

İçindekiler

1.Optimizasyon Nedir?	2
2.Meta-Sezgisel Yöntemler	2
2.1 Meta-sezgisel Algoritmalar Giriş	2
2.2 Meta-Sezgisel Yöntemler	2
2.2.1 Tek Çözüme Dayalı Yöntemler	4
2.2.2. Toplum Tabanlı Yöntemler	5
3. Parçacık Sürü Optimizasyonu (PSO).....	5
3.1 Parçacık Sürü Optimizasyonu Tarihi	5
3.2 Parçacık Sürü Optimizasyonu Nedir?	6
3.3 Parçacık Sürü Optimizasyonu Algoritması ve Çalışma Prensibi	5
3.3.1 Atalet	7
3.3.2 Pbest ve Gbest	7
3.3.3 Random	8
3.3.4 C Sabiti	8
3.3.5 Konum	8
3.3.6. Hız.....	8
3.4 Kullanıldığı Alanlar	8
3.5 Parçacık Sürü Optimizasyonu Uygulama Örneği.....	9
4. Kaynakça	14

1.Optimizasyon Nedir?

Optimizasyon, verilen bir problemin olası çözümleri için çözüm uzayındaki en uygun çözümü belirlemeyi hedefleyen bir disiplindir. Başka bir anlatımla problem için verilen durumlar ve belirli kısıtlamalar altında probleme en uygun çözümün belirlenme sürecidir. Burada her problem için durumlar ve kısıtlamalar farklılık göstermektedir. Olası en iyi çözüm problem içerisinde bulunan tüm olası durum ve kısıtlamaları karşılıyorsa uygun(optimal) çözümdür. Geçmişten günümüze gelene kadar değişik alanlarda kullanılmak üzere probleme uygun farklı optimizasyon teknikleri denenmiştir. Bu teknik ve yöntemler; matematiksel algoritmalar ve sezgisel algoritmalar. Matematik algoritmaların kullanılmasında sonucunda algoritmaların esnek olmaması ve çözüm uzayı geniş problemlerde tüm çözüm uzayını gezdiği için maliyetinin fazla olması gibi sebeplerden dolayı farklı yüksek performansa, az maliyete ve genel amaçlı kullanabilecekleri yeni yöntem arayışına gidilmiştir. Bu arayış içerisinde doğadaki düzen ve olaylardan esinlenerek farklı optimizasyon yöntemleri geliştirilmiştir. Bu yöntemlere de genel olarak sezgisel yöntemler denmektedir.

