T.C

SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BSM 310 – YAPAY ZEKA

Grup üyeleri:
G171210040 Emirhan TUĞTEKİN
G171210088 Ömer Faruk SARIIŞIK
G171210112 Efe YILDIZ

Parçacık Sürü Optimizasyonu

Giriş

Parçacık Sürü Optimizasyonu (PSO), 1995 yılında Kennedy ve Eberhart tarafından kuş ve balık sürülerinin iki boyutlu davranışlarından esinlenilerek geliştirilmiş popülasyon tabanlı bir optimizasyon tekniğidir. Bu iki boyutlu davranışlar; çevrelerine adapte olabilme, zengin yiyecek kaynakları bulabilme ve avcılardan kaçabilme gibi 'bilgi paylaşma' yaklaşımı gerektirmektedir. Bilgi paylaşımı sayesinde sürüler, yiyecek ararken ya da avcıdan kaçarken, hedefe en yakın olan sürü elemanını takip ederler ve kendi hızlarını ve konumlarını en başarılı elemana göre güncellerler.

Bu algoritma temelinde de bir sürü ve sürünün her bir bireyi olan parçacıklar bulunmaktadır. Her bir parçacık kendi pozisyonunu bir önceki tecrübesinden yararlanarak sürüdeki en iyi pozisyona ayarlar. Sürünün o andaki en iyi pozisyonuna sahip bireyine göre diğer parçacıklar hareketlerini günceller. Bu yaklaşma hızı rastgele gelişen bir durumdur ve genelde parçacıklar yeni hareketlerinde bir öncekine nazaran daha iyi bir konuma gelirler. Bu süreç hedefe ulaşana kadar devam eder.

PSO, Genetik Algoritmalar gibi evrimsel hesaplama teknikleri ile birçok benzerlikler gösterir. Sistem random çözümlerden oluşan bir popülasyonla başlatılır ve en iyi çözüm için jenerasyonları güncelleyerek arama yapar. Buna karşın, Genetik Algoritmaların tersine, PSO'da çaprazlama ve mutasyon gibi evrimsel operatörler yoktur. PSO'da parçacık (particles) denilen potansiyel çözümler, mevcut en iyi çözümleri takip ederek problem uzayında gezinirler (uçuş yaparlar).

Genetik Algoritmalarla ile karşılaştırılırsa; PSO'nun avantajı gerçekleştirilmesinin kolay olmasıdır ve ayarlanması gereken çok az parametresi vardır. PSO pek çok alanda başarılı bir şekilde uygulanmıştır. Bunlardan bazıları; fonksiyon optimizasyonu, yapay sinir ağları eğitimi, bulanık sistem kontrolü ve Genetik Algoritmaların uygulanabildiği diğer alanlardır.

Biyolojik sistemlerden esinlenen birçok hesaplama tekniği mevcuttur. Örneğin yapay sinir ağları insan beyninin basitleştirilmiş bir modelidir. Genetik algoritmalar biyolojideki evrimsel süreçten esinlenir. Burada ise ele alınan konu biyolojik sistemlerin farklı bir türü olan sosyal sistemlerdir. Özellikle birbiriyle ve çevresiyle etkileşim içinde olan basit bireylerin birliktelik davranışları incelenmektedir. Bu kavram parçacık zekası olarak isimlendirilir.

Parçacık sürüsü kavramı basitleştirilmiş sosyal sistemin benzetimi olarak oluşturuldu. Asıl amaç bir kuş veya balık sürüsü koreografisinin grafiksel olarak benzetimini yapmaktı. Ancak parçacık sürüsü modelinin bir optimizasyon aracı olarak kullanılabileceği düşünüldü.

Kullanılan Yerler

Bu optimizasyon tekniği Eberhart ve Kennedy'nin ardından birçok optimizasyon probleminin çözümünde de kullanılmıştır. Lineer ve lineer olmayan problemlerin çözümü, atış problemleri, çok bilinmeyenli denklemlerin köklerini bulma ve endüstriyel

problemlerin çözümü bunlardan bazılarıdır. Performans açısından bakıldığında diğer optimizasyon tekniklerine göre daha avantajlıdır.

Algoritma

PSO'da her bir çözüm arama uzayındaki bir kuştur ve bunlar bir "parçacık" olarak isimlendirilir. Tüm parçacıkların, optimize edilecek uygunluk fonksiyonu tarafından değerlendirilen bir uygunluk değeri ve uçuşlarını yönlendiren hız bilgileri vardır. Parçacıklar problem uzayında mevcut optimum parçacıkları takip ederek uçarlar.

