T.C.
DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KAR TANESİ OPTİMİZASYON ALGORİTMASI

ÖZGÜN AKKOYUN

YÜKSEK LİSANS TEZİ
ELEKTRİK ELEKTRONİK VE BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

DANIŞMAN
DR. ÖĞR. ÜYESİ METİN TOZ

DÜZCE, 2019

T.C.
DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KAR TANESİ OPTİMİZASYON ALGORİTMASI

Özgün AKKOYUN tarafından hazırlanan tez çalışması aşağıdaki jüri tarafından Düzce Üniversitesi Fen Bilimleri Enstitüsü Elektrik Elektronik ve Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Dr. Öğr. Üyesi Metin TOZ

Düzce Üniversitesi

Jüri Üyeleri

Dr. Öğr. Üyesi Metin TOZ
Düzce Üniversitesi _____

Prof. Dr. İbrahim YÜCEDAĞ
Düzce Üniversitesi _____

Doç. Dr. Devrim AKGÜN
Sakarya Üniversitesi _____

Tez Savunma Tarihi: 01/07/2019

BEYAN

Bu tez çalışmasının kendi çalışmam olduğunu, tezin planlanmasından yazımına kadar bütün aşamalarda etik dışı davranışımın olmadığını, bu tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara kaynak gösterdiğimi ve bu kaynakları da kaynaklar listesine aldığımı, yine bu tezin çalışılması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını beyan ederim.

1 Temmuz 2019

Özgün AKKOYUN

TEŞEKKÜR

Yüksek Lisans öğrenimimde ve bu tezin hazırlanmasında gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Dr. Öğr. Üyesi Metin TOZ'a en içten dileklerimle teşekkür ederim.

1 Temmuz 2019 **Özgün AKKOYUN**

İÇİNDEKİLER

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

ÇİZELGE LİSTESİ

Sayfa No

KISALTMALAR

ABC	Yapay Arı Kolonisi
ACO	Karınca Kolonisi Algoritması
ALO	Karınca Aslanı Optimizasyon Algoritması
ES	Evrim Stratejisi
FOA	Orman Optimizasyon Algoritması
GA	Genetik Algoritma
GBMO	Gaz Brownian Hareketi
GSA	Yerçekimsel Arama Algoritması
GWA	Gri Kurt Algoritması
HTS	Isı Transferi Arama
OIO	Optikten Esinlenilen Optimizasyon
PSO	Parçacık Sürüsü Optimizasyonu
SA	Benzetilmiş Tavlama Algoritması
SFO	Kar Tanesi Optimizasyon Algoritması
WOA	Balina Optimizasyon Algoritması
WSA	Ağırlıklı Süperpozisyon Çekimi

SİMGELER

Abs	Mutlak Değer Fonksiyonu
M	Mutasyon Fonksiyonu
R	Rüzgar Hızı
S	Kar Tanelerinin Oluşturduğu Nüfus
S_2	Rüzgarın Etkisinden Sonra Oluşan Nüfus
S_3	Sıralama Fonksiyonu Sonrası Oluşan Nüfus
S_4	Mutasyon Fonksiyonu Sonrası Oluşan Nüfus
s_i	Aday Çözümlerden Her Biri
S_r	Seçilen Uygunluk Değerine Ait Aday Çözüm
U_v	Uygunluk Değerleri Vektörü
$U_v(S_r)$	Rulet Tekeri Algoritması Yardımıyla Uygunluk Değerleri Göz Önüne Alınarak Seçilmiş Kar Tanelerinin Uygunluk Değeri
Vr	Referans Yükseklikte Esen Rüzgar Hızı
$V(z)$	İlgili Seviyedeki Hesaplanmış Olan Rüzgar Hızı
Y	Yer Değiştirme Fonksiyonu
z	Hız Bilgisine Göre Hesaplanmak İstenilen Yüksekliği
z_r	Referans Seviyedeki Yüksekliği
α	Mutasyon Oranı
β	Adım Büyüklüğü

ÖZET

KAR TANESİ OPTİMİZASYON ALGORİTMASI

Özgün AKKOYUN
Düzce Üniversitesi
Fen Bilimleri Enstitüsü, Elektrik Elektronik ve Bilgisayar Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi
Danışman: Dr. Öğr. Üyesi Metin TOZ
Temmuz 2019, 40 sayfa

Son yıllarda özellikle mühendislik problerinin çözümünde oldukça sık bir şekilde optimizasyon kullanılmaktadır. Özellikle sezgisel optimizasyon teknikleri bu problemlerin çözümünde popüler olmuşlardır. Dolayısıyla literatürde bu amaçla geliştirilmiş bir çok sezgisel optimizasyon tekniği yer almıştır. Bu tez çalışmasında da benzer şekilde optimizasyon problemlerinin çözümü için kullanılabilecek yeni bir sezgisel optimizasyon algoritması önerimektedir. Önerilen algoritma kar yağışını esas almaktadır. Algoritmaya göre kar tanelerinin gök yüzünden yer yüzüne ulaşmaları için geçirdikleri serüven optimizasyon açısından en iyiyi arama şeklinde modellenmiştir. Geliştirilen algoritmaya Kar Tanesi Optimizasyon Algoritması adı verilmiştir. Tez çalışmasında bu algoritmanın temelleri, çalışma mantığı ve formülasyonları detaylarıyla sunulmuştur. Ayrıca SFO'nun etkinliği literatürde yer alan onüç benchmark fonksiyon kullanılarak test edilmiştir. Elde edilen sonuçlar algoritmanın bu fonksiyonların çözümünde başarılı olduğunu ortaya koymuştur.

Anahtar sözcükler: Optimizasyon, Kar Tanesi Optimizasyon Algoritması, SFO.

ABSTRACT

SNOWFLAKE OPTIMIZATION ALGORITHM

Özgün AKKOYUN
Düzce University
Graduate School of Natural and Applied Sciences, Department of Electrical-Electronic and Computer Engineering
Master's Thesis
Supervisor: Assist. Prof. Dr. Metin TOZ
July 2019, 40 pages

In recent years, especially in the solution of engineering problems, optimization is frequently used. In particular, heuristic optimization techniques have become popular in solving these problems. Therefore, many heuristic optimization techniques have been developed in the literature for this purpose. In this thesis, a new heuristic optimization algorithm is proposed that can be used to solve optimization problems. The proposed algorithm is based on snowfall. According to the algorithm, the advent of snowflakes in order to reach the surface of the earth from the sky is modeled as searching for the best in terms of optimization. The developed algorithm is called Snowflake Optimization Algorithm. In this thesis, the fundamentals of this algorithm, working principle and formulations are presented in detail. In addition, the effectiveness of SFO was tested using thirteen benchmark functions from the literature. The results showed that the algorithm was successful in solving these functions.

Keywords: Optimization, Snowflake Optimization Algorithm, SFO.

1. GİRİŞ

Optimizasyonun en genel tanımı çözüme ulaştırılması hedeflenen problemin, veri ve kaynakların analiz edilerek, uygun şekilde toplanması ve elde edilen bu veri ve kaynakların en optimum biçimde kullanılmasıdır. Optimizasyonun kullanıldığı alana bağlı olarak daha farklı ve geniş tanımları da vardır. Matematik ve bilgisayar bilimleri alanlarında kullanılan optimizasyon tanımı, akademik alanda en çok kabul gören tanımdır. Bu tanıma göre, optimizasyon bir problemi eldeki verileri ve kaynakları analiz ederek; yazılımsal, donanımsal ve matematiksel olanaklar dahilinde maksimum ya da minimum sonuca ulaştıracak bir çözüm fonksiyonu hazırlamaktır [1]. Optimizasyon olanaklar dahilinde gerçekleştirilen bir çözüm olduğundan belirli kısıtlamalar ile sınanmaktadır.

Optimizasyon, canlıların günlük yaşamlarında neredeyse her karar aşamasında kullandıkları bir eylemdir. Örnek olarak bir insanın ulaşmayı hedeflediği konuma giderken hangi yolu kullanacağına karar verme süreci verilebilir. Optimizasyon problem ile başlar, bu örnekteki problem de kişinin hangi yolu kullanarak daha kısa sürede gideceği yere varacağının belirsizliğidir. İkinci aşamada problemle ilgili verilerin toplanması ve analiz edilmesi gelmektedir. Veri toplama ve analiz sürecinde kişi, öncelikle nerede olduğunu saptamalıdır. Süreç kişinin hedeflediği ve bulunduğu yerlerin konumlarını belirlemesi ve sonrasında bulunduğu yere göre ulaşmayı amaçladığı hedefinin nerede yer aldığının saptanmasıdır. Bu işlem sonrasında ise kullanılabilecek yolların, gidilecek yön ile olan bağlı konumlarını saptamak gerekecektir. Kişi analizini tamamladıktan sonra, problemi çözmek için çözüm üretme aşamasına geçer. Bu aşamada çözümler üretir ve ürettiği çözümleri sınamaya başlar. Çözümler arasından tahmini olarak en kısa sürecek olanı seçer. Bu sayede ulaştığı sonuçlar arasından en kısa sürede kendisini gideceği konuma ulaştırması muhtemel çözümü seçerek, kat edeceği yolu minimize etmiş olur. Kişi bu işlemi, gideceği yolu minimize ederken optimizasyon adımlarını takip ederek gerçekleştirmiştir.

Elbette optimizasyon her zaman bu kadar basit değildir. Örneğin, önceki örnekteki kişinin yeşil alan fobisi olduğundan parklara en uzak yerlerden geçmesi gerektiğini varsayarak

bu problemi karmaşıklaştırabiliriz. Bu durumda kişi gideceği yere ne kadar kısa sürede ulaşacağını ve aynı zamanda parkların ne kadar uzağından geçeceğini de değerlendirerek bir amaç fonksiyon oluşturulur. Optimizasyonun amacı bu tür durumlarda parametreleri en yüksek kazancı sağlayacak şekilde belirleyerek amaç fonksiyonunu en yüksek veya en düşük değere ulaştırmaktır. Bunun için örneğimizdeki probleme uygun bir çözüm hem kısa sürede gidilebilecek hem de parkların çok yakınından geçmeyecek şekilde optimize edilmiş bir rota içermelidir.

Canlılar farkında olmadan içgüdüsel olarak ortaya koydukları birtakım davranışlar ile optimizasyon örnekleri sergilemektedir. Bu bağlamda insanoğlu için optimizasyon doğum ile başlar. İlk insanlardan bu yana neredeyse her konuda optimizasyon örneği vermek mümkündür. Yunanistan'ın en büyük filozoflarından olan Platon (427 M.Ö den 347 M.Ö) ve Aristotle (384 M.Ö den 322 M.Ö) hem bireylerin hem de toplumun davranışlarının nasıl optimize edileceğine dair fikirler geliştirmişlerdir. Özellikle bireylerin yaşam tarzını iyileştirebilmek ve devletin işleyişini optimize edebilmek için çalışmalar yapmışlardır. Platon, Devlet adlı eserinde bir devletin optimum şekilde nasıl işleyeceğini, bunun sağlanması için ise devlet yöneticilerinin ve halkın nasıl davranması gerektiğine yönelik düşüncelerini aktarmıştır [2]. Platon'un kitabında anlattığı model, devletin işlemesi ile amaçlanan sorunların minimuma indirgenmesi ve refah seviyesi yüksek bir şekilde devlet düzeninin devam etmesidir. Bu açıdan bakıldığında Platon bu eseri ile devletin işleyişini kendi metotları ile optimize etmeye çalışmıştır. M.Ö Euclid, iki nokta arasındaki minimum mesafeyi göz önüne alarak bir karenin verilen toplam kenar uzunluğu ile dikdörtgenler arasında en büyük alana sahip olduğunu kanıtlamıştır [3]. Başka bir Yunan filozofu olan Herodotus ise Mısır halkının Nil nehrinin çekilmesi ve taşması dönemlerindeki davranışlarını incelemiştir. Nil nehri yıl içerisinde coğrafi şartların ve coğrafi şekillerin de etkisiyle birden çok kez çekilmekte ve havzasından taşmaktadır. Mısır halkı nehrin taşmasından günler önce uyarılmalarına rağmen en iyi ve uygun şekilde yerleşim yerlerini terk etme vaktini tespit edememiştir. Herodotus, Mısır halkının arazi vergilendirmesi ve nehrin her taşmasında halkın bölgeyi terk etmesi için optimum şekilde davranmasının şart olduğunu belirlemiştir [4].

17. ve 18. Yüzyıla gelindiğinde Kalkülüs'ün icadından (varyasyon analizi) önce bazı tekil optimizasyon algoritmaları geliştirilmeye başlanmıştır. 1615 yılında J. Kepler tarafından şarap varillerinin optimal boyutları araştırılmıştır [5]. 1636 yılında P. De Fermat uç noktada ki bir fonksiyonun türevinin olmadığını göstermiştir. 1657 yılında ise Fermat

ışığın ilerlerken asgari olarak iki nokta arasında geçtiğini göstermiştir [5].

Newton, 1666 yılında Kalkülüs üzerinde çalışmaya başlamıştır. Ancak on yıl kadar bir süre boyunca küçük bir detay vermek dışında bu çalışmasına dair bilgi vermekten kaçınmıştır. Gottfried Leibniz 1674 yılında Kalkülüs üzerine çalışmalarına başlamıştır ve kalkülüsü "Nova Methodus pro Maximis et Minimis" isimli ilk çalışmasında yayımlamıştır. Newton ve Leibniz tarafından Kalkülüs'ün 17. Yüzyılda geliştirilmesi, optimizasyon yöntemlerinin gelişmesinde önemli bir rol almaktadır. Kalkülüs, matematiksel fonksiyonların bağımsız değişkenlere göre maksimum ve minimum şekilde optimum sonuçlarını elde edebilmek için kullanmıştır [3].