2.Meta-Sezgisel Yöntemler

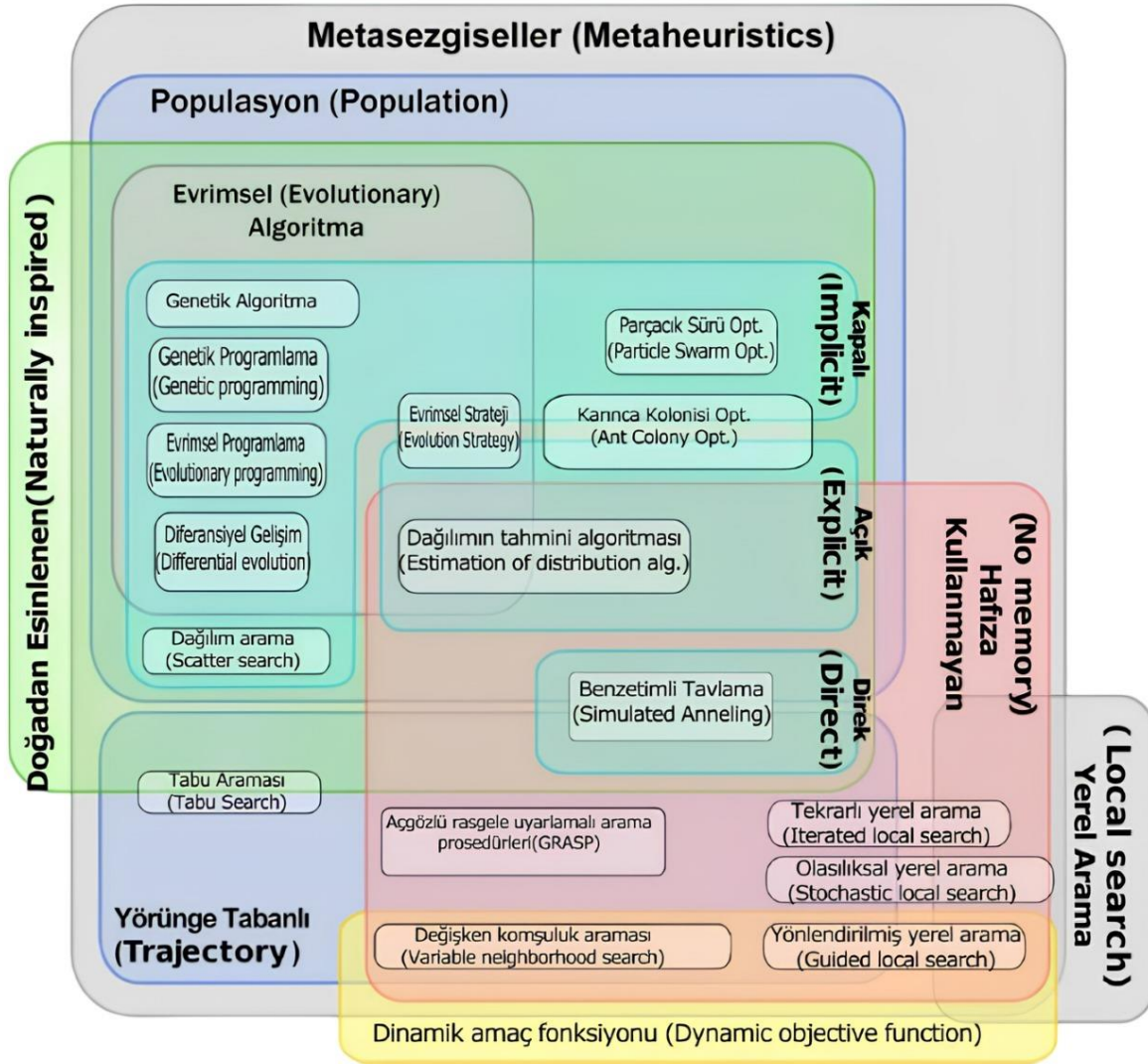
2.1 Meta-sezgisel Algoritmalar Giriş

Optimizasyon algoritmalarının konu edindiği gerçek dünya problemleri karmaşık ve sonuca ulaşılması zor olan problemlerdir. Karşılaşılan bu problemlerde genelde kullanılan algoritmalar çözüm odaklı algoritmalar olduğundan bu tür algoritmaları farklı gerçek dünya problemleri üzerinde kullanılamamaktadır. Bu soruna yönelik daha hızlı çalışan ve farklı problemlere uyarlanabilir algoritmalar geliştirilmiştir. Bu algoritmalar metasezgisel algoritmalar denmektedir. Meta-sezgisel kelimesi, yunanca kökenli bulmak anlamına gelen “heurisken” ve üst seviye yöntembilim anlamına gelen “meta” kelimelerinin birleşimi ile ortaya çıkmıştır. Yani üst seviye sezgisel algoritmalar olarak adlandırabiliriz. Meta-sezgisel yöntemler genellikle doğadan esinlenerek tasarlanmış olup farklı problemlere uyarlanabilir. Bu sayede farklı dünya problemlerine karşın sadece amaç fonksiyonunu değiştirmek yeterlidir. Meta-sezgisel yöntemler sezgisel yöntemlerden yararlanarak çözüm uzayını en verimli şekilde kullanmayı hedefler. Bu yöntemler kesin sonuç veremeselerde karmaşık ve büyük problemler için etkin çözümler ortaya koyabilir. Bu nedenle gerçek dünya problemlerinin çözümünde sezgisel algoritmaların aksine meta-sezgisel algoritmalar daha verimlidir.

2.2 Meta-Sezgisel Yöntemler

Meta-sezgisel yöntemler problemler için en uygun ya da en uyguna yakın çözümlerin bulunabilmesi için çözüm uzayını daha verimli şekilde kullanmayı ve taramayı hedefler. Bu yöntemler içerisinde birçok farklı teknik içermektedir. Bu teknikler yerel arama algoritmaları ve karmaşık öğrenme süreçleri olabilir. Meta-sezgisel algoritmalar tek bir probleme bağlı kalmadığı için farklı problemler üzerine uyarlanabilir.

Birçok farklı metasezgisel yöntem vardır. Bunlara örnek olarak genetik algoritmalar, parçacık sürü optimizasyonu, tabu araması, karınca kolonisi gibi örnek verilebilir. Bu tekniklerin özellikleri incelendiğinde farklı sınıflandırmalar yapılabilir. Meta-sezgisel yöntemlerin sınıflandırılması şekil 1'deki gibidir.