PSO bir grup rasgele üretilmiş çözümle (parçacıkla) başlatılır ve jenerasyonlar güncellenerek en uygun değer araştırılır. Her iterasyonda, her bir parçacık iki "en iyi" değere göre güncellenir. Bunlardan birincisi bir parçacığın o ana kadar bulduğu en iyi uygunluk değeridir. Ayrıca bu değer daha sonra kullanılmak üzere hafıza tutulur ve "pbest" yani parçacığın en iyi değeri olarak isimlendirilir. Diğer en iyi değer ise popülasyondaki herhangi bir parçacık tarafından o ana kadar elde edilmiş en iyi uygunluk değerine sahip çözümdür. Bu değer popülasyon için global en iyi değerdir ve "gbest" olarak isimlendirilir.

Şimdi parçacıkların değişim hızlarını hesapladığımız formüle bir göz atalım. Parçacık Sürü Optimizasyonunda parçacıkların değişim hızları;

x: Parçacık değeri

v: Parçacığın değişim hızı

c1,c2: sabit değerler

rand1, rand2: Rastgele üretilen değerler,

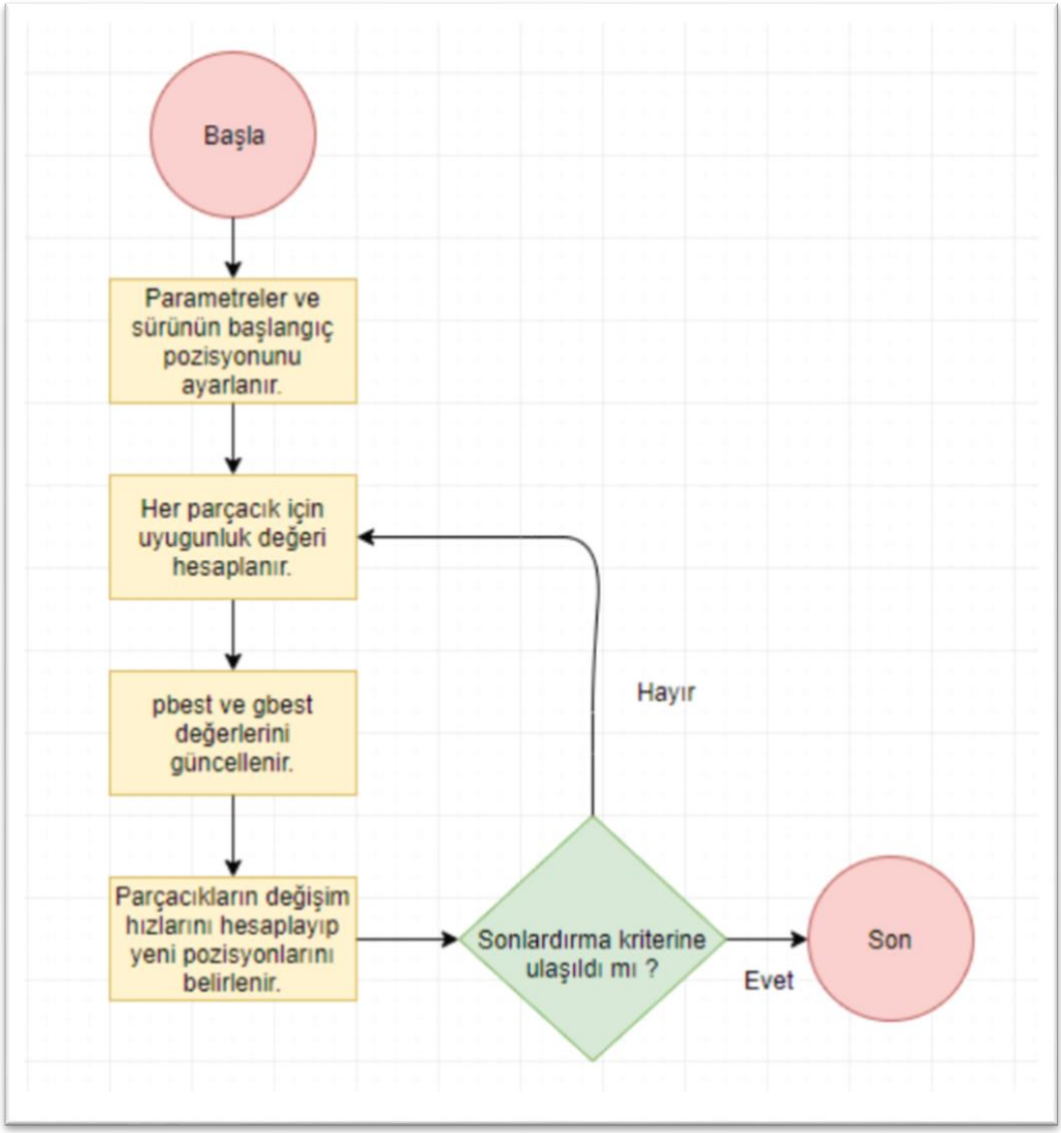
pbest: Parçacığın çözüme en çok yaklaştığı durum. (**pbest** terimi İngilizcede "*Personel Best*" kısaltmasıdır. Türkçede "*kişisel en iyi*" anlamına gelir.)