Teknolojinin ilerlemesi ile birlikte optimizasyon teknikleri çok çeşitli alanlarda kullanılmaya başlanmıştır. Çeşitli endüstri alanlarında kullanılan optimizasyon, maliyetin minimuma indirgenmesi veya karın maksimize edilmesi gibi ticari amaçlar için de kullanılmaktadır. Endüstri alanı dışında üretim sektörü, lojistik, finans, taşıma gibi alanlarda faydalı bir şekilde kullanılmaktadır. Taşıma için örnek vermek gerekirse; günlük, haftalık veya saatlik olarak yolcu sayılarındaki değişim göz önünde bulundurularak düzenlenecek olan seferlerin hem yolcular için yoğun saatlerde daha çok sefer düzenlenerek yoğunluğun azaltılması sağlanabilirken hem de yolcu sayısının daha az olacağının öngörüldüğü saatlerde sefer sayısı azaltılarak maliyette azalmalar sağlanabilmektedir.

Ekonomi, optimizasyonun yoğun bir şekilde kullanıldığı alanlardan biridir. Mikro iktisatta sıkça karşılaşılan marjinal fayda ve bunun sebeplerinden oluşmuş yapılan harcamaları minimize etme iktisat alanında sıkça karşılaşılan bir problemdir [6]. Firmalar ve tüketiciler kâr oranını maksimize etmek isterler ve bu konumda ise optimizasyon devreye girer. İşletmelerin kısa dönemlerdeki karlarının maksimize edilmesi, zararlarının ise minimize edilmesi için sıklıkla optimizasyon yöntemleri kullanılmaktadır. Bununla beraber işletmenin marjinal faydaya ulaşmasının sağlanması amaçlanırken oluşabilecek tehlikelerin de öngörülmesi ve alınacak önlemlerin belirlenmesi hedeflenir.

Optimizasyon insan doğasında içgüdüsel olarak bulunmasının yanında etkili sistemler oluşturmak ve bu sistemlerin belirli şartlarda çalışmasını sağlamak amacıyla bilinçli olarak da yapılan bir eylemdir. Optimizasyonun gerekliliği optimize edilmemiş sistemlerin fazla kaynak tüketerek az iş yapmasından kaynaklanmaktadır. Örneğin üç aşamalı bir üretim bandına sahip bir şirket varsayalım. Bu üretim bantlarından

birincisinden çıkan ara ürün ikincisine, ikincisinden çıkan ara ürün üçüncüsüne girmektedir. Şirket en az personel ile günlük en çok üretimi yaparak kazancını arttırmayı hedeflemektedir. Bunun için ise bir işçinin yeterli olacağını değerlendirerek, en az maliyete yönelmektedir. Bu durumda bir işçi tek başına önce birinci, sonra ikinci sonra da üçüncü banttaki işleri sırayla yapmak durumunda kalacaktır. Her işin aynı sürede gerçekleştiği varsayıldığında, bu işçi bir birim ürün için üç birim zaman harcayacaktır. Bu sorunu çözmek için işçi sayısı altıya çıkarılabilir. Ancak fabrikada üç bant bulunduğu için aynı anda maksimum üç işçi çalışabilecektir. Bu durum üretim süresi boyunca sürekli olarak üç işçinin boşta kalması anlamına gelecektir. Üç işçinin boşta kalması, üretim maksimum seviyede iken maliyetlerin artmasına neden olacaktır. Bu örnekte maliyet-işçi sayısı optimizasyonunun gerekliliği ekonomik yönden ele alınmıştır. Bilgisayar bilimlerinde ise optimizasyona daha önemli sebeplerden dolayı başvurulmaktadır. Bir bilgisayar, telefon, tablet ve benzeri elektronik cihazların bir görevi gerçekleştirmek için harcadıkları zamanı optimize etmek gerekmektedir. Optimize edilmediğinde uzun süren işlemler dolayısıyla performans problemleri ortaya çıkmaktadır. Bu performans problemlerinin en çok hissedildiği sistemlere örnek olarak donanım sürücü yazılımları, gerçek zamanlı görüntü sağlayan yazılımlar, sık kullanılan derleme dili fonksiyonları ve işletim sistemleri gelmektedir. Sürücü yazılımı optimize edilmemiş herhangi bir bilgisayar parçası, örneğin ekran kartı ya da işlemci, istenilen performansın altında kalır ve sisteme gereksiz enerji ve zaman kaybı yaşatır. Gerçek zamanlı görüntü sağlayan yazılımların optimize edilmesi ile daha kaliteli ve akıcı görüntüler ortaya çıkar. Derleme dili fonksiyonları, makine dili ile üst seviye diller arasında bir mesajcı görevi gördükleri için sık kullanılan fonksiyonlardaki optimizasyon eksiklikleri sistem genelinde sebep olacağı gecikme ile performans düşüşüne yol açar. Bu sık kullanılan fonksiyonlardan bazıları sıralama fonksiyonları, arama fonksiyonları, sıkıştırma fonksiyonları ve şifreleme fonksiyonlarıdır. İşletim sistemleri için de aynı durum söz konusudur. Optimize edilmemiş işletim sistemleri üzerinde çalışan uygulamalar alt fonksiyonları verimli çalışmadığından beklenilen performansı veremeyebilir.

Optimizasyon problemlerinin çözümü için birden fazla yönteme başvurulabilir. Ancak hangi problem için hangi çözümün daha etkili olacağını anlamak gerekir. Bu durumda ise iş problemi çözmek isteyen kişiye düşer. Kişi çözmek istediği problem için en uygun optimizasyon algoritmasını seçmek zorundadır.

Dar anlamda optimizasyon modellemesi matematiksel programlama olarak

adlandırılabilmektedir. Matematiksel programlama ise matematik düzenlemesi ve planlanması anlamına gelmektedir. Optimum değerin bulunması amacıyla matematiksel programlamaya başvurulur, bu durum operasyon araştırmasının bir alt işlemidir [7]. Operasyon araştırması, problemin en iyi ve uygun çözümünü bulma aşamasındaki karar verme tekniklerini içerir. Çözümün optimum değerde bulunması nihai hedeftir. Örneğin; bir kanepenin tasarlanmasının problem olarak kabul edildiğini varsayalım. Kanepenin; maliyetinin düşük olması, rahat olması, estetik görünüme sahip olması, hafif olması, dayanıklı olması gibi kriterler belirlenir. Elbette kriterler tamamen üreticinin inisiyatifinde istenildiği sayıda olabilmektedir. Bu kriterler göz önüne alındığında kriterlerin birbiri ile çeliştiği ve çekiştiği açıkça gözlemlenebilmektedir. Kanepenin rahat ve dayanıklı olması kriterlerine verilen önem ne kadar artırılırsa, kanepenin maliyeti de o derecede artacak ve maliyetin düşük olması kriteri bu durumdan olumsuz etkilenecektir. Tasarlanacak kanepenin optimum olarak alacağı şekil amaç fonksiyon olarak tanımlanır. Bu aşamada optimum değere ulaşılması için diğer kriterler ve değişkenlere ise tasarım değişkenleri adı verilmektedir. Buna karşılık bazı parametreler belirli değerlerin altına düşemeyecektir, bu nedenle kısıtlayıcıların kabul edilmesi gerekmektedir. Kısıtlayıcı olarak kanepenin rahat olması kriteri değerlendirilebilir. Eğer kısıtlayıcı kullanılmadan tasarım yapılırsa, dayanıklılığın artırılması için rahatlıktan tamamen vazgeçilebilir. Bunun önlenmesi için her tasarımda belirli kısıtlayıcılar kabul edilir. Kısıtlayıcılar, tasarım parametrelerinin alacağı önem ve değerlerin sınırlamalarıdır.

Algoritma belirli bir problemi çözüme ulaştırmak için tasarlanan bir çözüm stratejisinin basamaklandırılarak parçalara ayrıldığı bir yöntemdir [8]. Algoritmalar akış şemaları ile veya düz yazı ile gösterilebilir. Akış şemaları algoritmaların neden sonuç bağlamlarını görselleştirmek ve bağlantıları açıkça göstermek için tasarlanan şemalardır. Yukarıda kısaca bahsedildiği üzere, en çok bilinen algoritma türleri; arama algoritmaları, sıralama algoritmaları, sıkıştırma algoritmaları, şifreleme algoritmaları, genetik algoritmalar, en kısa yol algoritmaları, rastgele sayı üretme algoritmaları ve optimizasyon algoritmalarıdır. Sıralama algoritmalarına örnek olarak seçerek sıralama ve hızlı sıralama, sıkıştırma algoritmalarına örnek olarak delta algoritması, şifreleme algoritmalarına örnek olarak rsa ve sha algoritmaları, optimizasyon algoritmalarına örnek olarak parçacık sürüsü algoritması örnek verilebilir.

Optimizasyon algoritmaları optimizasyonun tanımında da bahsedildiği gibi mevcut veriler, donanım ve yazılım olanakları ile en iyi sonuca ulaşmak için tasarlanan

yöntemlerdir. Optimizasyon algoritmaları problem odaklı çözüm yöntemleri oldukları için sınıflandırmaları da problem sınıflandırmaları ile benzerlik göstermektedir. Optimizasyon problemleri dört basamaklı karakteristik ayrım ile sınıflandırılır [9]. Bu karakteristik ayrımlar sürekli ya da parçalı, kısıtlamalı ya da kısıtlamasız, deterministik ya da stokastik, son olarak da bir hedefli, birden çok hedefli ya da hiç hedefi olmayanlardır.

Parçalı karakteristiğe sahip problemlerin değişkenleri parçalı setlerden ya da parçalı setlerin alt kümelerinden oluşmaktadır. Sürekli karakteristikteki problemlerin değişkenleri ise her değeri alabilmektedir. Süreklilik özelliğine sahip bu optimizasyon problemleri herhangi bir x değeri için x'in komşu değerlerinden alınan bilgilerin doğruluğunun fazla olması sebebiyle fonksiyonlarının modellenebilirliği ve çözümü daha basittir [10].

Kısıtlamalı karakteristiğe sahip problemler kısıtlamasız karakteristikteki problemlerin bağlı ve bağsız değişken sayılarını ve/veya değişkenlerin alabilecekleri değerlerin toplam sayısını azaltmak ve çözüme uygunluk setini sağlayan kısmi parçalar ile ulaşmak için değişkenlerin sağlaması gereken koşullar ile doğruluğunun denetlendiği problemlerdir [10].

Deterministik karakteristikteki problemlerde sahip olunan verinin doğruluğu sorgulanmaz, veriler doğru kabul edilir. Stokastik karakteristikte ise verilerin doğruluğuna olasılıksal dağılımlarının etkisi de hesaplanarak bütün verileri kapsayan bir sonuç elde edilmeye çalışılır [10].

Son olarak bir ve birden çok hedefli karakteristiğe bağlı problemlerde optimizasyonun başarı skalası gerçekleştirilebilen hedeflerin sayısı ve varsa ağırlığı ve önemi ile ölçülür.

Doğadaki birçok canlı ve cansız varlığın göstermiş oldukları davranışlar optimumdur. Örneğin; ışık ulaşım süresini minimuma götürecek yolları takip eder ve hedefine en kısa sürede varmaya çalışır. Karıncalar yuvaları ile yiyecek arasındaki en kısa yolu bulmaya çalışıp bu yolu kullanarak yuvalarına yiyecekleri taşırlar. Karıncalar gibi arılar da kovanlarına veya geçici yerleşmelerine en yakın olan çiçeklerden polenleri toplayarak kovanlarına varmayı amaçlamaktadır. Doğadaki birçok canlı ve madde bir şekilde kendisini en uygun çözüme doğru yönlendirir. Doğadan esinlenilen optimizasyon algoritmaları bu canlı ve cansız varlıkların hareketlerini analiz edip, inceler. Bu analiz ve incelemenin amacı; canlı varlıkların, yiyecek aramak, göç etmek, barınak bulmak; cansız

varlıkların ise karın yağışı, ses ve ışık dalgalarının yayılması gibi ihtiyaç ve amaçlarına ulaşmak üzere en kısa ve uygun yolu veya yöntemi seçmek gibi eylemlerini inceleyerek ortaya çıkarmaktır.

Klasik matematiksel teknikler ile çözümlenemeyen optimizasyon problemleri üç ana algoritmik yaklaşım ile çözülür [11]. Bunlar sezgisel algoritmalar, fizik kurallarını esas alan algoritmalar ve evrimsel algoritmalardır [11].