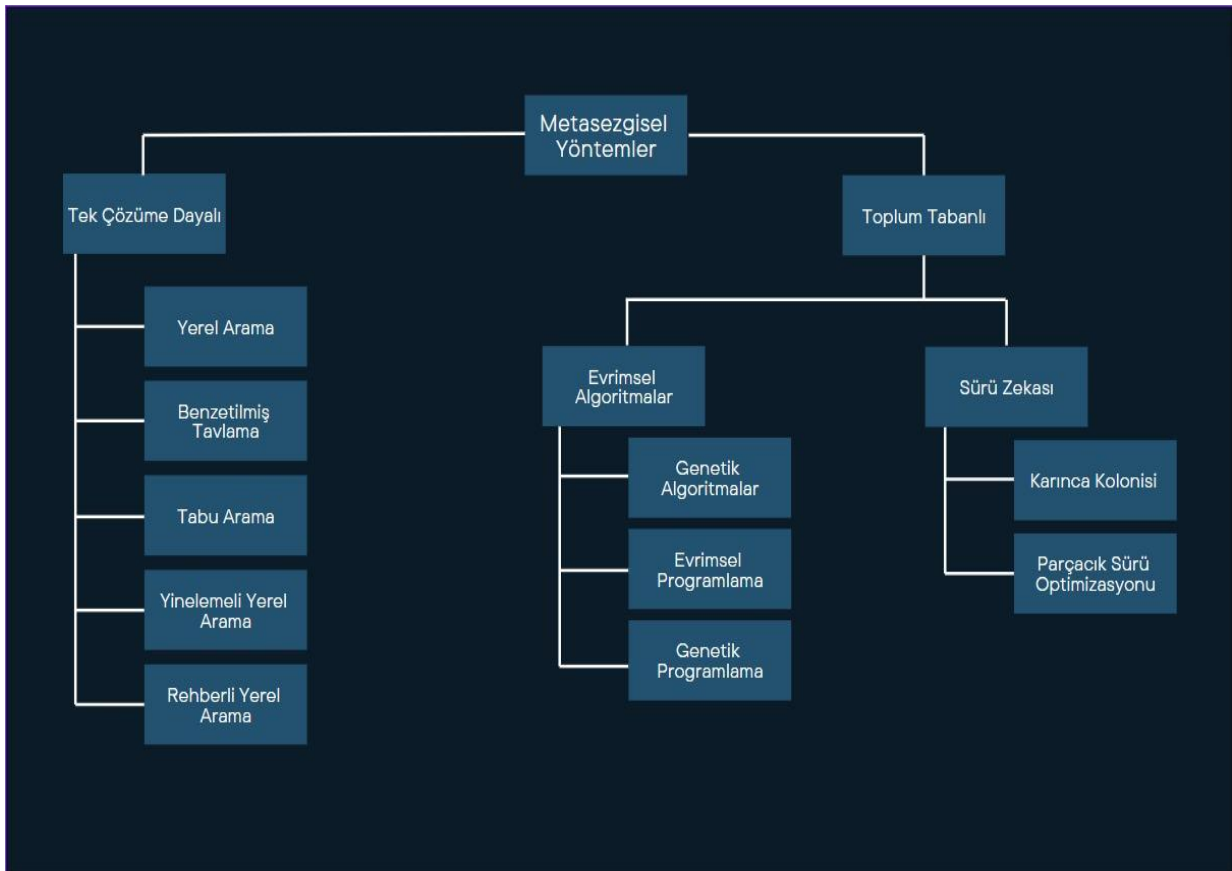


Şekil 1 / Meta-sezgisel Yöntemlerin Sınıflandırılması

Doğadan esinlenilen meta-sezgisel yöntemler, doğada gerçekleşen bir olaydan esinlenerek oluşturulan modeller ile farklı problemlere en uygun çözüm veya çözümleri ortaya koymaya çalışmaktadır. Karınca kolonisi, parçacık sürü optimizasyonu ve genetik algoritmalar doğadan esinlenmiş en bilindik meta-sezgisel yöntemlerdir. Tabu arama ve yinelenmeli yerel arama ise doğadan esinlenmeyen metasezgisel yöntemler arasında yer almaktadır. Dinamik amaç fonksiyonlarında çözüm arayışı içerisinde iken amaç fonksiyonu dinamik olarak değiştirilebilmektedir. Bunun amacı yerel en iyi arama sonuca takılı kalınmadan global en iyi arama sonucunun bulunmasını sağlamaktır. Meta-sezgisel yöntemlerin diğer bir sınıflandırma çeşidi de komşuluk yapılarına göre sınıflandırmadır. Meta-sezgisel yöntemlerin büyük bir kısmı komşuluk yapısına sahip algoritmalar olarak geçmektedir. Bu metasezgisel yöntemler komşuluk yapısını kullanarak aramalarda çeşitlilik kazanılmasını sağlar. Meta-sezgisel

yöntemlerin bir diğer sınıflandırma çeşidi de yöntemlerin hafıza kullanıp kullanmadığına göre yapılan sınıflandırmadır. Tabu araması, parçacık sürü optimizasyonu, genetik algoritma ve karınca kolonisi algoritmaları hafıza kullanan metasezgisel yöntemlerdir. Hafıza kullanılan algoritmalar probleme yönelik çözüm aşamalarında tespit edilen en uygun sonuçların daha sonraki aşamalarda kullanılmak üzere hafızada saklanır. Bu nedenle bu algoritmalara hafıza kullanılan metasezgisel algoritmalar denmektedir.

Farklı sınıflandırmalarda mevcuttur. Şekil 2’de farklı bir sınıflandırmaya ait tablo gösterilmiştir. Bu sınıflandırma esas alınan durum çözüm sayısına yönelik olarak incelenmiştir. Tek çözüme dayalı yöntemler, problem üzerinde ki arama sürecini tek bir çözüm ile gerçekleştirir. Toplum tabanlı algoritmalarda ise her aramada bir çözüm uzayı kullanarak ilerler.