gbest: Tüm parçacıklar arasında çözüme en çok yaklaşılan durum (**gbest** terimi İngilizcede *"Global Best"* ifadesinin kısaltmasıdır. Türkçede *"küresel en iyi"* anlamına gelir.) olmak üzere aşağıdaki formül ile hesaplanır.

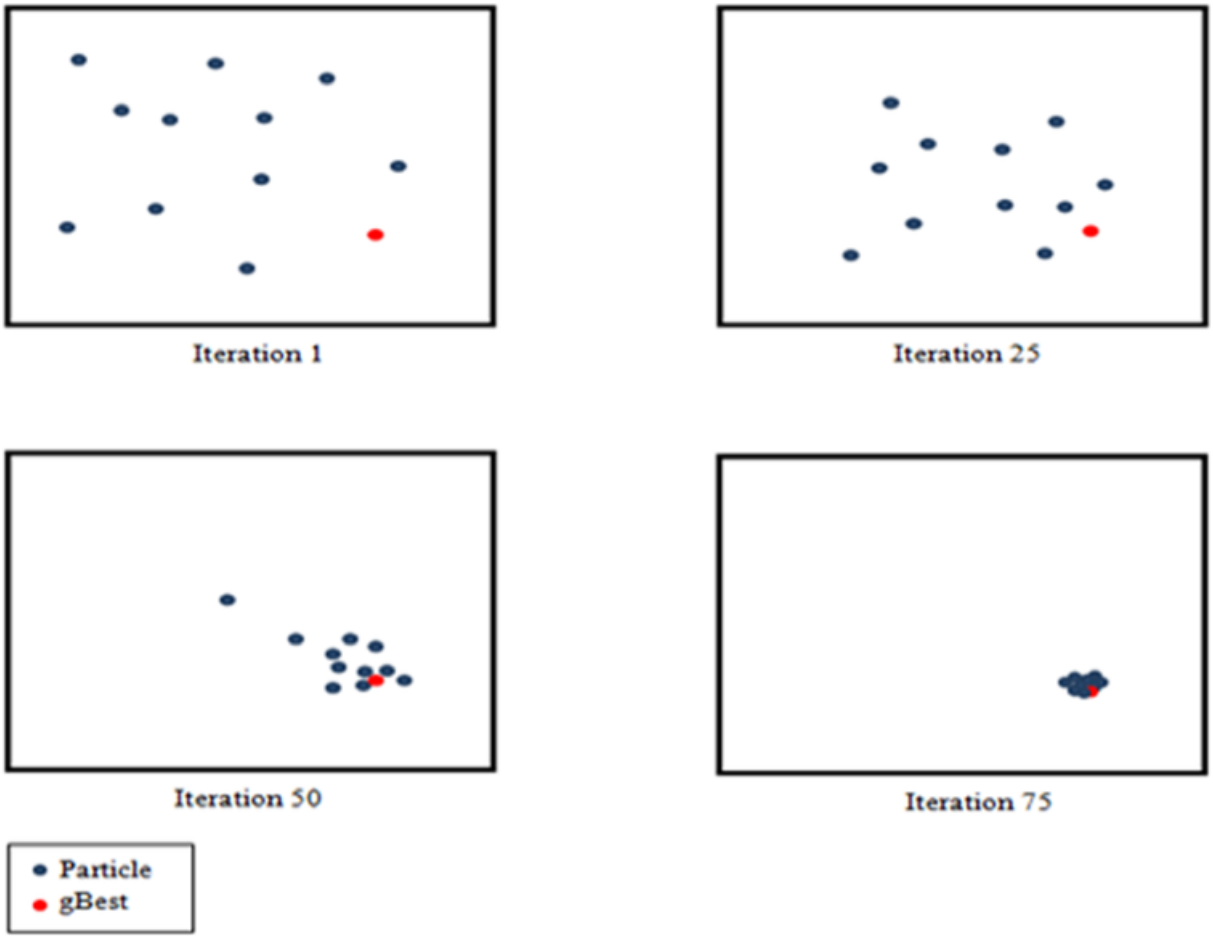
$$v_{i+1} = v_i + c_1 * rand_1 * (pbest - x) + c_2 * rand_2 * (gbest - x)$$