Evrimsel temelli algoritmalar; evrimsel olguları değerlendirerek, bu doğrultuda hesaplama yapan algoritmalardır [12]. Evrimsel bir algoritmanın akışı, topluluk içerisindeki bireylerin mutasyona uğratılıp, değiştirilmesi sonucunda belirli testlere tabi tutularak, yapılan testler sonucunda başarısız olanların elenmesini öngörür. Testler sonucunda başarılı olan bireylerle hedeflenen başarı oranına ulaşılıncaya dek testlere devam edilir. Evrimsel algoritmaların çalışma prensipleri evrim kuramına dayanmaktadır [12]. Evrim kuramı, doğal seleksiyonun gereklerinden ve sonuçlarındandır. Testler sonucunda yüksek performans değerine ulaşan örneklerin veya deneklerin çaprazlama yolu ile başarı oranı daha yüksek sonuçların elde edilmesi amaçlanmaktadır. Evrimin yönünü belirleyen unsur bu yüksek performanslı denekler ve bireylerdir. Aynı toplum içerisinde yaşayan bireylerin güçlü olanları belirli yönleri ile zayıf olanlardan ayrılmaktadır. Toplumda güçlü olan bireyler zayıflardan ayrılmakta ve daha güçlü bireylerin ortaya çıkmasını sağlamaktadır. Bu durum da evrim kuramına bir örnektir. Evrimsel algoritmalara örnek olarak genetik algoritma (GA) [13], evrim stratejisi (ES) [14] ve biyoloji tabanlı optimizasyon algoritması [15] verilebilir.

GA evrimsel algoritmaların ilkidir ve algoritmanın ilk bulguları 1975 yılında John Holland tarafından ortaya atılmıştır [13]. En basit şekilde adapte olma yeteneği en yüksek bireylerin diğerlerinden üstün gelerek yaşamaya devam edebilmesi olarak özetlenebilir. Bu algoritmada mutasyona uğratılan gen sayısının fazla olması durumu analizin anlamlandırılmasını güçleştirirken, arama uzayını rastlantısal bir hale getirmektedir [13]. GA'da seçim için bazı teknikler kullanılmaktadır. Bunlar rulet tekeri, sıralama ve turnuva yöntemleridir [13]. Rulet tekeri seçim yöntemi uygunluk fonksiyonlarıyla orantılı bir seçilim gerçekleştirerek uygulanır. Bu yöntemde seçilecek bireyin seçilme oranı uygunluk fonksiyonundan alınan değerlerin sıralaması ile belirlenmektedir. Turnuva seçim yöntemi ise rastgele seçilen bireylerin uygunluk değerlerinin karşılaştırılarak, daha yüksek değere sahip olan bireyin seçilmesi ile gerçekleşir. Bu yöntemler arasında en az çeşitlilik rulet tekeri, en çok çeşitlilik turnuva yöntemi ile elde edilebilir [13].

ES algoritması 1960'lı yılların başlarında oluşturuldu ve 1970'lerin sonlarına doğru Ingo Rechenberg ve Hans-Paul Schwefek tarafından geliştirilmeye devam edilmiştir [14]. Bu algoritmada diğer evrimsel algoritmalar gibi iterasyonlarla bireylerin seçilip bazı işlemlere tabi tutulması ile yeni bireylerin oluşumu sağlanmaktadır. Diğer algoritmalardan ayrılan en önemli özelliği sadece mutasyon ile yeni birey oluşturma işleminin gerçekleşmesidir [14]. Mutasyonlar gerçekleşirken kullanılan olasılık fonksiyonunun parametrelerinden olan değişimin boyutu da bireylerde kalıtım yoluyla taşınır. Bunun sebebi iyi bireylerin parametrelerinin sonraki nesile aktarılmasıdır. Ayrıca ES uygunluk fonksiyonlarına bakmaksızın rastgele seçimler gerçekleştirilen bir algoritmadır [14].

Fizik kurallarına dayalı algoritmalar ise fizik kurallarını esas alan algoritmalardır [11]. Bu algoritmalar fiziksel modellemeler, maddelerden elde edilen sabit veriler, bu verilere bağlı hipotezlerin doğruluğunun kanıtlanmasıyla kurallaşan prensipler ve bu prensiplerin birbirleriyle bağlantılarını temel algoritma hipotezini oluşturacak şekilde kullanırlar. Bu algoritmalara örnek olarak benzetilmiş tavlama algoritması (SA) [16], yerçekimsel arama algoritması (GSA) [17] ve elektromanyetik alan optimizasyonu [18] verilebilir.

SA 1983 tarihinde Kirkpatrick tarafından geliştirilmiştir [16]. Katıların ısıtılması, soğumaya bırakılması kısaca "tavlama" temeline dayanır. Bu algoritma kabul fonksiyonuna verilen komşu değerlerde artışa sebep olan değerlerin belirlenmesi ve elenmesi esasında çalışmaktadır. Başlangıç çözümü değerlendirilip, kabul edildiği takdirde ısı değişimi yaşanmışsa aramanın sona ulaşabileceği, aksi takdirde çözümün güncellenerek iterasyona tekrar parametre olarak verilmesi ile çalışmaktadır [16].

GSA 2009 yılında Rashedi tarafından geliştirilmiştir [17]. Bu algoritmada aday çözümler, kütleler olarak adlandırılır. Her kütle dört özelliğe sahiptir. Bunlar pozisyon, harekete devam etme yönelimli kütle, aktif yerçekimsel kütle ve pasif yerçekimsel kütledir [17]. Çalışma prensibi, belirlenen uzay sınırları içerisine yerleştirilen cisimlerin, uygunluk fonksiyonu aracılığıyla alınan değerlerin ortalaması; minimizasyon için en küçük, maksimizasyon için en büyüğü alınarak sonraki iterasyona aktarılmasını amaçlamaktadır. Hem lokal hem de evrensel hesaplamalar yapılarak, her iterasyonda evrensel minimum/maksimum değerler güncellenir [17].

Elektromanyetik alan optimizasyonu 2016 tarihinde Abedinpourshotorban tarafından geliştirilmiştir [18]. Bu algoritma elektromıknatısların işleyişi üzerine kurulmuş bir

algoritmadır. Bir telin içerisinden akım geçtiği zaman bu telin üzerinde bir elektromanyetik alan oluşur. Yük kutupları aynı olanlar birbirini iterken, zıt kutuplu olanlar ise birbirini çekerler [18]. Bu algoritmada kutupların yapmış oldukları bu hareketler altın oran kullanılarak dengelenir. Algoritmanın uygunluk fonksiyonundaki en iyi değerler pozitif yüklü kutuplar, en kötü değerler ise negatif yüklüler olarak kabul edilmiştir.

Bunlar dışında fizik kurallarına dayanan algoritmalara, ısı transfer arama (HTS) [19], Gaz Brownian hareketi (GBMO) [20], optikten esinlenilen optimizasyon (OIO) [21], ağırlıklı süperpozisyon çekimi (WSA) [22] optimizasyon algoritmaları verilebilir.

Sezgisel algoritmalar karmaşık problemleri basitleştirerek, çözme amacıyla tasarlanmış ve gerçek hayatta kullanılan yaklaşımlara yakın algoritmalardır [11]. Sezgisel algoritmalar en kısa sürede en iyi çözümü bulma garantisi vermez. Bunun yerine en kısa sürede çözüm veren, ancak en verimli olmayan ya da en verimli sonuca uzun sürede ulaşan algoritmalardır. Zaman ve verim parametreleri sezgisel algoritmalar için ters orantılıdır. En hızlı sonuca ulaşmak için kullanılan bu algoritmaların, problemi her zaman çözmesi beklenemez. Başarılı sezgisel optimizasyon algoritmaları minimum sürede maksimum verim sağlayan algoritmalardır. Karınca kolonisi optimizasyonu (ACO) [23], parçacık sürüsü optimizasyonu (PSO) [24], kasırga temelli optimizasyon algoritması [25], orman optimizasyon algoritması (FOA) [26] ve kara delik optimizasyon algoritması [27] sezgisel optimizasyon algoritmalarına örnek verilebilir.

ACO Dorigo tarafından 1997 de geliştirilmiştir [23]. Karıncalar bulundukları yerlerden yuvalarına en kısa yolu bulma sezisine sahiplerdir ve bunu görme duyularını kullanmadan başarmaktadırlar [23]. Buna ek olarak karıncalar çevresel değişimlere adapte olma konusunda da üstün bir başarıya sahiplerdir. Örneğin, yiyecek kaynağından yuvalarına dönen karıncaların önüne bir engel konulduğunda, gerçekleştirdikleri davranışlar izlenerek şu sonuca varılmıştır; karıncalar öncelikle rastlantısal olarak iki yoldan birini tercih ederler. Devam eden süreçte uzun yolu kullanan karıncalar da kısa yolu kullanan karıncaların artması ile birlikte yoldaki izleri inceleyerek, izlerin daha yoğun olduğu kısa yolu kullanmaya başlarlar. Kısa süre sonra bütün karıncaların kısa yolu kullanmaya başlandığı gözlemlenmiştir [23]. ACO hafıza sahibi, en kısa yolu hesaplayabilen yapay karıncaların parçalı zamanda iterasyon yoluyla minimum uzaklıktaki yolu seçene kadar süren bir sezgisel algoritmadır [23]. Karıncaların gidecekleri yolun seçimi rastlantısal olarak karıncalara bırakılmıştır. Her iterasyon sonucunda hesaplanan evrensel kısa yol,

bu rastlantı fonksiyonuna bir parametre olarak verilir. ACO başlangıç feromen değerlerinin belirlenmesi ile başlar. Sonraki adımda karıncalar rastlantısal olarak yerleştirilir. Her karınca gideceği yeri lokal arama olasılığıyla seçerek, seçtiği yola gider. Sonrasında her karınca için gidilen yol miktarı hesaplanır ve lokal feromen tekrar atanır. En kısa yol hesaplanır ve global feromen güncellenerek, maksimum iterasyon sayısına kadar bu adımlar tekrarlanır [23]. ACO için uygun karınca sayısı probleme göre değişiklik göstermektedir, ancak karınca sayısının artması daha kesin sonuçlar vermekle birlikte işlem sayısını arttırdığı için hesaplama süresini uzatmaktadır [23].

PSO algoritması kuşlar, balıklar ve sürü halinde yaşayan diğer hayvanların bilgi paylaşımı metoduyla doğa koşullarına adaptasyonlarını arttırmaları ve hayatta kalmak için gereken temel yeteneklere erişmelerinden esinlenerek ilk olarak 1995 yılında Kennedy tarafından geliştirilmiştir [24]. Önerilen PSO algoritması popülasyon temelli bir sezgisel optimizasyon algoritmasıdır. PSO her bir parçacığı bir kuş olarak modelleyen ve modellenen hafıza sahibi kuşların iterasyon ya da başarı kondisyonu sağlanana kadar döngüye dahil edilerek, her iterasyonda lokal ve evrensel uzaklıkları kuşlarda meydana gelen hız değişikliğine bağlı olarak hesaplanan algoritmadır [24]. Bu algoritmanın başlangıç nüfusu, sürünün her parçacığı için hız ve pozisyonlar rastgele seçilerek oluşturulur. Sürüdeki bütün parçacıkların uygunluk değeri hesaplanır [24]. Her iterasyonda en iyi değerler bir önceki iterasyon ile kıyaslanır ve daha iyi ise yer değiştirilir. Sonraki adımda en iyi değerler kendi içlerinde karşılaştırılarak en iyi olan, global en iyi olarak belirlenir. Hız ve pozisyon değerleri yenilenir. Bu işlemler maksimum iterasyon ya da durdurma şartı sağlanana kadar devam eder [24].

Kasırga temelli optimizasyon algoritması 2014 tarihinde Isamil Rbouh tarafından geliştirilmiştir [25]. Bu algoritma, kasırganın merkezinin etrafında toplanan bulutların, merkeze yakın noktalardaki yoğun yağmur ve rüzgarın gözlemlenmesine dayalı bir optimizasyon algoritmasıdır. Bu algoritmanın temelinde rüzgar ve atmosferin kasırga etrafında aldıkları şekillerin incelenmesi yer almaktadır [25]. Kasırga temelli optimizasyon algoritması gözcünün bakış açısının rastgele pozisyonlanması ile başlatılır. Sonrasında fonksiyon ile değişen gözcü bakış açısı rüzgarın değişimi ile sınanır. Sınama sonucunda üç olasılık bulunmaktadır; değişim görüş açısı dışındadır, değişim basıncı gözcü basıncından daha yüksek ise rüzgar pozisyonu yeni gözcü bakış açısı olarak değiştirilir ya da değişim fonksiyonu ile rüzgar sonraki pozisyona getirilir. İterasyon önceden belirlenen maksimuma ulaştığında en iyi sonuç gözcünün bakış açısı kabul edilir

[25].

Ormanlar canlı ve dinamik biyolojik topluluklardan oluşan sistemlerdir. FOA 2014 tarihinde Ghaemi tarafından geliştirilmiştir [26]. Bu algoritma bir ormanda bulunan ağaçların fiziksel, biyolojik ve kimyasal değişkenlerden etkilendikleri, aynalama adı verilen bir yöntem kullanmaktadır [26]. Gelişme, çoğalma ve ölme olmak üzere üç fazdan oluşan bu algoritma yerel minimumlara takılı kalmamak, erken yakınsamadan kaçınmak ve çeşitliliği arttırmak için seçilimin haricinde doğal ölüm fazına sahiptir [26]. FOA, akış planı habitat popülasyonuna göre belirlenen ağaç sayısı sınırını geçmemesi şartı ile ağaçların iterasyon veya başarı kondisyonunu sağlayana kadar tohumlanmasını öngörür. Algoritma, bu ağaçların ölümüne kadar her iterasyonda, diğer ağaçlardan daha iyi olanların bir sonrası iterasyona aktarılması ile gerçekleşir [26].

