderir ve bunlar içinden en iyi iki tane nokta yeniden işlenmek üzeri map nesnesinin çıkışı olarak gönderilir. Her map fonksiyonu yeni noktalarla işlenecek başlangıç nokta sayısını iki kat arttırmaktadır. Eğer çok fazla data olması arzu edilmiyor ise sadece en iyi yeni noktanın çıkışa gönderilmesi tercih edilebilirken, diğer yandan daha derinlemesine bir ama yapılması arzu ediliyorsa listenin tamamına kadar istenilen sayıda yeni nokta oluşturulabilir.

Tablo 4.13. MapReduce ile CWW algoritması için birinci Map1 sınıfı

```
public static class Map1 extends MapReduceBase
   implements Mapper<LongWritable, Text, Text, IntWritable> {

      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();

      public void map(LongWritable key, Text Değer,
            OutputCollector<Text, IntWritable> output,
            Reporter reporter) throws IOException {
              data d=new data(Value.toString());
              Sphere s=new Sphere();
              cww alg=new cww(s,d);
               List<data>  nd=alg.calculate();
               int a=-1,b=-1;
                  double best1=Double.MAX_VALUE,best2=Double.MAX_VALUE;
              for(int i=0;i<nd.size();i++)
              {
                        //word.set(nd.get(i).getDataString());
                        //one.set(one.get()+1);
                        //output.collect(word, one);

                        if(best1>nd.get(i).fval)
                        {
                                 best1=nd.get(i).fval;
                                 a=i;
                        }
                        else if(best2>nd.get(i).fval)
                        {
                                 best2=nd.get(i).fval;
                                 b=i;
                        }

              }
              one.set(one.get()+1);
                if(a!=-1)
              output.collect(new Text(nd.get(a).getDataString()),one);
                    if(b!=-1)
                     {
                                 output.collect(new Text(nd.get(b).getDataString()), one);
                     }
      }
   }
```

Çalışmamızda iki adet map fonksiyonu kullanılmıştır. İki adet map fonksiyonu kullanılmasının sebebi, ilk gelen verinin anahtar değerlere sahip olmayan ham veri oluşudur. İlk map fonksiyonu ham veriden <*değer,anahtar*> çiftlerini çıkarır.

Zincirlenecek diğer map nesnesi Tablo 4.14'de sunulmuş olup Tablo 4.13'deki map nesnesinden farkı; ham veri yerine, ilk map fonksiyonundan ve kendinden çıkan <*değer,anahtar*> çiftlerini işleyip gene <*değer,anahtar*> cinsinden çıktı üretme kapasitesinde olmasıdır.

Tablo 4.14. MapReduce ile CWW algoritması için birinci Map2 sınıfı

```
public static class Map2 extends MapReduceBase
   implements Mapper<Text, IntWritable, Text, IntWritable> {

      public void map(Text key, IntWritable value,
            OutputCollector<Text, IntWritable> output,
            Reporter reporter) throws IOException {
              data d=new data(key.toString());
         Sphere s=new Sphere();
         cww alg=new cww(s,d);
         List<data> nd=alg.calculate();
         int a=-1,b=-1;
         double best1=Double.MAX_VALUE,best2=Double.MAX_VALUE;
           for(int i=0;i<nd.size();i++)
                {
                     if(best1>nd.get(i).fval)
                     {
                              best1=nd.get(i).fval;
                              a=i;
                     }
                     else if(best2>nd.get(i).fval)
                     {
                              best2=nd.get(i).fval;
                              b=i;
                     }

                }
         IntWritable val=new IntWritable(value.get()+1);
         if(a!=-1)
               output.collect(new Text(nd.get(a).getDataString()), val);
         if(b!=-1)
                        output.collect(new Text(nd.get(b).getDataString()), val);

      }
   }
```