Bu formül sayesinde parçacık kendi en iyi çözümüne ve global en iyi çözüme yönelir. Bu da parçacığı çözümü en iyi parçacığın ve kendi en iyi durumunun yakınlarında aramaya iter.

Algoritmaya bir akış diyagramı üzerinden bakacak olursak;



PSO'da öncelikle çözümü arayacak sürü ve gerekli parametreler belirlenir. Uygunluk fonksiyonu yardımıyla parçacıkların çözüme yakınlığı ölçülür ve bu değerlere göre pbest ve gbest değerleri güncellenir. Daha sonra değişim hızı fonksiyonu ile her parçacığın yapacağı hareket belirlenir ve yeni durumları ayarlanır. Tekrar uygunluk fonksiyonu ile çözüme ne kadar yaklaşıldığı kontrol edilir. Bu döngü istenilen şartlara ulaşılıncaya kadar tekrarlanır.



Her iterasyon sonunda parçacıkların konumları.

Sözde Kod

```
For her parçacık için
        Parçacığı başlangıç konumuna getir
End

Do
    For her parçacık için
        Uygunluk değerini hesapla
        Eğer uygunluk değeri pbest ten daha iyi ise,
            Şimdiki değeri yeni pbest olarak ayarla
    End

    Tüm parçacıkların bulduğu pbest değerlerinin en iyisini, tüm parçacıkların gbest'i olarak ayarla

    For her parçacık için
        Denklem (1)'e göre parçacık hızını hesapla
        Denklem (2)'ye göre parçacık konumunu güncelle
    End
While maksimum  iterasyon sayısına veya minimum hata koşulu sağlanana kadar devem et.
```

Örnek Problem

- ***f(x)= x2 +2x -3*** verilen denklemin sonucunu 0 yapacak **x** değerini PSO ile bulunuz.

Çözüm:

1.Adım: Öncelikle kaç adet parçacık ile çözümü arayacağımızı belirliyoruz.

Bu sayı arama uzayının genişliğine ve sizin isteğinize bağlı olarak belirlenir. Biz bu örneğimizde 3 belirleyelim.

Parçacıklar rastgele belirlenir. Parçacıkları aşağıda verildiği gibi sırasıyla 3, 7, 5 olarak belirledik.

$$P_1 = 3$$

$$P_2 = 7$$

$$P_3 = 5$$

2.Adım: ***Her parçacığın uygunluk değeri hesaplanır.***

Bu örnekte amacımız problemin denklemini 0'a yaklaştırmak olduğundan uygunluk fonksiyonumuz problemin denkleminin ta kendisidir.

$f(x) = x_2 + 2x - 3$

1. $f(3) = 3_2 + 2.3 - 3 = 9 + 6 - 3 =$ **12**
2. $f(7) = 7_2 + 2.7 - 3 = 49 + 14 - 3 =$ **60**
3. $f(5) = 5_2 + 2.5 - 3 = 25 + 10 - 3 =$ **32**

$$f(3) = 12$$

$$f(7) = 60$$

$$f(5) = 32$$

3.Adım: ***pbest ve gbest değerleri hesaplanır.***

Şu an ilk iterasyonda olduğumuzdan parçacıkların kendileri zaten pbest'lerdir. gbest ise 0'a en yakın olan **P1** parçacığıdır.

4.Adım: ***c1, c2 değerleri ve rastgele olarak rand1, rand2 değerleri belirlenir.***

c1 ve c2 değerleri genellikle 2 belirlenir ben de bu parametreleri 2 olarak belirliyorum.

rand1 ve *rand2* değerini de hesaplama kolaylığı açısından 2 belirliyorum.

Ayrıca daha ilk iterasyonda olduğumuzdan parçacıklar herhangi bir hıza sahip değildir. Bu nedenle ilk değişim hızı *V0*'ı 0 kabul ediyoruz.