Kara delik optimizasyon algoritması 2013 yılında Hatamlou tarafından geliştirilmiştir [27]. Bu algoritma için başlangıçta belirlenen bir ilk çözüm vardır. Bu çözümler yıldızlarla ilişkilendirilir. Kara delikler ise konumları sabit varsayılarak, fonksiyonun minimum noktalarını belirlerler [27]. İleri iterasyon adımlarında yıldızlar, kara deliğe doğru hareket ederler. Eğer kara delikten daha düşük uygunluk değerine sahip bir değer elde edilirse, kara delikler güncellenerek iterasyonun devamlığı sağlanır.

Bunlar dışında sezgisel optimizasyon algoritmalarına örnek olarak, yapay arı kolonisi (ABC) [28], meyve sineği optimizasyon algoritması [29], levy uçuşuna dayalı guguk kuşu optimizasyonu [30], krill sürü optimizasyon algoritması[31], bakteriyel besin arama optimizasyon algoritması [32], yarasa algoritması [33], ateş böceği algoritması [34], aslan algoritması [35], gri kurt algoritması (GWA) [36] verilebilir. Daha güncel algoritmalara örnek olarak karınca aslanı optimizasyon algoritması (ALO) [37] ve balina optimizasyon algoritması (WOA) [38] verilebilir.

ALO algoritması 2015 yılında Mirjalili tarafından geliştirilmiştir [37]. Karınca aslanı, karıncagiller ailesine mensup bir üyedir. İsimlerini alma sebepleri ise avlanma yapılarına ve en sevdikleri av olan karıncalara dayanmaktadır. Karınca aslanlarının avlanma teknikleri tuzak kurma üzerine inşa edilmiştir. Kum içerisine huni şeklinde bir çukur kazarak, tuzaklarını hazır hale getirdikten sonra bu huni şeklindeki çukurun en alt kısmına konumlanarak avlarını beklemeye koyulurlar. Bu noktadan sonrasında ise algoritmada altı ayrı aşama bulunmaktadır. Birinci aşamada ALO, karıncaların yürüyüşlerini ve tuzağa düşme olasılıklarını hesaplayabilmek adına rastgele yürüyüş algoritmasını her

iterasyonda kullanır. İkinci evrede karıncaların tuzağa düşmesi üzerine kurulmuş bir yapı vardır. Üçüncü aşama yukarıda bahsedilen tuzağı kurma aşamasıdır [37]. Bu aşamada karınca aslanlarının tuzaklarını nereye kuracaklarını bulmak için rulet tekeri algoritmalarından bir tanesi kullanılmıştır. Dördüncü aşamada ise karıncaların kurulmuş olan tuzağa düşmeleri yer alır [37]. Ancak bu aşamada karıncalar kaçmaya çalışırlar ve karınca aslanı çenesiyle kum fırlatarak karıncayı tekrar tuzağa çekmeye çalışan bir davranış sergiler. Bu noktada optimizasyon algoritmalarında sıkça görülen mutasyon işlemlerine tabi tutulduğu söylenebilir [37]. Beşinci adımda ise karıncanın yakalanması sebebiyle bir sonraki iterasyona geçiş için hazırlıklar yapılır ve bir önceki konumundan daha iyi olan yeni bir yer belirlenir. Altıncı aşama ise yine optimizasyon algoritmalarında sıkça görülen elitizim bölümüdür. Elitizm var olan sürü içerisindeki en iyi konuma sahip üyeyi (bu durumda karınca aslanını) saklayarak, diğer üyeleri en iyi üyeye doğru yönlendirir. En iyiye yönelirken de, bu noktayı daha verimli bir şekilde kullanmayı hedeflemektedir. Daha sonrasında ise iterasyon adımları tekrarlanarak, bir sonraki aşamaya yeni elit üyenin avantajları aktarılarak devam edilir [37].

WOA 2016 yılında Mirjalili tarafından geliştirilmiştir [38]. Bu algoritma balinaların ilginç avlanma teknikleri üzerine kurulmuştur. Bu tekniğe ise baloncuk ağı adı verilir. Balinalar küçük ve grup halinde dolaşan balıklarla beslenirler. Baloncuk ağı avlanma tekniğinde, balinalar grup halindeki balıkların etraflarında dönerek ve baloncuklar çıkartarak balıkları bu baloncuk ağının içerisine hapsederler. Mirjalili bu ağı 9 rakamına benzetmiştir. Balinalar avlarının olduğu alanı belirleyerek, avlarını bir çember içerisine alırlar. Algoritma bu çember içerisindeki aday çözüm olarak minimuma en yakın avı kabul eder. Saldırı kısmında iki tür mekanizma bulunmaktadır. Bunlar küçülen çember yapısı ve spiral içerisinde pozisyon yenilemedir. Hedef belirlendikten sonra ise avı arayan bir fonksiyon devreye girer ve iterasyon bu şekilde devam eder [38].

Bu tez çalışmasında da doğadan esinlenilen, nüfus tabanlı bir optimizasyon algoritması olan kar tanesi optimizasyon (SFO) algoritması önerilmektedir. Bir sonraki bölümde bu algoritmanın temel ilkeleri ve formülasyonları sunulmuştur. Dördüncü bölümde ise örnek benchmark fonksiyonların bu algoritma ile çözümüne yer verilmiştir.

2. KAR TANESİ OPTİMİZASYON ALGORİTMASI

Kar, 0 °C'den daha düşük sıcaklıklarda havada bulunan su buharının anlık olarak soğuyarak kristalleşmesi ile oluşan tanelere verilen isimdir. Bu taneler bazı durumlarda diğer su damlalarını kendisine çekerek daha büyük kar tanelerini oluşturabilmektedir [39]. Kar taneleri çok hafiftir, bu sebeple saatte ortalama 1.5 kilometrelik bir düşey hıza sahiplerdir [39]. Bu hız kar tanelerinin büyüklüğüne ve şekline göre değişebilmektedir. Kar tanelerinin en dikkat çekici özelliği her bir kar tanesinin farklı bir şekle sahip olmasıdır [39]. Bu özellikleri ile tarihte birçok bilim insanının da araştırma konusu olmuştur. Johannes Kepler, Rene Descartes, Robert Hooke ile 1600'lerde başlayan çalışmalar [40][41] modern ve sistematik bir hale 1950'li yıllarda Ukichiro Nakaya [42] tarafından getirilmiştir. Son yıllarda ise Kenneth Libbrecht yaptığı çalışmalar ile atmosfer koşullarının kar tanelerinin oluşumu sırasındaki şekillenmelerini etkilediğini gösteren bulgulara ulaşmıştır. Bu çalışmalara göre havadaki nem oranı yükseldikçe, kar tanesi şekillerinin karmaşıklığı da artar.

Kar taneleri yeryüzünden yükselen nemin soğuk hava tabakası ile karşılaşması sonucunda oluşmaktadırlar. Oluşan kar kristallerinin birleşerek meydana getirdiği kar taneleri yeryüzüne düşmeye başlamaktadır. Kar taneleri düşüşleri esnasında bazen sıcak hava ile karşılaşabilmektedir. Bu durumda sıvı hale geçerek yağmura dönüşmektedirler. Bu düşüş sırasında kar taneleri hava sıcaklığı, nem, rüzgar, basınç, sürtünme kuvveti ve yerçekimi gibi faktörlerden etkilenmektedir [39].

Isınan hava yükselir. Soğuk hava ise sıcak havadan daha yoğun bir yapıya sahiptir. Bu durumda ısınıp yükselen havanın yerini soğuk hava doldurur. Bu olayda rol alan yoğunluk farkı aslında bir basınç farkının oluşmasına da neden olmaktadır. Rüzgar ise yüksek basınç alanından düşük basınç alanına doğru eser. Bu bulgulara dayanarak da rüzgar hızının yükseklikle birlikte arttığını söyleyebiliriz.

SFO algoritmasında bu etkin faktörlerden sadece rüzgar ve yerçekiminin aktif olduğu, geri kalan kuvvet ve faktörlerin etkinliklerinin karın yağışı sırasındaki hız ve yön değişkenlerini etkilemediği varsayılmıştır. Buna göre SFO algoritması ile çözülecek problemlerin arama uzayı, kar tanelerinin düşmeye başladığı yer ile ile bu kar tanelerinin

ulaşabileceği en alçak nokta olan deniz seviyesinin arasında kalan bütün alandır. Yukarıda belirtilen alanlara ait faktörler Şekil 2.1'de gösterilmiştir.

Şekil 2.1. SFO algoritmasında kar taneleri ve bu kar tanelerine etki eden kuvvetler

SFO algoritmasına göre kar taneleri yeryüzüne düşerken bulunduğu yüksekliğe göre farklı rüzgarlar ile karşılaşmaktadır. Farklı rüzgarların etkisine giren kar taneleri arama uzayında yatay düzlemde hareket ederken, yer çekimi etkisiyle de dikey eksende dolaşmaktadır. Kar tanelerinin bu hareketi SFO algoritmasına göre problem tek boyuta indirgendiğinde, iki farklı şekilde modellenir. Bunlardan birincisi kar tanelerinin problemin arama uzayında dolaşma işlemi ve yatay eksendeki hareketi ile modellenir. Diğeri ise düşey eksendeki hareketin her bir adayın uygunluk değerine göre modellenmesidir. Buna bağlı olarak algoritma için düşey düzlemdeki devinimler boyut temsil etmemektedir. Bunun yerine en iyiye yaklaşma değeri olarak tanımlanmaktadır. Bahsi geçen yaklaşım doğrultusunda SFO algoritması bir boyutlu problemlere ek olarak, birden çok boyutu olan problemlere de uygulanabilir hale gelmektedir. Bir diğer şekilde şöyle açıklanabilir; algoritmanın arama uzayını taramak için rüzgâr kullanılırken, yer çekimi en iyiye yönelimi sağlamaktadır.

SFO algoritması nüfus tabanlı bir algoritma olma özelliğine sahiptir. Her bir aday çözüm, atmosferden düşen bir kar tanesini ifade etmektedir; ancak gerçek kar tanelerinden farklı olarak, SFO algoritmasında her bir kar tanesinin n adet boyuta sahip olabileceği düşünülmüştür.

$$S = \begin{bmatrix} s_{i,j} & \cdots & s_{i,d} \\ \vdots & \ddots & \vdots \\ s_{n,1} & \cdots & s_{n,d} \end{bmatrix} \tag{2.1}$$

Denklemde;

S: kar tanelerinin oluşturduğu nüfusu,

s_i: aday çözümlerden her birini,

$i = 1,2,3, \ldots . d$: problemin boyutunu temsil etmektedir.

SFO algoritmasında her bir döngüde kar tanelerinine ait uygunluk değerleri hesaplanmaktadır. Sonrasında ulaşılan bu uygunluk değerleri vasıtasıyla bir rüzgâr değeri elde edilir. Bunun için ise kar tanelerinin belirli bir rüzgârın etkisi altında oldukları varsayılmıştır. Bu rüzgâr, kar taneleri yere düşene kadar geçen süre içerisinde her an farklı büyüklük ve yönlerde etki etmektedir. Bu etkiyi modelleyebilmek için literatürde power-law kuralı olarak geçen, rüzgârın yükseklik seviyesine bağlı olarak hızını veren bir güç fonksiyonu kullanılmıştır [43]. Bu formül referans seviyedeki bir rüzgâr hızını kullanarak, diğer seviyelerdeki rüzgâr hızlarını hesaplar [43].

$$V(z) = Vr(\frac{z}{z_r})^2 \tag{2.2}$$

Bu denklemde kullanılmış olan değişkenler:

Vr: Referans yükseklikte esen rüzgarın hızı,

z_r: Referans seviyedeki yüksekliği,

z: Hız bilgisine göre hesaplanmak istenilen yüksekliği,

$V(z)$: İlgili seviyedeki hesaplanmış olan rüzgar hızını ifade eder [43].

Bu denklem rastgeleliğin lokal minimum ve maksimumlara takılması durumu göz önüne alındığında, yeterlilik düzeyi düşük kalmaktadır. Bu formül SFO algoritmasında işlemlere rasgelelik ekleme adına formül aşağıdaki gibi değiştirilmiştir.

$$R_{i,j} = r * \beta * \left(V_r * abs\left(\frac{(S_{i,j})}{1 + Sr_j} \right) \right) \tag{2.3}$$

Denklemde;

r: [0,1] aralığında normal dağılımlı rastgele bir sayıyı,

Vr: Referans seviyedeki yükseklik değerini temsil etmektedir.

SFO algoritması için V_r hesaplanırken, bir rulet tekeri algoritması yardımıyla nüfusun amaç fonksiyon değerlerinden aday bir çözüm seçilir. Bu seçim sırasında rulet tekeri algoritması rastgeleliğin yanında amaç fonksiyon değeri odaklı bir seçim gerçekleştirmek için kullanılmıştır. V_r seçilen aday çözümün uygunluk değeri kullanılarak hesaplanır.

$$V_r = \frac{U_v(S_r)}{max(U_v)} \tag{2.4}$$

$$V_r = \begin{bmatrix} f(s_{1,1} & \cdots & s_{1,j}) \\ \vdots & \ddots & \vdots \\ f(s_{i,1} & \cdots & s_{i,j}) \end{bmatrix} \tag{2.5}$$

Denklemde;

U_v: SFO algoritması ile çözüme ulaşması hedeflenen probleme ait amaç fonksiyonunun S nüfusu için üretmiş olduğu uygunluk değerleri vektörüdür,

$U_v(S_r)$: Rulet tekeri algoritması yardımıyla uygunluk değerleri göz önüne alınarak seçilmiş kar tanelerinin uygunluk değerini,

max(U_v): O anki nüfusa ait uygunluk değerleri içerisindeki en kötü değeri göstermektedir (minimizasyon yapılırken max olarak alınır),

S_r: Seçilen uygunluk değerine ait aday çözümdür,

abs: Mutlak değer fonksiyonudur.