Tablo 3.1.'de gösterilen chainMapper sınıfının yardımıyla algoritmanın kaç iterasyon çalışması isteniliyorsa o kadar sayıda Map fonksiyonu zincirlenir. MapReduce altyapısı kullanılarak belli bir hata payında durdurma imkânı yoktur, ancak; CWW algoritmasının

pozitif yanlarından biri de elde edilen çıktının yeniden kullanılabilir yapıda olmasıdır. Elde edilen çıktının verdiği sonuç, tatmin edici aralıkta değil ise; söz konusu çıktıyı girdi olarak kullanıp, çalışmaya devam etmek mümkündür. Tablo 4.15'de chainMapper sınıfının kullanılması ve konfigürasyonların yapılmasıyla ilgili kodlar sunuluştur.

Tablo 4.15. chainMapper Kullanımı ve Konfigürasyonlar

```
public int run(String[] args) throws Exception {
        JobConf conf = new JobConf(getConf(), ChainOptimization.class);
        conf.setJobName("Optimizasyon");

        if (args.length != 2) {
           System.out.println("ERROR: Wrong number of parameters: " +
                 args.length + " instead of 2.");
           return printUsage();
        }
        FileInputFormat.setInputPaths(conf, args[0]);
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        JobConf mapAConf = new JobConf(false);
        ChainMapper.addMapper(conf, Map1.class, LongWritable.class, Text.class,
Text.class, IntWritable.class, true, mapAConf);


        JobConf mapBConf = new JobConf(false);
        ChainMapper.addMapper(conf, Map2.class, Text.class, IntWritable.class,
Text.class, IntWritable.class, true, mapBConf);

        for(int i=0;i<550;i++)
        {
        ChainMapper.addMapper(conf, map2.class, Text.class, IntWritable.class,
Text.class, IntWritable.class, true, mapBConf);
        }
        JobClient.runJob(conf);
        return 0;
     }
```

Jobconf sınıfından türetilen bir iş konfigürasyon nesnesi ile sistemin konfigürasyon ayarları yapılır. İşin ismi *conf.setJobName(JobName)* fonksiyonu ile girilir. Giriş dosyası yolu *FileInputFormat.setInputPaths(conf, args[0])*satırı ile ayarlanır. Gene benzer şekilde çıkış dosyasının yolu *FileOutputFormat.setOutputPath(conf, new Path(args[1])* satırıyla ayarlanır. Farklı sınıflar için ayrı ayrı konfigürasyon nesneleri oluşturularak bu

konfigürasyon nesneleri ile map sınıfları Tablo 4.15.'de görüldüğü üzere chainMapper sınıfı aracılığıyla eklenir. İstenilen iterasyon sayısı kadar Map2 sınıfı bir for döngüsü ile chainMapper nesnesine eklenerek bu map sınıflarının ardışıl olarak çalıştırılması sağlanır.

Bu çalışmada herhangi bir reduce fonksiyonu kullanılmamıştır. Ancak istenilmesi halinde sadece en iyi sonucun çıktı verilmesi amacıyla ya da en iyi belli sayıdaki sonucun çıktı olarak üretilmesi amacıyla bir reduce fonksiyonu kullanmak mümkündür.

Hadoop platformunun verileri text olarak kaydedip, text olarak dağıtması sebebiyle küçük ölçekli problemlerde bu yöntemin kullanılması verimli olmamaktadır. Ancak çok büyük ölçekli problemlerde, verilerin hızlı ve güvenli paylaşımını sağlanması amacıyla, hadoop platformunun optimizasyon problemlerinde kullanılmasının faydalı olacağına kanaat edilmektedir. Ancak her optimizasyon algoritmasının Hadoop platformunda iyi performans sağlamayacağına dikkat edilmelidir. CWW algoritması özellikle dağıtık ortamlarda çalışmak üzere tasarlanmış veri paylaşımına minimal oranlarda ihtiyaç duyan etkili bir algoritma olması vesilesiyle Hadoop platformu içinde uygun bir seçimdir.