5.Adım: ***Parçacıkların değişim hızları hesaplanır.***

$v_{i+1} = v_i + c_1 * rand_1 *(pbest – x) + c_2 * rand_2 * (gbest – x)$

$$P_1 \rightarrow 0 + 2 * 2 * (3 - 3) + 2 * 2 * (3 - 3) = 0$$

$$P_2 \rightarrow 0 + 2 * 2 * (7 - 7) + 2 * 2 * (3 - 7) = -16$$

$$P_3 \rightarrow 0 + 2 * 2 * (5 - 5) + 2 * 2 * (3 - 5) = -8$$

Böylece formülümüzü kullanarak parçacıkların değişimi hesaplanmış oldu.

6.Adım: ***Parçacıkların yeni değerleri belirlenir.***

$P_1 = 3$
$P_2 = 7$ → $P_1 \rightarrow 0+2*2*(3-3)+2*2*(3-3)=0$, $P_2 \rightarrow 0+2*2*(7-7)+2*2*(3-7)=-16$, $P_3 \rightarrow 0+2*2*(5-5)+2*2*(3-5)=-8$ →
$P_3 = 5$

$P_1 \rightarrow 3 + 0 = 3$
$P_2 \rightarrow 7 - 16 = -9$
$P_3 \rightarrow 5 - 8 = -3$

7.Adım: ***Yeni parçacıkların uygunluk değerleri bulunur.***

$P_1 \rightarrow 3 + 0 = 3$
$P_2 \rightarrow 7 - 16 = -9$
$P_3 \rightarrow 5 - 8 = -3$

→

f(x) = x_2 + 2x – 3

1. f(3) = 3_2 + 2.3 – 3 = 9 + 6 – 3 = **12**
2. f(-9) = -9_2 + 2.-9 – 3 = 81 – 18 – 3 = **60**
3. f(5) = -3_2 + 2.-3 – 3 = 9 – 6 – 3 = **0**

$f(3) = 12$
$f(-9) = 60$
$f(-3) = 0$

Denklemimizi 0 yapan değer ***-3*** olarak bulunmuş oldu.

Eğer çözüme ulaşamamış olsaydık; yeni parçacıklar da göz ününe alınarak yeniden ***pbest*** değeri ve bu zamana kadar gelmiş tüm ***pbest'***lerin en iyisi olan ***gbest*** değeri belirlenecekti. Ek olarak **rand1** ve ***rand2*** değerleri tekrar belirlenip parçacıkların değişimi hesaplanacak ve yeni parçacıklar bulunacaktı. Ve bir kez daha uygunluk değerini bulup çözüme ne kadar yaklaştığımızı değerlendirecektik.

Sonuç

PSO algoritmasını baz istasyonların en az maliyetle en uygun yerlere yerleştirilmesi, tasarımsal ürünlerin en verimli halinin belirlenmesi gibi herhangi bir optimizasyon problemine uygularken genellikle kaç iterasyon çalışacağını belirlemeniz gerekir.

Örneğin; *“1000 iterasyon çalış ve bana bulduğun en iyi çözümü ver”* diyebilirsiniz. Aksi halde tam çözümü bulana kadar aramaya devam edecektir.

Algoritmanın avantaj ve dezavantajlarından bahsetmek gerekirse; hatırlayacağınız üzere **gbest** olan parçacığın değişim hızı 0 bulunmuştu. Bu durum **PSO**’nun dezavantajlarından biridir. Çözüme en yakın olan parçacık belki de kolayca çözüme ulaşabilecek iken adeta diğer parçacıkların onu yakalayıp geçmesini beklemektedir.

Ayrıca ***PSO*** algoritmasında parçacıklar en iyiyi takip etme eğiliminde olduğundan dolayı, bir süre sonra tüm parçacıkları bir yerde kümelenmiş halde çözümü arıyor olarak bulabilirsiniz. Bu durum çözümün arandığı uzayın detaylı taranması adına olumsuz bir durum oluşturur. Fakat kümelenme geçekten çözümün bulunduğu yerde ise çözüme de daha çabuk ulaşılacağından bir avantaj anlamına gelir.

Kaynak

1) http://web.firat.edu.tr/iaydin/bmu579/bmu_579_bolum3.pdf
2) https://medium.com/deep-learning-turkiye/parçacık-sürü-optimizasyonu-pso-1402d4fe924a
3) http://www.emo.org.tr/ekler/235f3310045b155_ek.pdf
4) https://calismagruplari.itu.edu.tr/docs/librariesprovider5/default-document-library/ae2013ugurkusku.pdf?sfvrsn=0
5) https://en.wikipedia.org/wiki/Particle_swarm_optimization