Şekil 2/ Meta-sezgisel Yöntemlerin Çözüm Sayısına Dayalı Sınıflandırılması

2.2.1 Tek Çözüme Dayalı Yöntemler

Tek çözüme dayalı metasezgisel yöntemler, başlangıçta tek bir çözüm ile başlar ve devamında arama uzayında tek çözüme bağlı kalarak diğer olası çözümleri ilk başta oluşturulan tek çözümü referans alarak çözümü araştırır ve uygun olan çözümü seçer. Bu çözüm arayışını genel de ilk başta oluşturulan çözümün komşuluklarını tarayarak gerçekleştirir. Bu yöntemde bazı ortak

kavramlar vardır; komşuluk, yerel optimum ve ilk çözüm. Tek çözüme dayalı meta-sezgisel yöntemlerde ki en önemli faktör komşuluk tanımlanmasıdır. Bu komşuluğun tanımlanması algoritmanın performansına ve maliyetinde önemli bir yer edinmektedir. Komşuluk tanımlanmasının yeterli olarak yapılmadığı algoritmalarda problem ya çözüme ulaştırılamaz ya da çözüm yetersiz kalacaktır. Bu yöntemde komşuluk yapısının büyük seçilmesi çözüm kalitesine olumlu bir etkisi olurken hesaplama süresinde ciddi bir oranda artışa sebep olmaktadır. Çözüm kalitesi ve hesaplama süresi arasında ters orantı vardır.

2.2.2. Toplum Tabanlı Yöntemler

Toplum tabanlı metasezgisel algoritmalar tek çözüme dayalı algoritmaların aksine tek bir çözüme bağlı kalmayıp her yinelemede oluşturduğu çözüm kümesi üzerinden kendini geliştirerek arama yapan algoritmalarlardır. Bu metasezgisel yöntemle oluşturulan arama uzayının daha gerçekçi bir yapıda araştırılmasına olanak tanır. Toplum tabanlı algoritmaların performans ölçütü toplumun veya herhangi bir popülasyonun birbiri arasında ki uyuma bağlıdır. Bu toplum tabanlı yöntemler 2 ye ayrılır. Bunlar; evrimsel algoritmalar ve sürü zekasıdır. Doğadan esinlenen sürü zekâsı içerisinde; Parçacık sürü optimizasyonu, karınca kolonisi gibi algoritmalar yer alırken Genetik algoritmalar, Evrimsel programlama ve genetik programlama algoritmaları da Evrimsel algoritmalar kapsamında incelenmektedir.

3. Parçacık Sürü Optimizasyonu (PSO)

3.1 Parçacık Sürü Optimizasyonu Tarihi

Parçacık sürü optimizasyonunun temel fikri 1987 senesinde Reynolds tarafından yapılmış kuş sürülerinden ilham alınarak benzer parçacıkların modellenmesi üzerine yapılmış olduğu çalışmalar sonucunda ortaya çıkmıştır. Bu yapılan çalışmada sürü de ki her bireyin hareketleri önceki nesilden gelen genlerinde saklı olduğu bilgisine dayanarak bu teknik oluşturulmuştur.

Eberhart ve Kennedy sürü olarak hareket eden hayvan popülasyonlarının yiyecek bulmak veya göç etmek gibi davranışlar içerisindeyken ortaya koydukları hareketlerin aynı popülasyon içerisindeki bireyleri etkilediğinin ve popülasyonu hedefledikleri noktaya ya da yiyeceğe daha kolay ulaşabildiklerini gözlemleyip modellemişlerdir. Eberhart ve Kennedy 1995 yılında dayanarak bu modellemelerin optimizasyon problemlerinin çözümü veya olası çözüm potansiyelini fark edip bu konuya daha da yoğunlaşarak parçacık sürü optimizasyonu tekniğini ortaya koymuştur. Şekil 3'te Esinlenen davranışlar ve algoritmada karşılık gelen eylemler belirtilmiştir. (Medium ve T027025 ve ae2013uğur kuşku)

Esinlenen Davranışlar	Algoritmada Karşılık Gelen Eylem
1. Kuşlara rastgele konumları ve hızları belirlenir. Bu konumların besin kaynaklarına yakınlığı hesap edilir.	1. Rastgele çözüm kümeleri oluştur. Her çözüm kümesi parametre sayısı kadar eleman içerir. Oluşturulan çözüm kümelerinin uygunluk değerleri bulunur.
2. Kuşlar konumlarını sürüde en iyi konuma sahip kuşa ve geçmişteki kendi en iyi konumlarına bir hız belirlerler. Belirlenen hızla birlikte kuş konumunu yeniler.	2. Tüm çözüm kümelerine sırayla iyileştirilme formülü uygulanır.
3. Belirlenen tekrar sayısına ulaşıncaya kadar 2. Aşamaya geri dönlür.	3. Maksimum iterasyona ulaşıncaya kadar 2. Aşamaya geri dönlür.