Denklem 2.3'te yer alan β katsayısı SFO algoritmasının kullanıcı tarafından ayarlanan ilk parametresidir. Bir kar tanesi arama uzayını dolaşırken adım büyüklüğünü ayarlamak için bu katsayı tanımlanmıştır. β parametresi kullanıcı tarafından çözülmesi istenilen probleme uyacak şekilde ayarlanır.

Denklem 2.6'da her bir aday çözüm için her bir boyutta ona etki eden rüzgar bilgisi $R_{i,j}$ hesaplandıktan sonra yeni nüfus aşağıdaki gibi elde edilir.

$$V_r = \begin{bmatrix} f(s_{1,1} & \cdots & s_{1,j}) \\ \vdots & \ddots & \vdots \\ f(s_{i,1} & \cdots & s_{i,j}) \end{bmatrix} ; \; R_{i,j} = \begin{cases} R_{i,j} \; eğer \; r < 0.5 \\ -R_{i,j} \; eğer \; r > 0.5 \end{cases} \tag{2.6}$$

$$S_2 = S + R \tag{2.7}$$

SFO algoritmasının bir döngüsünde yeni nüfus oluştuktan sonra nüfusun uygunluk değerleri de güncellenir. Yapılan güncellemeden sonra daha iyi uygunluk değerine sahip olan kar tanelerinin ağırlıklarının arttığı gözlemlenmektedir. Ağırlığı artan kar tanelerinin, diğer kar tanelerinden daha aşağı noktalara yöneldiği düşünülmüştür. Bu durum için matematiksel formülasyon aşağıdaki şekilde verilmiştir.

$$S_3 = Y(S_2) \tag{2.8}$$

Denklemde $S3$, $S2$ nüfusunun yerçekimi etkisiyle kar tanelerinin kendi aralarında yer değiştirmesi sonucu oluşmuş yeni nüfustur. Yer değiştirme işlemi sonrasında uygunluk değeri daha iyi olanlar nüfus içerisinde daha alt sıralarda yer almaktadırlar. Bu şekilde atmosferden yeryüzüne kadar olan alanda yükseklik açısından sıralanmış kar tanelerinin konumları ile nüfus içerisinde birinci sıradan sonuncu sıraya kadar sıralanmış aday çözümler arasında bir konumsal ilişki kurulmaktadır. Bu eşleştirme ve sıralama işlemi bir ruket tekeri algoritması yardımıyla gerçekleştirilmektedir. Denklemde yer alan Y fonksiyonu rulet tekeri algoritmasını ifade etmektedir.

SFO algoritmasında S3 nüfusunun elde edilmesinden sonra bir sonraki iterasyona geçmeden önce son bir işlem daha gerçekleştirilir. Elbette kar taneleri yeryüzüne düşerken, binalar veya ağaçlar gibi engellere takılabilmektedir. Bu durum engele takılan kar tanesi arama uzayı içerisindeki hareketinin son bulduğunu göstermektedir. Bu işlem ile yeryüzüne ulaşamayacağı düşünülen aday çözümler modellenmektedir. SFO algoritmasında bu durumu modelleyebilmek için her bir iterasyonda rastgele seçilen kar tanelerinin yok olduğu varsayılmaktadır. Yok olan kar tanelerinin yerine ise nüfus büyüklüğünü muhafaza etmek için rastgele olarak yeni aday çözümler tanımlanmaktadır.

$$S_4 = M(S_3, a) \tag{2.9}$$

Denklemde M fonksiyonu $S3$ nüfusundan rastgele seçilecek olan aday çözümlerin yeniden oluşturulmasına yarayan bir fonksiyondur. Bu fonksiyonda α değeri, rastgele seçilecek kar tanelerinin oranını belirtmektedir. α katsayısı SFO algoritmasının kullanıcı tarafından ayarlanan ikinci parametresidir. Buna göre SFO algoritmasında kullanıcının belirleyebildiği iki parametre vardır.

SFO algoritmasında $S4$ nüfusu için uygunluk değeri hesaplanmakta ve bu uygunluk değerlerine göre S nüfusu aşağıdaki gibi güncellenmektedir.

$$Eğer\ U_v(S_4)_{i,1} > U_v(S)_{i,1}\ ise\ S_i = S_{4_i} \quad (2.10)$$

Bu aşamadan sonra SFO algoritmasında döngü bir sonraki aşamaya geçer. SFO algoritmasının sözde kodu aşağıda verilmiştir.

SFO algoritması

S nüfusu üretilir.

For i=1:döngü sayısı

U_v uygunluk vektörü hesaplanır.

S_r rulet tekeri algoritması yardımıyla seçilir.

R rüzgar fonksiyonu oluşturulur.

Eğer (r < 0.5) ise

$R_{i,j}$ negatif olarak kabul edilir.

Eğer ($r \geq 0.5$) ise

$R_{i,j}$ pozitif olarak kabul edilir.

S_2 nüfusu, R ve S'nin toplanması ile oluşturulur.

S_2 nüfusu sıralama fonksiyonuna gönderilir.

S_3 nüfusu sıralama fonksiyonunun çıktısı olarak elde edilir.

S_3 M (mutasyon) fonksiyonunda ki işlemlere tabii tutulur ve S_4 elde edilir.

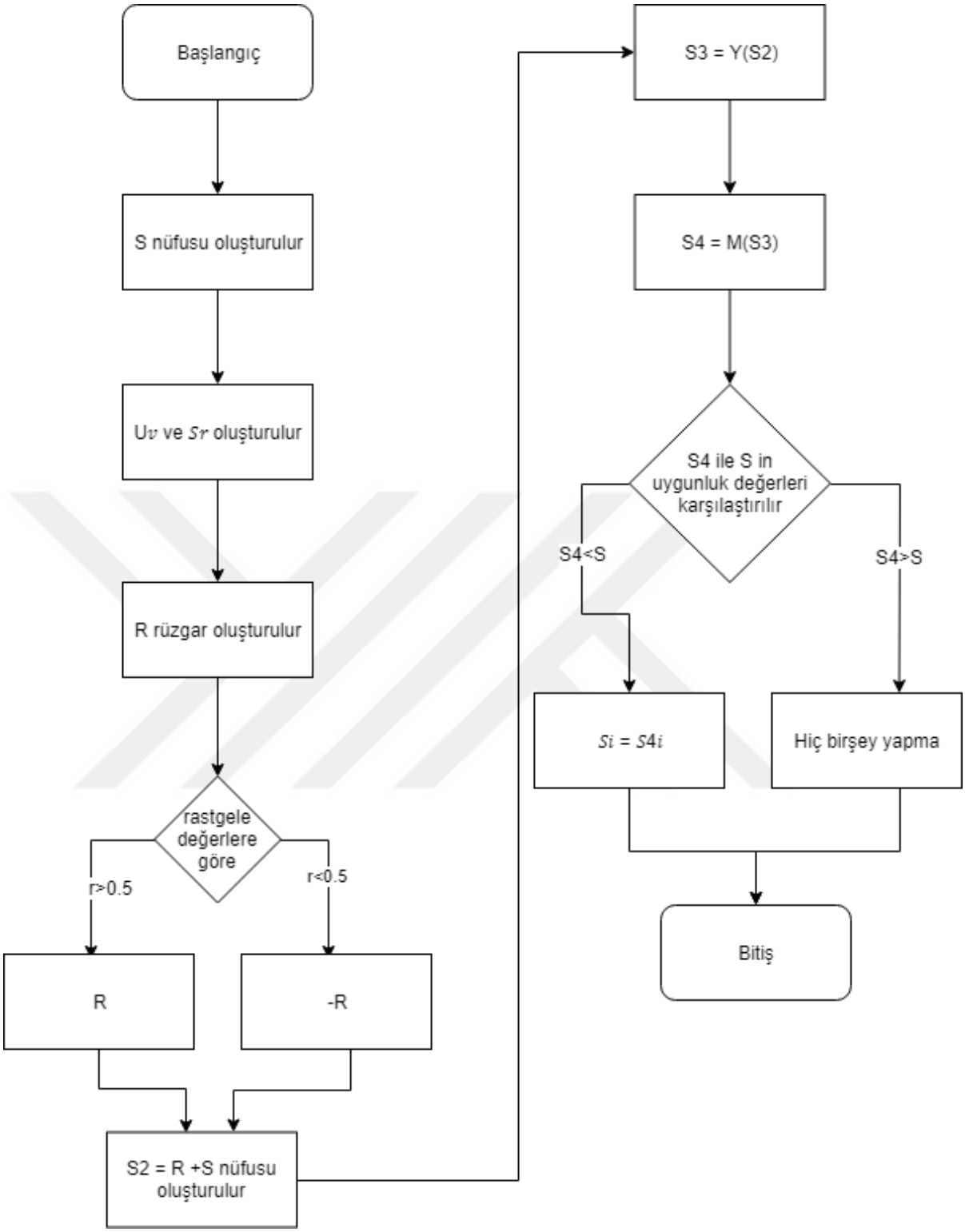
S_4 nüfusunun uygunluk değerleri S nüfusu ile karşılaştırılır.

Eğer ($U_v(S_4)_{i,1} < U_v(S)_{i,1}$ ise

$S_i = S_{4i}$

For end

Önerilen SFO algoritmasının akış şeması Şekil 2.2'deki gibidir.



Şekil 2.2. SFO algoritması akış diyagramı.

3. BENCHMARK FONKSİYONLARIN SFO İLE ÇÖZÜMÜ

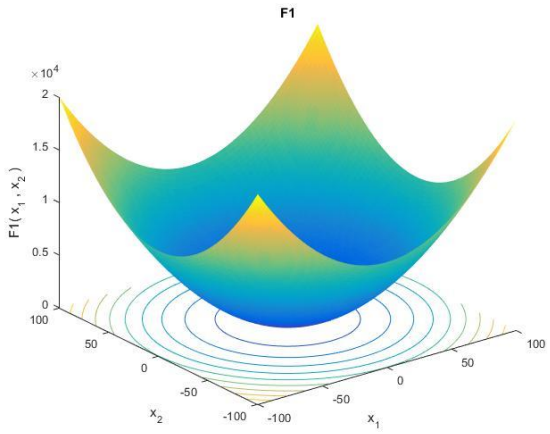
Önerilen SFO algoritmasının performansının test edilmesi amacıyla 13 tane benchmark fonksiyon seçilmiştir. Bu fonksiyonlara ilişkin matematiksel ifadeler ve grafikler aşağıda sunulmuştur.

3.1. FONKSİYON GRAFİKLERİ

3.1.1. F1 Fonksiyonu

$$F_1(x) = \sum_{i=1}^{n} x_i^2 \tag{3.1}$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.1'de gösterilmiştir.



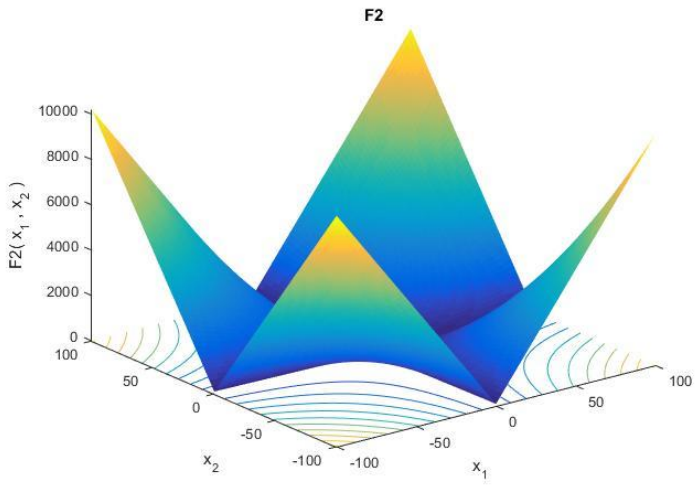
Şekil 3.1. F1 fonksiyonu 3D grafiği.

3.1.2. F2 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_2(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i| \tag{3.2}$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.2'de gösterilmiştir.



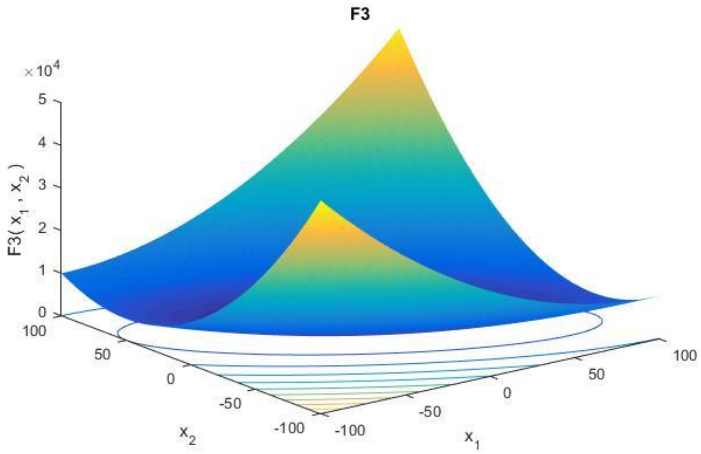
Şekil 3.2. F2 fonksiyonu 3D grafiği.

3.1.3. F3 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_3(x) = \sum_{i=1}^{n} (\sum_{j-1}^{i} x_j)^2 \tag{3.3}$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.3'de gösterilmiştir.