5. SONUÇLAR

Bu çalışmada optimizasyon algoritmaları incelenmiş ve MapReduce çalışma mantığı, yapısı ve örnek programları üzerinde durulmuştur. Optimizasyon algoritmalarının kullanımına değinilerek; giderek artan büyüklüklerdeki optimizasyon problemlerinin çözümünde dağıtık işlemenin önemine dikkat çekilmiştir. Dağıtık işleme kapasitesine sahip bir sistem olan MapReduce'e bir giriş yapılarak optimizasyon algoritmalarının MapReduce üzerinde çalıştırılması üzerinde durulmuştur.

Çalışma kapsamında yeni ve özgün bir optimizasyon algoritması olan *"Dairesel Su Dalgaları (Circular Water Waves-CWW)* algoritması tasarlanmıştır.

CWW algoritmasının Hadoop platformunda çalıştırılarak sonuç alma yöntemi sunulmuştur. Hadoop platformunda psuedo-distributed modda çalışılmıştır. Küçük ölçekli problemlerde dağıtık modda çalışmanın verimli olmadığına dikkat çekilmiş ancak, çok büyük ölçekli problemlerin bu yolla çözülmesinin uygun olduğuna kanaat getirilmiştir.

EKLER

EK-1 Örnek MapReduce Programı [34]

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
```

```
                    ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Ek-2 LongWritable Sınıfı [34] .

```
19 package org.apache.hadoop.io;
20
21
22 import java.io.DataInput;
23 import java.io.DataOutput;
24 import java.io.IOException;
25
26 import org.apache.hadoop.classification.InterfaceAudience;
27 import org.apache.hadoop.classification.InterfaceStability;
28
30 @InterfaceAudience.Public
31 @InterfaceStability.Stable
32 public class LongWritable implements WritableComparable<LongWritable> {
33  private long value;
34
35  public LongWritable() {}
36
37  public LongWritable(long value) { set(value); }
38
40  public void set(long value) { this.value = value; }
41
43  public long get() { return value; }
44
45  @Override
46  public void readFields(DataInput in) throws IOException {
47  value = in.readLong();
48  }
49
50  @Override
51  public void write(DataOutput out) throws IOException {
52  out.writeLong(value);
53  }
```

```
54
56  @Override
57  public boolean equals(Object o) {
58  if (!(o instanceof LongWritable))
59  return false;
60  LongWritable other = (LongWritable)o;
61  return this.value == other.value;
62  }
63
64  @Override
65  public int hashCode() {
66  return (int)value;
67  }
68
70  @Override
71  public int compareTo(LongWritable o) {
72  long thisValue = this.value;
73  long thatValue = o.value;
74  return (thisValue<thatValue ? -1 : (thisValue==thatValue ? 0 : 1));
75  }
76
77  @Override
78  public String toString() {
79  return Long.toString(value);
80  }
81
83  public static class Comparator extends WritableComparator {
84  public Comparator() {
85  super(LongWritable.class);
86  }
87
88  @Override
89  public int compare(byte[] b1, int s1, int l1,
```

```
90  byte[] b2, int s2, int l2) {
91  long thisValue = readLong(b1, s1);
92  long thatValue = readLong(b2, s2);
93  return (thisValue<thatValue ? -1 : (thisValue==thatValue ? 0 : 1));
94  }
95  }
96
98  public static class DecreasingComparator extends Comparator {
99
100  @Override
101  public int compare(WritableComparable a, WritableComparable b) {
102  return -super.compare(a, b);
103  }
104  @Override
105  public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
106  return -super.compare(b1, s1, l1, b2, s2, l2);
107  }
108  }
109
110  static { // register default comparator
111  WritableComparator.define(LongWritable.class, new Comparator());
112  }
113
114 }
115
```

KAYNAKLAR

[1] URL-1, http://www.mitrikitti.fi/opthist.html Optimizasyon tarihçesi 03 OCAK 2015

[2] **Kennedy J. , Eberhart R.**, 1995, Particle Swarm Optimization, *Proceedings of IEEE International Conference on Neural Networks IV.,* 1942–1948. doi:10.1109/ICNN.1995.488968.