Şekil 3/ Esinlenen Davranışlar

3.2 Parçacık Sürü Optimizasyonu Nedir?

Parçacık sürü optimizasyonu parçacıklar arası bilgi aktarımı sağlanan toplumsal tabanlı bir metasezgisel algoritmadır. Bu bilgi aktarımını kuş sürülerinin yemek arayışı esnasında sürüye üye olan her kuşun yemek arayışı içinde oldukları alanda rastgele dağılması ve her bir kuşun arama alanında arayışı bittiğinde eş zamanlı olarak bir araya gelerek yiyeceğin nerede olduğu hakkında bilgi paylaşımında bulunurlar. Sürü içerisinde ki kuşlar yemeğin ne kadar uzaklıkta olduğunu ve yemeğe en yakın olan kuşun konumunu belirler. Bu sayede sürüde ki kuşlar da yiyeceğe doğru yönelirler. Bu hareket Parçacık sürü optimizasyonun parçacıklar arası bilgi aktarımını açıklayan bir harekettir. Parçacık sürü optimizasyonu bu senaryo referans olarak oluşturulmuş ve çalışma mantığı bu senaryo üzerinden açıklanmaktadır. (T027025)

3.3 Parçacık Sürü Optimizasyonu Algoritması ve Çalışma Prensibi

Bu teknikte sistem popülasyonda ki parçacıklara rastgele potansiyel çözümler vererek başlatır. Popülasyonda bulunan her parçacık kendinden bir önceki parçacığın deneyiminden faydalanarak kendi pozisyonunu en iyi pozisyona göre ayarlar. Parçacık sürü optimizasyonunda parçacıklar kendi arasında haberleşerek en iyi konumdaki parçacığa yaklaşmayı hedefler. Bu teknikte her parçacık yeni iterasyonda bir önceki kendi konuma göre daha iyi bir konuma gelir ve bu konum değişikliğini en iyi konuma gelene süreci devam ettirir. Şekil 4'te parçacık sürü optimizasyonunun akış şeması verilmiştir.



Şekil 4 / Algoritma Akış Diyagramı

Akış diyagramındaki hız ve konum güncellemesi adımı belirli bir denkleme sahiptir. Bu denklemin matematiksel ifadesi aşağıdaki gibidir.

$$V_{id} = W * V_{id} + c_1 * rand_1 * (P_{id} - X_{id}) + c_2 * rand_2 * (P_{gd} - X_{id})$$

$$X_{id} = X_{id} + V_{id}$$

Belirtilen formülde parçacıkların hareketleri sürü içerisindeki konumlarına(x), hızlarına(v) ve ataletlerine bağlı olarak hesaplanmaktadır. Formüldeki değişkenlerin detaylı açıklamaları, ne işe yaradığı ve performansa etkisi aşağıda açıklanmıştır.

3.3.1 Atalet (W)

Atalet terimi parçacıkların hareket yönünü koruma ve değiştirme derecesini değiştirir. Atalet bir parçacığın hareket durumunun değişmesine karşı gösterdiği direnci ifade eder. Atalet değerinin düşük olması parçacığın rastgele hareket etme eğilimini azaltır. Düşük atalet parçacıkları daha fazla yerel aramaya yönlendirir yani daha önce keşfedilen iyi çözümlere daha fazla yönelir. Atalet değerinin yüksek olması ise rastgele hareket etme eğilimini artırır ve daha fazla global aramaya yönlendirir. Yani daha geniş bir arama uzayına sahip olur.

3.3.2 Pbest ve Gbest

Parçacık sürü optimizasyonunda ki parçacıklar iki farklı hareket eğilimi göstermektedir. Bunlar pbest ve gbest'tir. Gbest değeri sürü içerisindeki tüm parçacıkların en iyi konuma yaklaşma eğilimi olarak açıklanabilir. Pbest ise sürü içerisindeki tek bir parçacığın kendisine ait en iyi konumu koruma eğilimi olarak açıklanabilir.

İlk Pbest değeri arama işlemine başlamadan önce rastgele verilmiş başlangıç konumlarına eşittir. Her iterasyonda bir önceki pbest değeri ile kıyaslanarak yeni pbest değerini günceller. Parçacıklar bu tarama esnasında en iyi pbest değerini bulmayı amaçlar.

Sürü içerisindeki tüm parçacıkların arasından seçilen en iyi pbest değeri en iyi gbest değeri olarak seçilir. Her iterasyonda gbest değeri de kıyaslanarak güncellenir.

3.3.3 Random (rand)

Sürü içerisindeki parçacıkların boyutları parçacıkların problemin sonuçlarını etkilemektedir. Bu nedenle parçacıkların boyutları PSO algoritmasında önemli bir yere sahiptir. Verilecek random boyut değeri $[0,1]$ değer aralığına sahiptir.

3.3.4 C Sabiti

C sabiti sürü içerisindeki parçacıkların konumlarını ve yayıldıkları alanı etkilemektedir. C değerinin düşük olması parçacıkların daha dar alana yayılmasına, c değerinin yüksek olması parçacıkların daha geniş alana yayılmasına neden olur. C değeri problemin durumuna göre $[0,2]$ aralığında alınır. C değerinin yüksek olması verilen problemin çözümünde optimum sonucun verilmesini sağlar.