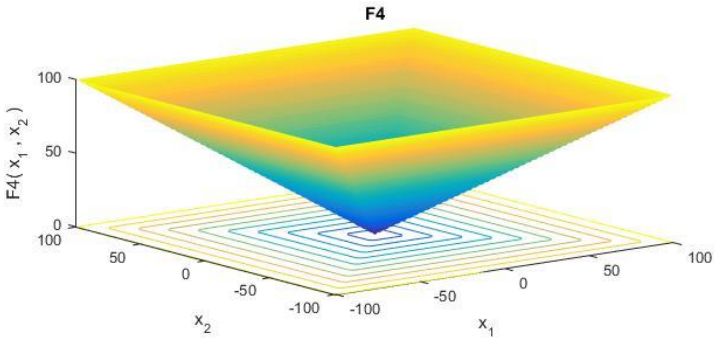
Şekil 3.3. F3 fonksiyonu 3D grafiği.

3.1.4. F4 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_4(x) = \max\{|x_i| \; 1 \leq i \leq n\} \tag{3.4}$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.4'de gösterilmiştir.



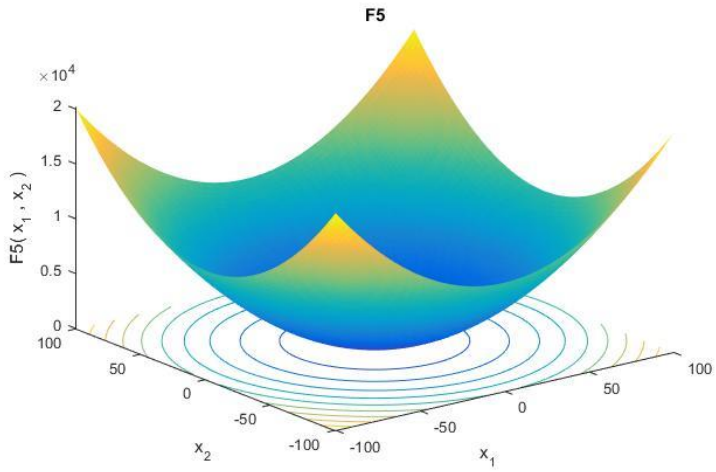
Şekil 3.4. F4 fonksiyonu 3D grafiği.

3.1.5. F5 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_5(x) = \sum_{i=1}^{n-1} (|x_i + 0.5|)^2 \tag{3.5}$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.5'de gösterilmiştir.



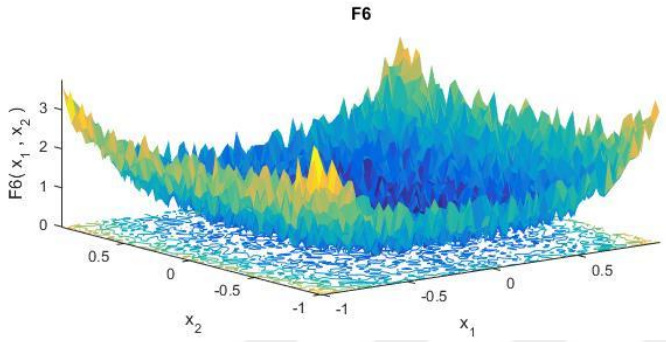
Şekil 3.5. F5 fonksiyonu 3D grafiği.

3.1.6. F6 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_6(x) = \sum_{i=1}^{n} ix_i^4 + random[0,1) \quad (3.6)$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.6'da gösterilmiştir.



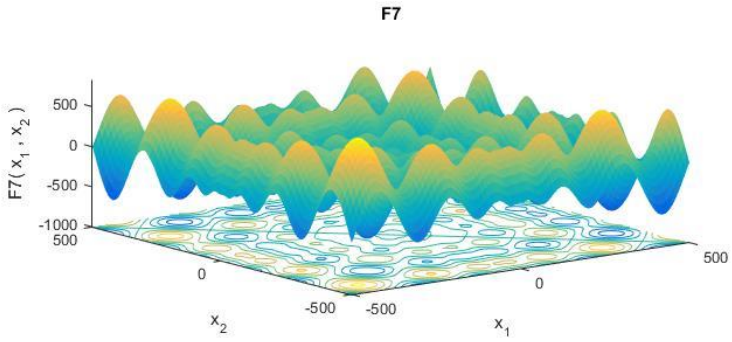
Şekil 3.6. F6 fonksiyonu 3D grafiği.

3.1.7. F7 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_7(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{|x_i|}) \quad (3.7)$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.7'de gösterilmiştir.



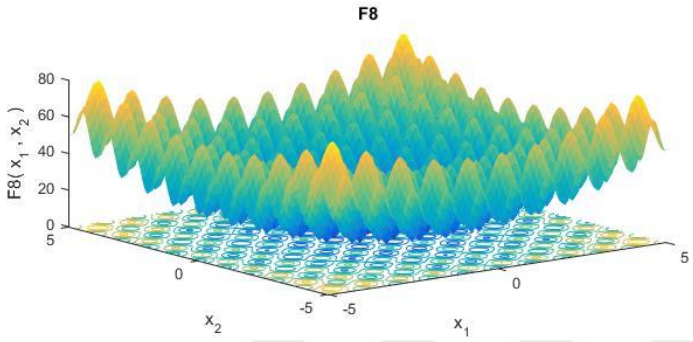
Şekil 3.7. F7 fonksiyonu 3D grafiği.

3.1.8. F8 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_8(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10] \tag{3.8}$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.8'de gösterilmiştir.



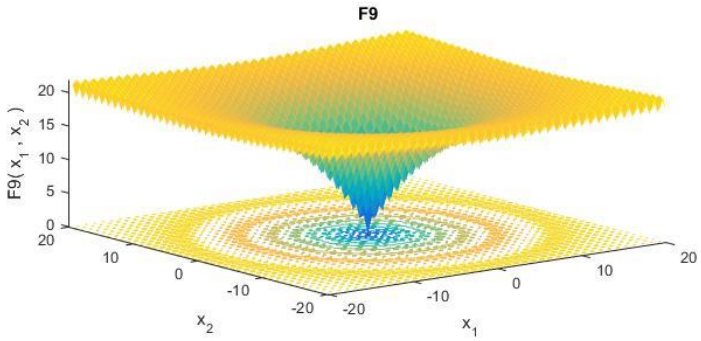
Şekil 3.8. F8 fonksiyonu 3D grafiği.

3.1.9. F9 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_9(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \tag{3.9}$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.9'da gösterilmiştir.



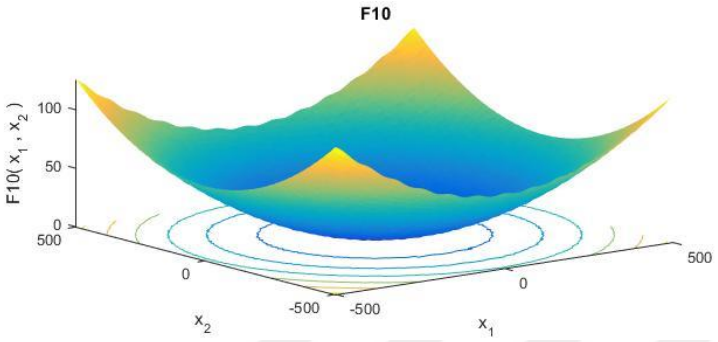
Şekil 3.9. F9 fonksiyonu 3D grafiği.

3.1.10. F10 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_{10}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{3.10}$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.10'da gösterilmiştir.



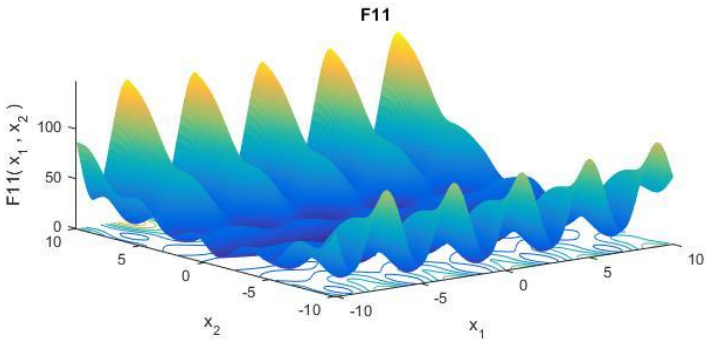
Şekil 3.10. F10 fonksiyonu 3D grafiği.

3.1.11. F11 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_{11}(x) = \frac{\pi}{n}\left\{10\sin(\pi y_1) + \sum_{i=1}^{n-1}(y_i)^2[1 + 10sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} + \sum_{i=1}^{n}\mu(x_i, 10,100,4) \tag{3.11}$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.11'de gösterilmiştir.



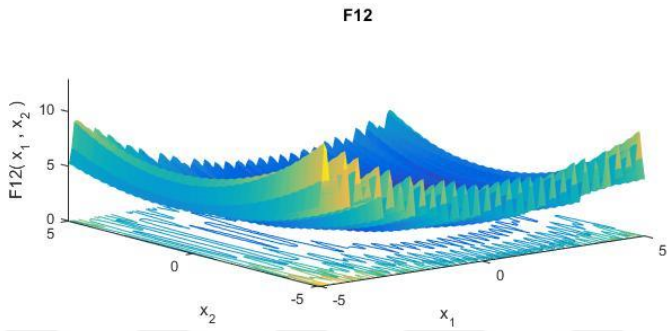
Şekil 3.11. F11 fonksiyonu 3D grafiği.

3.1.12. F12 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_{12}(x) = 0.1\Bigg\{ sin^2(3\pi x_1) + \sum_{i=1}^{n}(x_i - 1)^2[1 + sin^2(3\pi x_i + 1)] + (x_n - 1)^2[1 + sin^2(2\pi x_n)]\Bigg\} + \sum_{i=1}^{n}\mu(x_i, 5,100,4) \quad (3.12)$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.12'de gösterilmiştir.



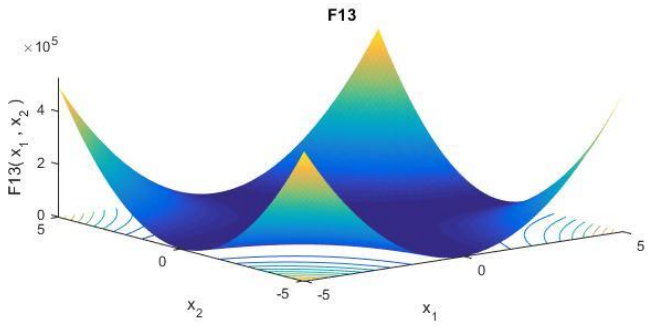
Şekil 3.12. F12 fonksiyonu 3D grafiği.

3.1.13. F13 Fonksiyonu

Fonksiyonun formülü aşağıda verilmiştir [37].

$$F_{13}(x) = \sum_{i=1}^{11}[a_i - \frac{x_i(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}]^2 \quad (3.13)$$

Fonksiyonun 3 boyutlu grafiği Şekil 3.13'de gösterilmiştir.



Şekil 3.13. F13 fonksiyonu 3D grafiği.

Bu fonksiyonların detayları aşağıdaki tabloda gösterilmiştir.

Çizelge 3.1. Benchmark fonksiyonlar.

Fonksiyonlar	Değerler		
	Boyut	Aralık	En küçük değer
$F_1(x) = \sum_{i=1}^{n} x_i^2$	10	[-100,100]	0
$F_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	10	[-10,10]	0
$F_3(x) = \sum_{i=1}^{n} (\sum_{j-1}^{i} x_j)^2$	10	[-100,100]	0
$F_4(x) = \max\{\lvert x_i \rvert \, 1 \le i \le n\}$	10	[-100,100]	0
$F_5(x) = \sum_{i=1}^{n-1} (\lvert x_i + 0.5 \rvert)^2$	10	[-100,100]	0
$F_6(x) = \sum_{i=1}^{n} i x_i^4 + random[0,1)$	10	[-1.28,1.28]	0
$F_7(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	10	[-500,500]	-418.982*n
$F_8(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	10	[-5.12,5.12]	0
$F_9(x) = -20 \exp\left(-0.2 \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e$	10	[-32,32]	0
$F_{10}(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	10	[-600,600]	0
$F_{11}(x) = \frac{\pi}{n}\left\{10 \sin(\pi y_1) + \sum_{i=1}^{n-1} (y_i)^2[1 + 10sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} + \sum_{i=1}^{n} \mu(x_i, 10,100,4)$	10	[-50,50]	0
$F_{12}(x) = 0.1\left\{sin^2(3\pi x_1) + \sum_{i=1}^{n} (x_i - 1)^2[1 + sin^2(3\pi x_i + 1)] + (x_n - 1)^2[1 + sin^2(2\pi x_n)]\right\} + \sum_{i=1}^{n} \mu(x_i, 5,100,4)$	10	[-50,50]	0
$F_{13}(x) = \sum_{i=1}^{11} [a_i - \frac{x_i(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}]^2$	10	[-5,5]	0

Bu fonksiyonların SFO ile çözümü için kullanılan parametreler ise şu şekilde belirlenmiştir. Problem boyutu 10, nüfus sayısı 100, α=0.25 ve β=2 katsayıları belirlenmiş ve her fonksiyon SFO algoritması ile 30 defa çözülmüştür. Bir sonraki bölümde bu

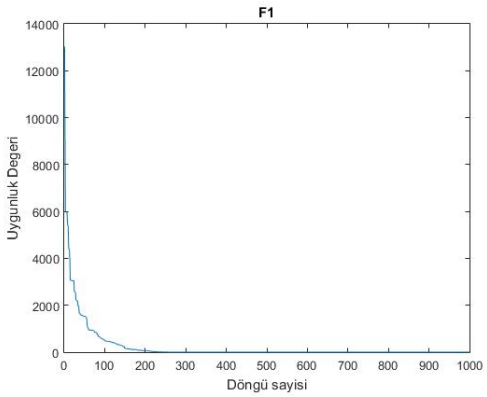
fonksiyonların SFO ile çözümünde elde edilen örnek Döngü Sayısı-Uygunluk Değeri grafikleri ve bazı istatiksel veriler sunulmuştur.