[3] **M. Dorigo**, "Optimization, Learning and Natural Algorithms", PhD thesis, Politecnico di Milano, Italy, 1992.

[4] URL-2, http://inndustry.blogspot.com.tr/2013/03/ant-colony-algorithm-lecture-and-videos.html Karınca koloni algoritması eğitim videoları, 03 OCAK 2015

[5] **D. Karaboğa**, 2005 Yapay Zeka ve Optimizasyon Algoritmaları, Nobel Yayınevi ISBN: 978-605-395-434-7

[6] **Yang, X.-S.,** 2008, Nature-Inspired Metaheuristic Algorithms, Luniver Press.

[7] Yu-Jun Zheng, 2015, Water wave optimization: A new nature-inspired metaheuristic *Computers & Operations Research* 55 (2015) 1–11

[8] **A. R. Jordehi**, 2015, Enhanced leader PSO (ELPSO): A new PSO variant for solving global optimisation problems, *Applied Soft Computing* **Vol. 26**, 401–417, to be published.

[9] **A. Kaveh , T. Bakhshpoori, E. Afshari**, 2014, An efficient hybrid Particle Swarm and Swallow Swarm Optimization Algorithm, *Computers and Structures* **143**, 40–59

[10] **A. Banitalebi, M. I. A. Aziz, A. Bahar, A. Aziz**, 2014, Enhanced compact artificial bee colony, *Information Sciences*, to be published.

[11] **H. Shah-Hosseini**, 2012, An approach to continuous optimization by the Intelligent Water Drops, *Procedia Social and Behavioural Sciences*, **Vol. 32**, 224-229

[12] **S. Yılmaz, E. U. Kucuksille, Y. Cengiz**, 2014, Modified Bat Algorithm, *Elekronika IR Elektrotechnika*, ISSN 1392-1215, **Vol. 20**, No. 2, 2014

[13] **Trilochan P. , Ganapati P. , Bernard M. , Babita M.**, 2013, Distributed DOA estimation using clustering of sensor nodes and diffusion PSO algorithm, *Swarm and Evolutionary Computation* **9**, 47–57

[14] **R.M. Rizk-Allah, Elsayed M. Zaki, Ahmed Ahmed El Sawy**, 2013, Hybridizing

ant colony optimization with firefly algorithm for unconstrained optimization problems, *Applied Mathematics and Computation* **224,** 473-483

[15] **M. Reed, A. Yiannakou, R. Evering**, 2014, An ant colony algorithm for the multi-compartment vehicle routing problem, *Applied Soft Computing* **Volume 15**, 169–176

[16] **D. Tang , Y. Cai, J. Zhao, Y. Xue**, 2014, A quantum-behaved particle swarm optimization with memetic algorithm and memory for continuous non-linear large scale problems, *Information Sciences* **289,** 162–189

[17] **M. Ehrgott, J. Ide, A. Schöbel**, 2014, Minmax robustness for multi-objective optimization problems, *European Journal of Operational Research* **239,** (2014) 17–31

[18] **M. Tuba, N. Bacanin**, 2014, Improved seeker optimization algorithm hybridized with firefly algorithm for constrained optimization problems, *Neurocomputing* **143,** 197–207

[19] **F. Valdez, P. Melin, O. Castillo**, 2014, A survey on nature-inspired optimization algorithms with fuzzy logic for dynamic parameter adaptation, *Expert Systems with Applications* **41,** (2014) 6459–6466

[20] URL-3, http://blogs.technet.com/b/microsoft_blog/archive/2013/01/15/what-is-the-cloud-os.aspx Bulut nedir, 3 Ocak 2015