3.3.5 Konum (x)

Parçacık sürü optimizasyonu algoritmasında sürü içerisinde bulunan parçacıkların konumu başlangıçta rastgele olarak atanmaktadır. Verilecek bu rastgele değerler belirli koşullara bağlıdır ve bunun neticesinde parçacıkların sapması azalmaktadır. Her parçacığın konumu her iterasyonda bir önceki konumlarına ve o anki hızlarına bağlı olarak belirlenmektedir.

3.3.6 Hız (v)

PSO algoritmasındaki parçacıkların o anki hızları bir sonraki adımda yeni oluşturulacak olan parçacıkların konum ve hızının belirlenmesinde önemli bir yere sahiptir. (ae2013ugurkuskusu)

3.4 Kullanıldığı Alanlar

Parçacık sürü optimizasyonu günümüzde farklı alanlarda kullanılmaktadır. Bunlara örnek olarak; Atölye tipi çizelgeleme, yapay sinir ağları, tasarım, çizelgeleme problemleri, veri madenciliği ve biyoenformatik gibi alanlarda kullanılır. Bu problemlere uygun optimum çözümleri verir. Her problem için kesin çözüm verme garantisi yoktur.

3.5 PSO Uygulama Örneği

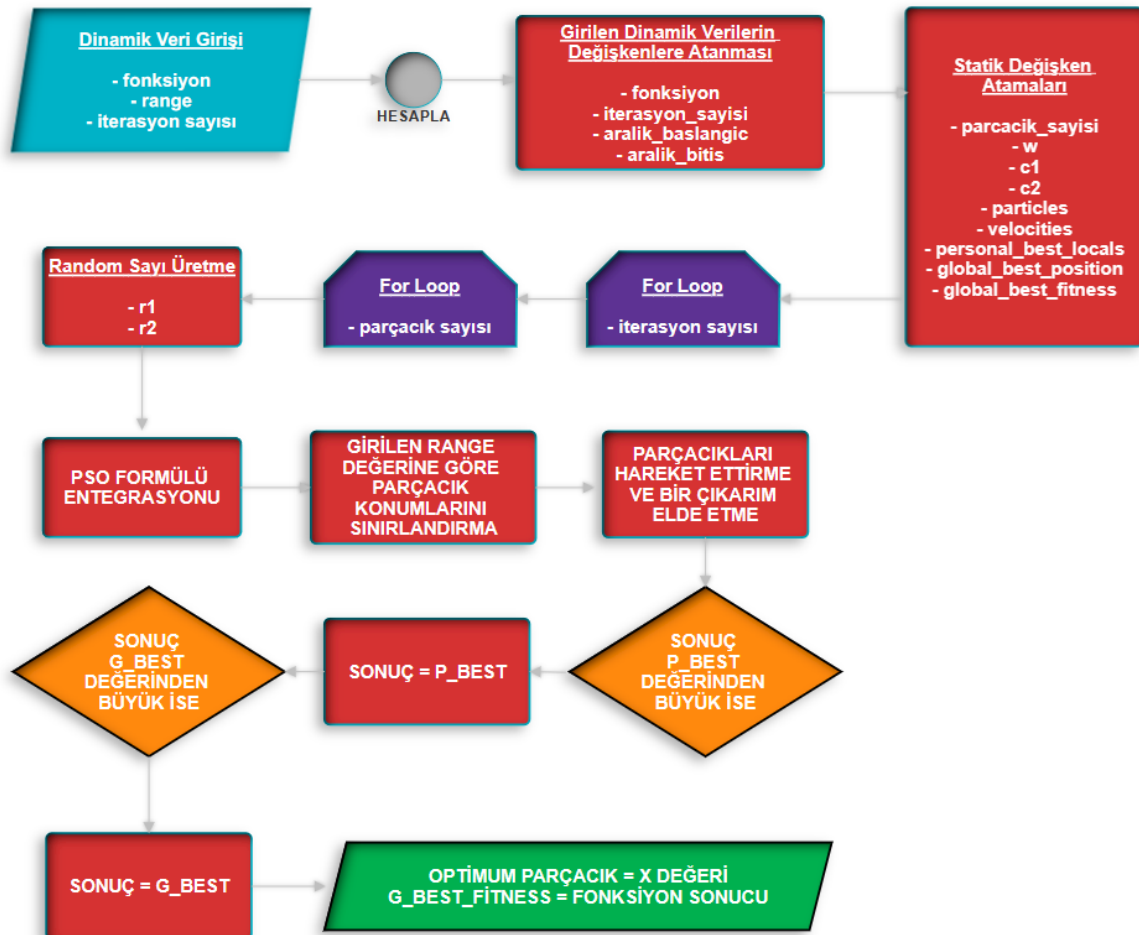
Çalışmanın Amacı: Matematiksel bir fonksiyonun maksimum değerini veren x değerini bulmak

Yöntem: Parçacık sürü optimizasyonu (PSO)