3.2. FONKSİYONLARIN SFO İLE ÇÖZÜMÜ

Bu bölümde tez çalışması kapsamında geliştirilen SFO algoritması onüç tane benchmark fonksiyon için test edilmiştir. Her bir fonksiyon için algoritmanın döngü sayısı – uygunluk değeri grafikleri verilmiştir.

3.2.1. F1 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.14'de SFO algoritmasının F1 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

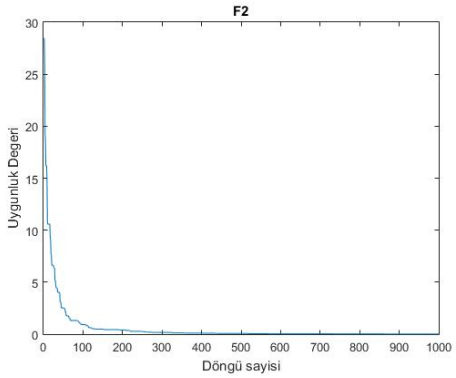


Şekil 3.14. SFO algoritmasının F1 fonksiyonu için elde etmiş olduğu yakınsama grafiği

Grafiğe göre F1 fonksiyonu için algoritmanın global minimuma çok hızlı yakınsadığı ve tüm iterasyonlar bittiğinde 5,18E-37 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

3.2.2. F2 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.15'de SFO algoritmasının F2 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

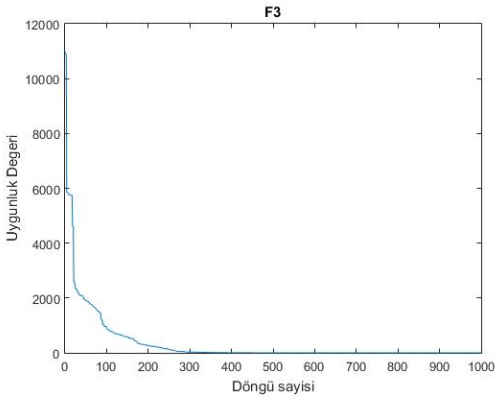


Şekil 3.15. SFO algoritmasının F2 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F2 fonksiyonuna baktığımızda algoritmanın global minimuma hızlı bir şekilde yakınsadığı ve tüm iterasyonlar bittiğinde 1,03E-36 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

3.2.3. F3 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.16'de SFO algoritmasının F3 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

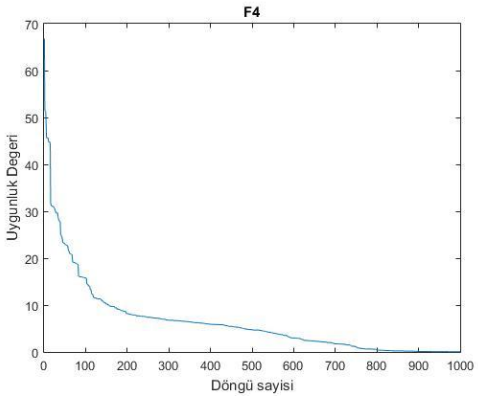


Şekil 3.16. SFO algoritmasının F3 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F3 fonksiyonu üzerinde SFO'nun çalışması incelendiğinde algoritmanın global minimuma hızlı bir şekilde yakınsadığı ve tüm iterasyonlar sonucunda 3,65E-26 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

3.2.4. F4 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.17'de SFO algoritmasının F4 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

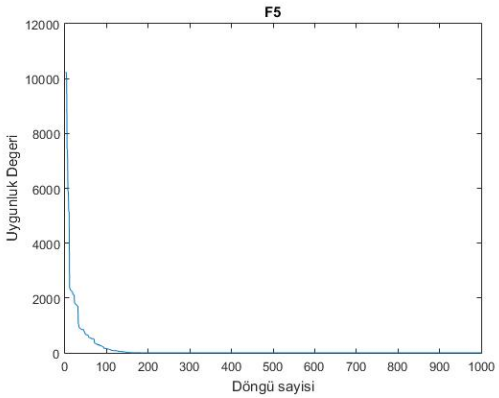


Şekil 3.17. SFO algoritmasının F4 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F4 fonksiyonuna baktığımızda algoritmanın global minimuma iterasyonun ortalarında yakınsamaya başladığı görülmektedir. Tüm iterasyonlar bittiğinde 2,32E-28 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

3.2.5. F5 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.18'de SFO algoritmasının F5 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

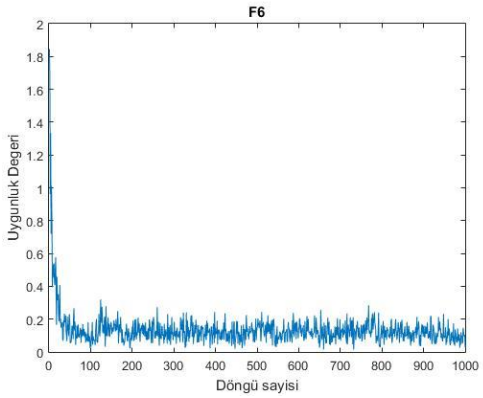


Şekil 3.18. SFO algoritmasının F5 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F5 fonksiyonuna baktığımızda algoritmanın global minimuma çok hızlı yakınsadığı ve tüm iterasyonlar bittiğinde 4,75E-05 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

3.2.6. F6 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.19'de SFO algoritmasının F6 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

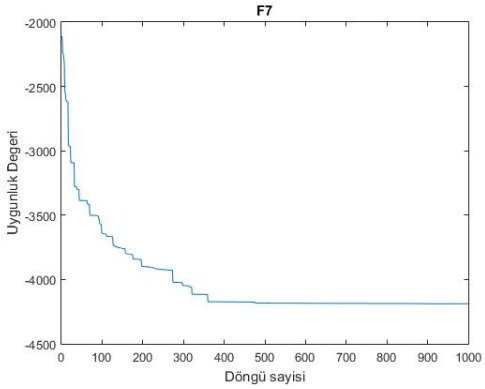


Şekil 3.19. SFO algoritmasının F6 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F6 fonksiyonuna baktığımızda algoritmanın global minimuma hızlı bir şekilde yakınsadığı ancak sonrasında lokal minumum çevresinde aramaya devam ettiği görülmektedir. Tüm iterasyonlar bittiğinde 1,05E-04 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

3.2.7. F7 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.20'de SFO algoritmasının F7 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

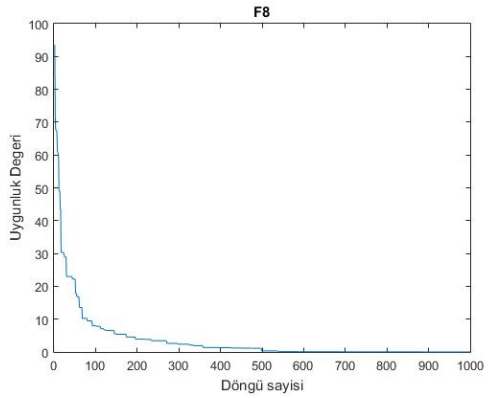


Şekil 3.20. SFO algoritmasının F7 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F7 fonksiyonu üzerinde SFO'nun çalışması incelendiğinde algoritmanın global minimuma hızlı bir şekilde yakınsadığı ve tüm iterasyonlar sonucunda -4,19E+03 değeri ile global minimum olan -418.9829*boyut değerine yaklaştığı görülmektedir.

3.2.8. F8 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.21'de SFO algoritmasının F8 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

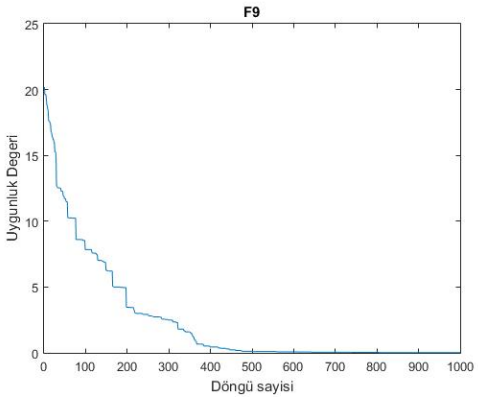


Şekil 3.21. SFO algoritmasının F8 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F8 fonksiyonuna baktığımızda algoritmanın global minimuma çok hızlı yakınsadığı ve tüm iterasyonlar bittiğinde 7,11E-14 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

3.2.9. F9 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.22'de SFO algoritmasının F9 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

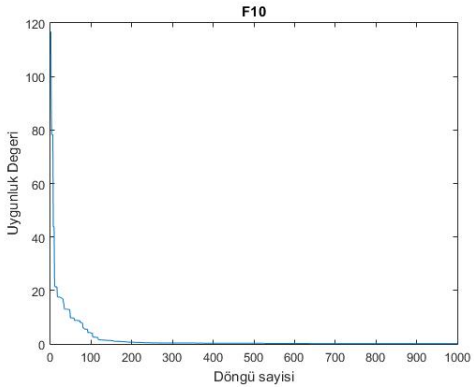


Şekil 3.22. SFO algoritmasının F9 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F9 fonksiyonuna baktığımızda algoritmanın global minimuma iterasyonun ortalarında yakınsamaya başladığı görülmektedir. Tüm iterasyonlar bittiğinde 8,88E-16 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

3.2.10. F10 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.23'de SFO algoritmasının F10 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

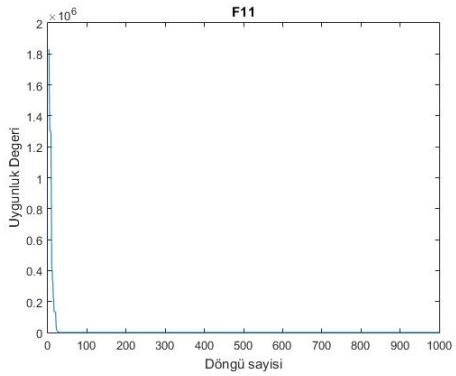


Şekil 3.23. SFO algoritmasının F10 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F10 fonksiyonu üzerinde SFO'nun çalışması incelendiğinde algoritmanın global minimuma hızlı bir şekilde yakınsadığı ve tüm iterasyonlar sonucunda 2,22E-16 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

3.2.11. F11 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.24'de SFO algoritmasının F11 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

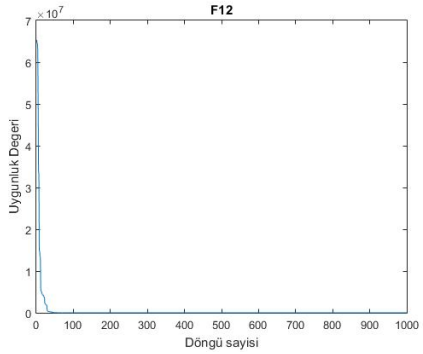


Şekil 3.24. SFO algoritmasının F11 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F11 fonksiyonu üzerinde SFO'nun çalışması incelendiğinde algoritmanın global minimuma çok hızlı bir şekilde yakınsadığı ve tüm iterasyonlar sonucunda 8,56E-06 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

3.2.12. F12 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.25'de SFO algoritmasının F12 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.

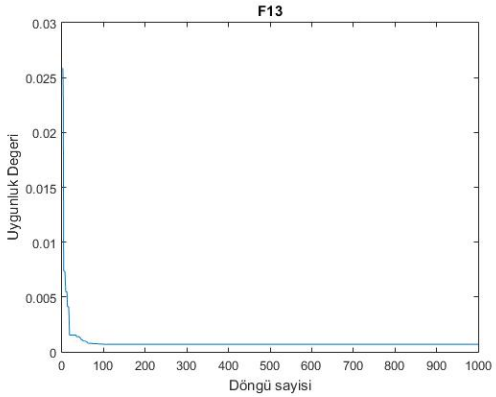


Şekil 3.25. SFO algoritmasının F12 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F12 fonksiyonunda SFO'nun çalışmasına bakıldığında algoritmanın global minimuma çok hızlı bir şekilde yakınsadığı ve tüm iterasyonlar sonucunda 2,27E-05 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

3.2.13. F13 Fonksiyonunun SFO ile Çözümü

Aşağıda Şekil 3.26'de SFO algoritmasının F13 fonksiyonu için yapılan çalıştırmalarından örnek bir tanesinin döngü sayısı-uygunluk değeri grafiği gösterilmiştir.



Şekil 3.26. SFO algoritmasının F13 fonksiyonu için elde etmiş olduğu yakınsama grafiği.

Grafiğe göre F13 fonksiyonuna baktığımızda algoritmanın global minimuma hızlı bir şekilde yakınsadığı ve tüm iterasyonlar bittiğinde 4,35E-04 değeri ile global minimum olan 0 değerine yaklaştığı görülmektedir.

Tüm fonksiyonların 30 defa çalıştırılması sonucunda elde edilen uygunluk değerlerinin minimum, maksimum ve ortalama değerleri aşağıdaki tabloda verilmiştir.