[21] **Tom White**, 2012, Hadoop The Definitive Guide, O'Reilly ISBN: 978-1-449-31152-0

[22] **Jeffrey D. and Sanjay G.,** 2004, MapReduce: Simplified Data Processing on Large Clusters, *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA

[23] URL-4 http://hadoop.apache.org/ Hadoop resmi sitesi, 05 Ocak 2015

[24] **Sabeur A. , Philippe L., , Libo R. , Benjamin V.**, 2015, A MapReduce-based approach for shortest path problem in large-scale networks Engineering *Applications of Artificial Intelligence*, **Volume 41**, May 2015, Pages 151–165

[25] **Ying Li et al.**,2015 , An efficient MapReduce-based rule matching method for production system, *Future Generation Computer Systems,* Available online 31 March 2015, In Press

[26] **Botao W., Shan H., Junhao Q., Yu L., Guoren W.**, 2015, Parallel online

sequential extreme learning machine based on MapReduce, *Neurocomputing,* **Volume 149**, Part A, 3 February 2015, Pages 224–232

[27] **Jin Qian et al.**, 2014, Hierarchical attribute reduction algorithms for big data using MapReduce, *Knowledge-Based Systems,* **Volume 73**, Pages 18–31

[28] **Zhu-Hong Y. et al.**, 2014, A MapReduce based parallel SVM for large-scale predicting protein–protein interactions*, Neurocomputing,*Volume **145**, Pages 37–43

[29] **Wen-hui L. et al.**, 2013, MapReduce optimization algorithm based on machine learning in heterogeneous cloud environment, *The Journal of China Universities of Posts and Telecommunications*, Volume 20, **Issue 6**, Pages 77–87, 121

[30] Xin Bi et al., 2015, Distributed Extreme Learning Machine with kernels based on MapReduce, *Neurocomputing*, **Volume 149, Part A**, Pages 456–463

[31] **Jin Qian et al.**, 2014, Parallel attribute reduction algorithms using MapReduce, *Information Sciences,* **Volume 279**, Pages 671–690

[32] **Muhammed Emre Ç. , Asaf V. ,** 2015, A Novel Intelligent Optimization Algorithm Inspired from Circular Water Waves, *19th International Conference ELECTRONICS 2015* accepted for oral represantation and *Elektronika ir Elektrotechnika* (IF-0,445 (2013); ISSN: 1392-1215). to be published.

[33] URL-5 http://2012books.lardbucket.org/books/principles-of-general-chemistry-v1.0/section_10/1fd32a5a7af89ce657a1129e583d47be.jpg Dairesel su dalgaları 01.04.2015

[34] **U**RL-6 http://fossies.org/dox/hadoop-2.5.1-src/LongWritable_8java_source.html LongWritable ve IntWritable sınıflarına ait kaynak kodları, 27 Eylül 2014

ÖZGEÇMİŞ

1984 doğumlu Muhammed Emre ÇOLAK ilk, orta ve lise öğrenimini Elazığ'da tamamladı. 2008 yılında kazandığı Fırat Üniversitesi, Bilgisayar Mühendisliği Bölümünden 2012 yılında mezun oldu. 2012 yılında Konya Selçuk Üniversitesi Teknoloji Fakültesi Bilgisayar Mühendisliği Anabilim Dalına ÖYP puanı ile araştırma görevlisi olarak yerleştirildi ve yüksek lisans eğitimine başladı. 2014 yılında, Fırat Üniversitesi, Fen Bilimleri Enstitüsü, Yazılım Mühendisliği Anabilim Dalına ÖYP puanı ile yerleştirildi. 2014 yılı Ocak ayında, Fırat Üniversitesi, Mühendislik Fakültesi, Yazılım Mühendisliği Anabilim Dalında Araştırma Görevlisi olarak göreve başlamış ve halen bu görevine devam etmektedir.