Programlama Dili: Python

Kullanılan Kütüphaneler: tkinter, math, random

Kod Akış Diyagramı



Şekil 5 / PSO Kod Diyagramı

```

import tkinter as tk
import random
import math

def hedef_fonksiyon(x):

    try:
        fonksiyon = textbox_fonksiyon.get()
        result = eval(fonksiyon, {'x': x, 'math': math})
        return result
    except ZeroDivisionError:
        text_islemler.insert("1.0", "Payda 0 olamayacağı için x değeri aralığı 0'dan başlayamaz\n")

def pso():

    parcacik_sayisi = 20
    iterasyon_sayisi = int(textbox_iterasyon.get())
    aralik_baslangic = int(textbox_min.get())
    aralik_bitis = int(textbox_max.get())

    w = 0.5
    c1 = 0.5
    c2 = 0.9

    particles = [random.uniform(aralik_baslangic, aralik_bitis) for _ in range(parcacik_sayisi)]
    velocities = [0 for _ in range(parcacik_sayisi)]
    personal_best_locals = particles[:]
    global_best_position = particles[0]
    global_best_fitness = hedef_fonksiyon(particles[0])
    text_islemler.delete("1.0", tk.END)

    for iterasyon in range(int(iterasyon_sayisi)):
        for i in range(parcacik_sayisi):
            r1 = random.random()
            r2 = random.random()

            velocities[i] = w * velocities[i] + c1 * r1 * (personal_best_locals[i] - particles[i]) + c2 * r2 *
(global_best_position - particles[i])
            particles[i] = particles[i] + velocities[i]

            particles[i] = max(min(particles[i], (aralik_bitis)), (aralik_baslangic))

            fonksiyonun_sonucu = hedef_fonksiyon(particles[i])

            if fonksiyonun_sonucu > hedef_fonksiyon(personal_best_locals[i]):
                personal_best_locals[i] = particles[i]
            if fonksiyonun_sonucu > global_best_fitness:
                global_best_position = particles[i]
                global_best_fitness = fonksiyonun_sonucu

```

```

        text_islemler.insert("1.0", "iterasyon: " + str(iterasyon+1) + "\tFonksiyonun sonucu: " +
str(global_best_fitness) + "\tX degeri: " + str(global_best_position) + "\n")

        textbox_sonuc.delete(0, tk.END)
        textbox_sonuc.insert(0, str(global_best_position))

#GUI
root = tk.Tk()
root.title("PSO ile Matematiksel İfadeyi Maksimum Yapan X Değeri Bulma")

label_fonksiyon = tk.Label(root, text="Matematiksel İfade")
label_fonksiyon.pack()
textbox_fonksiyon = tk.Entry(root, width=50)
textbox_fonksiyon.pack()

label_minmax = tk.Label(root, text="X değeri aralığı")
label_minmax.pack()
textbox_min = tk.Entry(root, width=10)
textbox_min.pack()
textbox_max = tk.Entry(root, width=10)
textbox_max.pack()

label_iterasyon = tk.Label(root, text="İterasyon Sayısı")
label_iterasyon.pack()
textbox_iterasyon = tk.Entry(root, width=10)
textbox_iterasyon.pack()

buton_hesapla = tk.Button(root, text="Hesapla", command=pso)
buton_hesapla.pack()

label_sonuc = tk.Label(root, text="Fonksiyonun En Büyük Değerini Veren X Değeri")
label_sonuc.pack()
textbox_sonuc = tk.Entry(root, width=50)
textbox_sonuc.pack()

text_islemler = tk.Text(root)
text_islemler.pack()

root.mainloop()

```

Program çalıştırıldığında kullanıcıyı basit tasarımlı bir GUI ekranı karşılamaktadır. Bu ekranda kullanıcıdan beklenen girdiler şunlardır:

- Sadece x değerine bağlı matematiksel bir fonksiyon
- X değerinin değerlendirileceği pozitif tam sayı aralığı
- İterasyon sayısı

Bu girdiler uygun formatta kullanıcıdan alınıp hesapla butonuna basıldıktan sonra “fonksiyon, iterasyon_sayisi, aralik_baslangic, aralik_bitis” değerleri tanımlanmış olur. Programın içerisinde tanımlanan statik değişkenler ise şunlardır:

- parcacik_sayisi
- w
- c1
- c2
- particles
- velocities
- personal_best_locals
- global_best_position
- global_best_fitness

Değişken tanımlamaları tamamlandıktan sonra PSO uygulaması için ilk adıma geçilmektedir. Bu adımda iterasyon sayısı kadar bir döngü ve bu döngünün içerisine belirlenen parçacık sayısı kadar bir döngü daha oluşturulmuştur. Bu sayede her bir iterasyonda her bir parçacık üzerinde işlem yapılabilir.

Döngü içerisinde yapılan ilk işlem PSO formülünde kullanılacak r1 ve r2 değerlerinin 0-1 aralığında olacak şekilde rastgele olarak tanımlanmasıdır. Bu aşamadan sonra PSO algoritmasının formülü için gereken tüm değerler tanımlanmış ve PSO formülü yazılmış olur.