Çizelge 3.2. Fonksiyonların 30 kez çalıştırılması sonucunda 13 benchmark fonksiyon için hesaplanan amaç fonksiyon değerlerine dair istatiksel veriler.

Fonksiyonlar	Minimum	Maximum	Ortalama	Std. Sapma
F1	5,18E-37	2,28E-31	9,62E-33	4,09E-32
F2	1,03E-36	2,55E-33	2,31E-34	4,99E-34
F3	3,65E-26	4,78E-09	1,60E-10	8,58E-10
F4	2,32E-28	1,02E-24	5,50E-26	1,91E-25
F5	4,75E-05	8,59E-04	1,75E-04	1,64E-04
F6	1,50E-04	8,27E-02	2,51E-02	2,06E-02
F7	-4,19E+03*10	-4,17E+03*10	-4,18E+03*10	3,57E+00
F8	7,11E-14	6,58E-03	4,37E-04	1,25E-03
F9	8,88E-16	8,88E-16	8,88E-16	9,86E-32
F10	2,22E-16	3,18E-01	2,27E-02	7,42E-02
F11	8,56E-06	3,07E-04	7,75E-05	6,29E-05
F12	2,27E-05	1,58E-04	8,65E-05	3,56E-05
F13	4,35E-04	8,21E-04	7,11E-04	8,27E-05

Tabloya göre; ortalama değerlere bakıldığında SFO'nun tüm fonksiyonlar için global minumumlara yakınsadığı gözlemlenmiştir. Ayrıca algoritmanın genel olarak standart sapmasının düşük olduğu söylenebilir. Bu da algoritmanın kararlılığının bir göstergesidir. Bir diğer açıdan elde edilen mininmum değerlere bakıldığında ise algoritmanın local minimuma takılma oranının düşük olduğu gözlenmiştir (Şekil 3.19'da görülen F6 fonksiyonu haricinde).

4. SONUÇLAR VE ÖNERİLER

Bu çalışmada doğadan esinlenilen bir optimizasyon algoritması geliştirmek hedefiyle yola çıkılmıştır. Yapılan bir dizi araştırma ve literaratür çalışması sonucunda bir kar tanesinin gökyüzünden yeryüzüne olan yolculuğunun bir optimizasyon algoritması olarak modellenebileceği görülmüş ve tez çalışması bu yönde ilerletilmiştir. Elde edilen sonuçlar açısından bakıldığında tez çalışmasının literatüre katkıları şu şekilde özetlenebilir.

1. Oldukça karmaşık fiziksel süreçler içeren bir doğa olayı olan kar yağışı bir optimizasyon algoritması olarak modellenebilmesi için iki fiziksel kural ile ifade edilmiştir.
2. Bu iki kural rüzgar ve yerçekimi olarak belirlenmiştir.
3. Rüzgar için literatürde power-law kuralı olarak bilinen ve yeryüzündeki farklı seviyelerdeki rüzgar hızlarının önceden belirlenmiş referans bir yükseklikteki ölçülmüş rüzgar hızına göre hesaplayan formül kullanılmıştır. Bu formül ile algoritmanın aday çözümlerinin arama uzayındaki gezinme hareketi modellenmiştir.
4. Yerçekimi ise kuramsal olarak algoritmaya dahil edilmiş ve yerçekimi ile aday çözümlerin uygunluk değeri arasında bir ilişki kurulmuştur. Bu ilişki kullanılarak bu aday çözümler arasındaki bilgi transferi gerçekleştirilmiştir. Bilgi transferi yapılırken kar tanelerinin gökyüzünden yeryüzüne düşerken yabancı cisimlere çarpması durumu modellenmiştir.
5. Bu iki formülasyon kullanılarak adına kar tanesi optimizasyon algoritması denilen yeni bir optimizasyon algoritması geliştirilmiştir.
6. Geliştirilen algoritma literatürde kullanılan 13 farklı benchmark fonksiyon üzerinde test edilmiştir. Elde edilen sonuçlar SFO algoritmasının optimizasyon problemlerinin çözümünde kullanılabileceğini göstermiştir.

Geliştirilen algoritmanın daha iyi ve daha kararlı sonuçlar üretmesi hedefi açısından bakıldığında ileriki çalışmalarda algoritmanın özellikle local minimuma takılma açısından daha da geliştirilebileceği düşünülmektedir. Ayrıca bu algoritmanın gerçek

optimizasyon problemlerinin çözümünde kullanılarak bu açıdan performansının değerlendirilmeside hedeflenmektedir.

5. KAYNAKLAR

[1] S. Kahaner , D., Moler, ve C., Nash, "Numerical methods and software", *Mathematics of Computation*, c. 55, sayı 192, ss. 865–867, 2006.

[2] Platon, *Devlet*, 38. baskı. İstanbul, Türkiye: Türkiye İş Bankası Kültür Yayınları, 2019.

[3] S. Kiranyaz, T. Ince, ve M. Gabbouj, *Multidimensional Particle Swarm Optimization for Machine Learning and Pattern Recognition*, 15. baskı, Berlin, Heidelberg: Springer, 2014.

[4] T. S. Brown, "Herodotus speculates about egypt", *The American Journal of Philology*, c. 86, sayı 1, s. 60, 2006.

[5] M. E. Çolak, "MapReduce ile metasezgisel optimizasyon", Yüksek Lisans tezi, Yazılım Mühendisliği, Fen Bilimleri Enstütüsü, Fırat Üniversitesi, Elazığ, Türkiye, 2015.

[6] L. V. Kantorovich, "Mathematical methods in the organization and planning of production", *Management Science*, c. 6, sayı 4, ss. 366–422, 1960.

[7] S. S. Rao, *Engineering Optimization: Theory and Practice*, 4. baskı, Hoboken, New Jersey: John Wiley & Sons, Inc, 1996.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, ve C. Stein, *Introduction to Algorithms - Selected Solutions*, Boston, MA: The MIT Press, 2009.

[9] D. P. Loucks ve E. V. Beek, "Water resources systems planning & management", *UNESCO*, Prishtina, Kosovo, 2005, ss. 452–457.

[10] B. C. Kress, "Numerical Optimization", *Field Guide to Digital Micro-Optics*, Bellingham WA, USA: Society of Photo-Optical Instrumentation Engineers (SPIE), 2015.

[11] P. Erdoğmuş, "Doğadan esinlenen optimizasyon algoritmaları ve optimizasyon algoritmalarının optimizasyonu", *Düzce Üniversitesi Bilim ve Teknoloji Dergisi*, c. 4, ss. 293-304, 2016.

[12] D. E. Goldberg ve J. H. Holland, "Guest editorial genetic algorithms and machine learning", *Machine Learning*, sayı 3, ss. 95–99, 1988.

[13] K. A. De Jong ve W. M. Spears, "A formal analysis of the role of multi-point crossover in genetic algorithms", *Annals of Mathematics and Artificial Intelligence*, c. 5, sayı 1, ss. 1–26, 1992.

[14] H. G. Beyer ve H. P. Schwefel, "Evolution Strategies - A comprehensive introduction", *Natural Computing*, c. 1, sayı 1, ss. 3–52, 2002.

[15] S. He, Q. H. Wu, ve J. R. Saunders, "Group search optimizer: An optimization algorithm inspired by animal searching behavior", *IEEE Transactions on Evolutionary Computation*, c. 13, sayı 5, ss. 973–990, 2009.

[16] S. Kirkpatrick, C. D. Gelatt, ve M. P. Vecchi, "Optimization by simulated

annealing", *Science*, c. 220, sayı 4598, ss. 671–680, 1983.

[17] E. Rashedi, H. Nezamabadi-pour, ve S. Saryazdi, "GSA: A Gravitational Search Algorithm", *Information Sciences*, c. 179, sayı 13, ss. 2232–2248, 2009.

[18] H. Abedinpourshotorban, S. Mariyam Shamsuddin, Z. Beheshti, ve D. N. A. Jawawi, "Electromagnetic field optimization: A physics-inspired metaheuristic optimization algorithm", *Swarm and Evolutionary Computation*, c. 26, ss. 8–22, 2016.

[19] V. K. Patel ve V. J. Savsani, "Heat transfer search (HTS): A novel optimization algorithm", *Information Sciences*, c. 324, ss. 217–246, 2015.

[20] M. Abdechiri, M. R. Meybodi, ve H. Bahrami, "Gases brownian motion optimization: An algorithm for optimization (GBMO)", *Applied Soft Computing Journal*, c. 13, sayı 5, ss. 2932–2946, 2013.

[21] A. H. Kashan, "An effective algorithm for constrained optimization based on optics inspired optimization (OIO)", *CAD Computer Aided Design*, c. 63, ss. 52–71, 2015.

[22] A. Baykasoğlu ve Ş. Akpinar, "Weighted Superposition Attraction (WSA): A swarm intelligence algorithm for optimization problems – Part 1: Unconstrained optimization", *Applied Soft Computing Journal*, c. 56, ss. 520–540, 2017.

[23] M. Dorigo ve L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem", *IEEE Transactions on Evolutionary Computation*, c. 1, sayı 1, ss. 53–66, 1997.

[24] B. Chopard ve M. Tomassini, "Particle swarm optimization", *Natural Computing Series*, Berlin, Heidelberg: Springer, 2018, ss. 97–102.

[25] I. Rbouh ve A. A. El Imrani, "Hurricane-based Optimization Algorithm", *AASRI Procedia*, c. 6, ss. 26–33, 2014.

[26] M. Ghaemi ve M. R. Feizi-Derakhshi, "Forest optimization algorithm", *Expert Systems with Applications*, c. 41, sayı 15, ss. 6676–6687, 2014.

[27] A. Hatamlou, "Black hole: A new heuristic optimization approach for data clustering", *Information Sciences*, c. 222, ss. 175–184, 2013.

[28] D. Karaboga ve B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm", *Journal of Global Optimization*, c. 39, sayı 3, ss. 459–471, 2007.

[29] W. Pan, "Knowledge-Based Systems A new Fruit Fly Optimization Algorithm : Taking the financial distress model as an example", *Knowledge-Based Systems*, c. 59, ss. 159–160, 2012.

[30] X. S. Yang ve S. Deb, "Cuckoo search via Lévy flights", *2009 World Congress on Nature and Biologically Inspired Computing, NABIC 2009 - Proceedings*, 2009, ss. 210–214.

[31] A. H. Gandomi ve A. H. Alavi, "Krill herd: A new bio-inspired optimization algorithm", *Communications in Nonlinear Science and Numerical Simulation*, c. 17, sayı 12, ss. 4831–4845, 2012.

[32] S. Das, A. Biswas, S. Dasgupta, ve A. Abraham, "Bacterial foraging optimization algorithm: Theoretical foundations, analysis, and applications", *Studies in*

Computational Intelligence, c. 203, ss. 23–55, 2009.

[33] X. S. Yang, "A new metaheuristic Bat-inspired Algorithm", *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*, Berlin, Heidelberg: Springer, 2010, ss. 65–74.

[34] X. S. Yang, "Firefly algorithms for multimodal optimization", *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Berlin, Heidelberg: Springer, 2009, ss. 169–178.

[35] B. R. Rajakumar, "The Lion's Algorithm: A New nature-inspired search algorithm", *Procedia Technology*, c. 6, ss. 126–135, 2012.

[36] S. Mirjalili, S. M. Mirjalili, ve A. Lewis, "Grey Wolf Optimizer", *Advances in Engineering Software*, c. 69, ss. 46–61, 2014.

[37] S. Mirjalili, "The ant lion optimizer", *Advances in Engineering Software*, c. 83, ss. 80–98, 2015.

[38] S. Mirjalili ve A. Lewis, "The Whale Optimization Algorithm", *Advances in Engineering Software*, c. 95, ss. 51–67, 2016.

[39] N. Günal, "Türkiye'de Kar Yağışı, Karın Yerde Kalma Süresi ve Daimi Kar Sınırı", *Acta Turcica*, c. 5, sayı 1, 2013.

[40] H. S. M. Coxeter, J. Kepler, C. Hardie, L. L. Whyte, ve B. F. J. Mason, "The Six-Cornered Snowflake.", *The American Mathematical Monthly*, c. 75, sayı 6, s. 692, 2006.

[41] F. C. Frank, "Early discoverers XXXI: descartes' observations on the amsterdam snowfalls of 4, 5, 6 and 9 february 1635", *Journal of Glaciology*, c. 13, sayı 69, ss. 535–539, 1974.

[42] U. Nakaya, (2019, 04 Mayıs). *The deformation of single crystals of ice.* [Online]. Erişim: http://hydrologie.org/redbooks/a047/04724.pdf.

[43] M.Embaby, K. S. M. Essa, ve M. E. Soad, "A Notional Variation of the wind profile power-law exponent as a function of surface roughness and stability", *Conference on Nuclear and Particle Physics*, 2003.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Özgün AKKOYUN

Doğum Tarihi ve Yeri : 02/12/1992 – Yenimahalle/Ankara

Yabancı Dili : İngilizce / İyi.

E-posta : akkoyun.ozgun@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Lisans	İstatistik	Hacettepe Üniversitesi	2014
Lise		Kayabayazıtoğlu Lisesi	2010

YAYINLAR

Ö. Akkoyun ve M. Toz, "Snow Flake Optimization Algorithm", *26th IEEE Signal Processing and Communications Applications Conference*, İzmir, Türkiye: SIU 2018, 2018.