Parçacıkların konumunun kullanıcıdan alınan başlangıç ve bitiş değerleri arasında kalması için bir sınırlandırma yapılmaktadır.

Parçacıklar girilen matematiksel ifadede x yerine konularak işleme sokulmaktadır ve işlem sonucu fonksiyonun_sonucu değişkenine atanmaktadır.

Elde edilen sonuç personal_best_locals değeri ile karşılaştırılmaktadır. Koşul bloğunda yapılan bu karşılaştırmada sonuç değeri personal_best_locals değerinden büyük ise yeni personal_best_locals değeri işleme alınan parçacığa eşit olur.

Aynı şekilde sonuç değeri global_best_position değeri ile bir koşul bloğunda karşılaştırılmaktadır. Sonuç değeri global_best_position değerinden de büyükse yeni global_best_position değeri işleme alınan parçacığa eşit olur ve fonksiyonun_sonucu, global_best_fitness değerine atanır.

Bu döngü iterasyon sayısı kadar tekrarlandığında global_best_fitness değeri kullanıcının girdiği fonksiyonun maksimum değerini, global_best_position değeri ise fonksiyonu maksimum yapan x değerini vermektedir.

Şekil 6’da örnek kod çıktısı bulunmaktadır.

PSO ile Matematiksel İfadeyi Maksimum Yapan X Değeri Bulma

Matematiksel İfade

$2/x * \text{math.sin}(x)$

X değeri aralığı

5

450

İterasyon Sayısı

1000

Hesapla

Fonksiyonun En Büyük Değerini Veren X Değeri

7.7252518399607375

iterasyon: 1000 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

iterasyon: 999 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

iterasyon: 998 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

iterasyon: 997 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

iterasyon: 996 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

iterasyon: 995 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

iterasyon: 994 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

iterasyon: 993 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

iterasyon: 992 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

iterasyon: 991 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

iterasyon: 990 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

iterasyon: 989 Fonksiyonun sonucu: 0.2567491070517983 X degeri: 7.7252518399607375

Şekil 6 / PSO Uygulaması Örnek Çıktısı

4. Kaynakça

- 1) Kılıç, G. (2023). Sıra bağımlı ilişkisiz paralel makine çizelgeleme problemi için yeni bir sezgisel algoritma önerisi.
- 2) ÖZSAĞLAM, M. Y., & ÇUNKAŞ, M. (2008). Optimizasyon problemlerinin çözümü için parçacık sürü optimizasyonu algoritması. Politeknik Dergisi, 11(4), 299-305.
- 3) Özsağlam, M. Y. (2009). Parçacık sürü optimizasyonu algoritmasının gezgin satıcı problemine uygulanması ve performansının incelenmesi (Master's thesis, Selçuk Üniversitesi Fen Bilimleri Enstitüsü).
- 4) Demirci, H. (2014). Parçacık sürü optimizasyonu ve çoğalan sürü algoritmasının yüzey geri çatımı probleminde uygulanması ve karşılaştırılması (Doctoral dissertation, Sakarya Üniversitesi (Turkey)).
- 5) ERDOĞMUŞ, P. (2016). Doğadan esinlenen optimizasyon algoritmaları ve optimizasyon algoritmalarının optimizasyonu. Düzce üniversitesi bilim ve teknoloji dergisi, 4(1), 293-304.
- 6) Özer, S. (2017). Eşit olmayan hazırlama süreli tek makine toplam ağırlıklı gecikme problemlerinin parçacık ağırlıklı gecikme problemlerinin parçacık sürü optimizasyonu ile çözümü (Doctoral dissertation, Sakarya Üniversitesi (Turkey)).
- 7) Aytuğ, O. N. A. N. (2013). Metasezgisel yöntemler ve uygulama alanları. Çukurova Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi, 17(2), 113-128.
- 8) ORTAKCI, Y., & Göloğlu, C. (2012). Parçacık sürü optimizasyonu ile küme sayısının belirlenmesi.
- 9) ÖZSAĞLAM, M. Y., & ÇUNKAŞ, M. (2008). Optimizasyon problemlerinin çözümü için parçacık sürü optimizasyonu algoritması. Politeknik Dergisi, 11(4), 299-305.
- 10) Çelik, Y., Yıldız, İ., & Karadeniz, A. T. (2019). Son Üç Yılda Geliştirilen Metasezgisel Algoritmalar Hakkında Kısa Bir İnceleme, Avrupa Bilim ve Teknoloji Dergisi, 463-477.
- 11) <https://calismagruplari.itu.edu.tr/docs/librariesprovider5/default-document-library/ae2013ugurkusu.pdf?sfvrsn=0>
- 12) <https://medium.com/deep-learning-turkiye/par%C3%A7ac%C4%B1k-s%C3%BCr%C3%BC-optimizasyonu-pso-1402d4fe924a>
- 13) <https://biryazilimciningunlugu.wordpress.com/2017/05/16/metasezgisel-algoritmalar/>